

SUBJECT CODE : CS8392

Strictly as per Revised Syllabus of
ANNA UNIVERSITY
Choice Based Credit System (CBCS)
Semester - III (CSE / IT)
Semester - V (EEE)
Semester - V (ECE) Professional Elective - I

OBJECT ORIENTED PROGRAMMING

Anuradha A. Puntambekar

M.E. (Computer)

Formerly Assistant Professor in
P.E.S. Modern College of Engineering, Pune.



OBJECT ORIENTED PROGRAMMING

Subject Code : CS8392

Semester - III (CSE / IT)

Semester - V (EEE)

Semester - V (ECE) Professional Elective - I

First Edition : June 2018

Second Revised Edition : June 2019

Third Revised Edition : June 2020

© Copyright with Author

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders

Sr.No. 10/1A,

Ghule Industrial Estate, Nanded Village Road,

Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-332-1924-2



9 789333 219242

AU 17

PREFACE

The importance of **Object Oriented Programming** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Object Oriented Programming**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
A. A. Puntambekar

Dedicated to God.

SYLLABUS

Object Oriented Programming - CS8392

UNIT - I Introduction to OOP and Java Fundamentals

Object Oriented Programming - Abstraction - objects and classes - Encapsulation - Inheritance - Polymorphism - OOP in Java - Characteristics of Java - The Java Environment - Java Source File - Structure - Compilation. Fundamental Programming Structures in Java - Defining classes in Java - Constructors, methods - access specifiers - static members - Comments, Data Types, Variables, Operators, Control Flow, Arrays, Packages - JavaDoc comments. **(Chapter - 1)**

UNIT - II Inheritance and Interfaces

Inheritance - Super classes - sub classes - Protected members - constructors in sub classes - the Object class - abstract classes and methods - final methods and classes - Interfaces - defining an interface, implementing interface, differences between classes and interfaces and extending interfaces - Object cloning - inner classes, Array Lists, Strings. **(Chapters - 2, 3)**

UNIT - III Exception Handling and I/O

Exceptions - exception hierarchy - throwing and catching exceptions - built-in exceptions, creating own exceptions, Stack Trace Elements. Input / Output Basics - Streams - Byte streams and Character streams - Reading and Writing Console - Reading and Writing Files. **(Chapters - 4, 5)**

UNIT - IV Multithreading and Generic Programming

Differences between multi-threading and multitasking, thread life cycle, creating threads, synchronizing threads, inter-thread communication, daemon threads, thread groups. Generic Programming - Generic classes - generic methods - Bounded Types - Restrictions and Limitations. **(Chapters - 6, 7)**

UNIT - V Event Driven Programming

Graphics programming - Frame - Components - working with 2D shapes - Using color, fonts and images - Basics of event handling - event handlers - adapter classes - actions - mouse events - AWT event hierarchy - Introduction to Swing - layout management - Swing Components - Text Fields, Text Areas - Buttons - Check Boxes - Radio Buttons - Lists - choices - scrollbars - Windows - Menus - Dialog Boxes. **(Chapters - 8, 9, 10)**

TABLE OF CONTENTS

UNIT - I

Chapter - 1 Introduction to OOP and Java Fundamentals

(1 - 1) to (1 - 108)

1.1 Object Oriented Programming	1 - 2
1.1.1 Abstraction	1 - 2
1.1.2 Object	1 - 2
1.1.3 Classes	1 - 3
1.1.4 Encapsulation	1 - 4
1.1.5 Inheritance	1 - 5
1.1.6 Polymorphism	1 - 6
1.2 Benefits and Drawbacks of OOP	1 - 6
1.3 Applications of OOP	1 - 7
1.4 OOP in Java	1 - 8
1.4.1 Characteristics of Java	1 - 8
1.4.2 Difference between C, C++ and Java	1 - 10
1.5 The Java Environment	1 - 12
1.5.1 Java Development Kit	1 - 12
1.5.2 Application Programming Interface	1 - 13
1.5.3 Java Runtime Environment	1 - 14
1.5.4 Architecture of JVM	1 - 14
1.6 Structure of Java Program	1 - 16
1.6.1 Writing Simple Java Program	1 - 16
1.7 Fundamental Programming Structures in Java	1 - 18
1.7.1 Java Tokens	1 - 18
1.7.2 Constants	1 - 21
1.7.3 Variables	1 - 22
1.7.4 Data Types	1 - 22

1.7.5 Operators	1 - 25
1.7.6 Expressions.....	1 - 28
1.8 Control Statements.....	1 - 28
1.9 The break and continue Statement	1 - 43
1.10 Defining Classes in Java	1 - 45
1.10.1 Methods Defined in Class	1 - 47
1.10.1.1 Parameter Passing	1 - 48
1.11 Constructors	1 - 52
1.11.1 Properties of Constructor	1 - 55
1.12 Access Specifiers.....	1 - 67
1.13 Static Members.....	1 - 69
1.14 Arrays.....	1 - 71
1.14.1 One Dimensional Array	1 - 71
1.14.2 Two Dimensional Arrays	1 - 73
1.15 Packages	1 - 88
1.15.1 Defining Package	1 - 88
1.15.2 Creating and Accessing Package.....	1 - 89
1.15.3 CLASSPATH.....	1 - 90
1.15.4 Importing Package.....	1 - 91
1.16 JavaDoc Comments.....	1 - 97
1.16.1 Class Level Comments	1 - 97
1.16.2 Member Level Comments.....	1 - 97
Two Marks Questions with Answers	1 - 99

UNIT - II

Chapter - 2	Inheritance	(2 - 1) to (2 - 40)
2.1 Inheritance.....		2 - 2
2.2 Concept of Super and Sub Classes		2 - 2
2.3 Types of Inheritance		2 - 3

2.4 Protected Members.....	2 - 4
2.5 Implementation of Different Types of Inheritance.....	2 - 4
2.5.1 Single Inheritance	2 - 4
2.5.2 Multilevel Inheritance	2 - 7
2.6 Constructors in Sub Classes	2 - 16
2.7 Method Overloading and Method Overriding.....	2 - 18
2.7.1 Overriding.	2 - 18
2.7.2 Methods Overloading	2 - 20
2.8 Use of keyword Super.....	2 - 22
2.9 Object Class	2 - 26
2.9.1 toString Method of Object Class.	2 - 26
2.9.2 equals Method of Object Class	2 - 27
2.10 Abstract Classes and Methods.....	2 - 28
2.10.1 Difference between Abstract Class and Concrete Class	2 - 30
2.11 Final Methods and Classes.....	2 - 34
2.11.1 Final Variables and Methods	2 - 34
2.11.2 Final Classes to Stop Inheritance	2 - 35
2.12 Finalize() Method.....	2 - 35
Two Marks Questions with Answers	2 - 36

Chapter - 3	Interfaces	(3 - 1) to (3 - 36)
3.1 Defining an Interface	3 - 2	
3.1.1 Difference between Class and Interface	3 - 2	
3.1.2 Difference between Abstract Class and Interface.	3 - 2	
3.2 Implementing Interface	3 - 3	
3.3 Applying Interfaces	3 - 5	
3.3.1 Nested Interface	3 - 13	
3.4 Variables in Interface.....	3 - 15	
3.5 Extending Interface.....	3 - 16	
3.6 Multiple Inheritance	3 - 17	

3.7 Object Cloning	3 - 18
3.8 Inner Classes	3 - 21
3.9 ArrayList	3 - 23
3.10 Strings	3 - 25
3.10.1 Finding Length of String	3 - 27
3.10.2 Character Extraction	3 - 28
3.10.3 String Comparison	3 - 29
3.10.4 Searching for the Substring	3 - 30
3.10.5 Replacing the Character from String	3 - 30
3.10.6 Upper and Lower Case	3 - 31
3.10.7 Concatenating Strings	3 - 31
Two Marks Questions with Answers	3 - 33

UNIT - III

Chapter - 4	Exception Handling	(4 - 1) to (4 - 24)
4.1 Exceptions.....		4 - 2
4.2 Exception Hierarchy.....		4 - 3
4.3 Types of Exception.....		4 - 4
4.4 Throwing and Catching Exceptions.....		4 - 5
4.4.1 try-catch Block		4 - 5
4.4.2 Uncaught Exception		4 - 6
4.4.3 Nested try Statements		4 - 9
4.4.4 Multiple Catch		4 - 10
4.4.5 Using finally		4 - 12
4.4.6 Using throws		4 - 13
4.4.7 Using throw		4 - 14
4.5 Built in Exceptions		4 - 16
4.6 Creating Own Exceptions.....		4 - 16
4.7 Stack Trace Elements.....		4 - 20
Two Marks Questions with Answers		4 - 21

Chapter - 5 Input and Output (5 - 1) to (5 - 32)

5.1 Streams.....	5 - 2
5.2 Byte Streams and Character Streams	5 - 2
5.2.1 Byte Stream	5 - 2
5.2.2 Character Stream.....	5 - 3
5.2.3 Comparison between Byte Stream and Character Stream.	5 - 5
5.3 Reading and Writing Console	5 - 5
5.3.1 Reading Console Input.	5 - 5
5.3.2 Writing Console Output	5 - 11
5.4 Reading and Writing Files	5 - 11
5.4.1 FileInputStream / FileOutputStream	5 - 13
5.4.2 FilterInputStream / FilterOutputStream	5 - 16
5.4.2.1 DataInputStream / DataOutputStream	5 - 16
5.4.2.2 BufferedInputStream / BufferedOutputStream	5 - 18
5.4.3 Programming Example on Reading and Writing Files.....	5 - 20
Two Marks Questions with Answers	5 - 29

UNIT - IV

Chapter - 6 Multithreading (6 - 1) to (6 - 32)

6.1 What is Thread ?.....	6 - 2
6.2 Difference between Multithreading and Multitasking.....	6 - 2
6.3 Thread Life Cycle.....	6 - 3
6.4 Creating Threads.....	6 - 4
6.4.1 Extending Thread Class	6 - 5
6.4.2 Implementing Runnable Interface	6 - 7
6.5 Creating Multiple Threads	6 - 10
6.6 Synchronizing Threads.....	6 - 15
6.7 Inter-thread Communication.....	6 - 20
6.7.1 Producer Consumer Pattern	6 - 23

6.8 Daemon Threads.....	6 - 26
6.9 Thread Groups	6 - 28
Two Marks Questions with Answers	6 - 29

Chapter - 7	Generic Programming	(7 - 1) to (7 - 14)
--------------------	----------------------------	----------------------------

7.1 What is Generic Programming ?	7 - 2
7.2 Generic Methods	7 - 2
7.3 Generic Classes	7 - 5
7.4 Bounded Types	7 - 10
7.5 Restrictions and Limitations	7 - 11
Two Marks Questions with Answers	7 - 11

UNIT - V

Chapter - 8	Graphics Programming	(8 - 1) to (8 - 44)
--------------------	-----------------------------	----------------------------

8.1 Introduction to Graphics Programming.....	8 - 2
8.2 Frame.....	8 - 3
8.3 Components	8 - 5
8.3.1 Labels	8 - 5
8.3.2 Buttons	8 - 6
8.3.3 Canvas	8 - 8
8.3.4 Scrollbars	8 - 9
8.3.5 Text Components.....	8 - 10
8.3.6 Checkbox	8 - 11
8.3.7 Checkbox Group.....	8 - 13
8.3.8 Choices	8 - 14
8.3.9 List Panels.....	8 - 15
8.4 Working with 2D Shapes.....	8 - 16
8.4.1 Lines	8 - 17
8.4.2 Rectangle	8 - 18

8.4.3 Oval	8 - 20
8.4.4 Arc	8 - 21
8.4.5 Polygons	8 - 24
8.5 Applet	8 - 26
8.6 Life Cycle of Applet	8 - 26
8.7 Executing an Applet	8 - 28
8.8 Using Color in Applet	8 - 30
8.9 Using Fonts in Applet	8 - 36
8.10 Using Images in Applet	8 - 40
Two Marks Questions with Answers	8 - 41

Chapter - 9	Event Handling	(9 - 1) to (9 - 20)
--------------------	-----------------------	----------------------------

9.1 Basics of Event Handling	9 - 2
9.1.1 Event	9 - 2
9.1.2 Event Delegation Model	9 - 2
9.1.2.1 Example : Handling Button Click	9 - 3
9.2 Event handlers	9 - 4
9.2.1 Event Classes	9 - 4
9.2.2 Event Listeners	9 - 8
9.3 Adapter Classes	9 - 11
9.4 Actions	9 - 14
9.5 Mouse Events	9 - 14
9.6 AWT Event Hierarchy	9 - 17

Two Marks Questions with Answers	9 - 18
---	---------------

Chapter - 10	Programming with Swing	(10 - 1) to (10 - 62)
---------------------	-------------------------------	------------------------------

10.1 Introduction to Swing	10 - 2
10.1.1 Difference between AWT and Swing	10 - 2
10.1.2 Limitations of AWT	10 - 2
10.1.3 Swing Components	10 - 3

10.1.3.1 Swing Class Component Hierarchy	10 - 3
10.2 Layout Management.....	10 - 5
10.2.1 FlowLayout	10 - 5
10.2.2 BorderLayout	10 - 7
10.2.3 GridLayout	10 - 11
10.2.4 CardLayout	10 - 13
10.2.5 GridBagLayout	10 - 17
10.3 Creating Frames.....	10 - 20
10.4 Using Swing Components	10 - 25
10.4.1 Label and Image Icon	10 - 25
10.5 Text Field	10 - 27
10.6 Text Area.....	10 - 28
10.7 Buttons	10 - 30
10.8 Checkboxes.....	10 - 32
10.9 Radio Buttons	10 - 36
10.10 Lists.....	10 - 38
10.11 Choices.....	10 - 39
10.12 Scrollbars	10 - 43
10.13 Menus.....	10 - 48
10.14 Dialog boxes.....	10 - 56
Two Marks Questions with Answers	10 - 61

Object Oriented Programming Laboratory

(L - 1) to (L - 44)

Solved AU Question Papers

(S - 1) to (S - 8)

December - 2018.....	(S - 1) to (S - 2)
May - 2019	(S - 3) to (S - 5)
December - 2019.....	(S - 6) to (S - 8)

UNIT-I

1

Introduction to OOP and Java Fundamentals

Syllabus

Object Oriented Programming - Abstraction - objects and classes - Encapsulation - Inheritance - Polymorphism - OOP in Java - Characteristics of Java - The Java Environment - Java Source File - Structure - Compilation. Fundamental Programming Structures in Java - Defining classes in Java - Constructors, methods - access specifiers - static members - Comments, Data Types, Variables, Operators, Control Flow, Arrays, Packages - JavaDoc comments.

Contents

1.1	Object Oriented Programming	Dec.-11,13,18, May-12,14, . Marks 16
1.2	Benefits and Drawbacks of OOP	
1.3	Applications of OOP	
1.4	OOP in Java	May-19, Dec.-18,19, Marks 7
1.5	The Java Environment	Dec.-19,..... Marks 5
1.6	Structure of Java Program	
1.7	Fundamental Programming Structures in Java.....	May-19, Marks 6
1.8	Control Statements	May-19, Marks 7
1.9	The break and continue Statement	
1.10	Defining Classes in Java	IT : May-12,
	CSE : Dec.-10, 14, 18 Marks 6
1.11	Constructors.....	May-14, Dec.-14,18 Marks 16
1.12	Access Specifiers.....	IT : Dec.-10, 11,
	CSE : Dec.-12,..... Marks 8
1.13	Static Members	CSE : Dec.-10,
	IT : May-11, 12, 13,..... Marks 8
1.14	Arrays.....	May-14, 19, Dec.-19 Marks 16
1.15	Packages	IT : May-12, 13, 14, Dec.-11, 12,
	CSE : Dec.-13,..... Marks 16
1.16	JavaDoc Comments	

1.1 Object Oriented Programming

AU : Dec.-11, 13, 18, May-12, 14, Marks 16

- In object oriented programming approach there is a **collection of objects**. Each object consists of corresponding data structures and the required operations (procedures). This is basically the **bottom up problem** solving approach.
- The object oriented programming approach is developed in order to remove some of the flaws of procedural programming.
- OOP has a complete focus on data and not on the procedures. In OOP, the data can be protected from accidental modification.
- In OOP the most important entity called **object** is created.
- Each object consists of **data** attributes and the **methods**. The methods or **functions** operate on **data** attributes.
- There are various characteristics of object oriented programming and those are -
 - (1) Abstraction
 - (2) Object and Classes
 - (3) Encapsulation
 - (4) Inheritance and
 - (5) Polymorphism.

1.1.1 Abstraction

- **Definition** : Abstraction means representing only essential features by hiding all the implementation details. In object oriented programming languages like C++, or Java class is an entity used for data abstraction purpose.

- **Example**

```
class Student
{
    int roll;
    char name [10];
    public;
    void input ( );
    void display ( );
}
```

In main function we can access the functionalities using object. For instance

```
Student obj;
obj.input ( );
obj.display ( );
```

Thus only abstract representation can be presented, using class.

1.1.2 Object

- Object is an instance of a class.

- Objects are basic run-time entities in object oriented programming.
- In C++ the class variables are called objects. Using objects we can access the member variable and member function of a class.
- Object represent a person, place or any item that a program handles.
- For example - If the class is country then the objects can be India, China, Japan, U.S.A and so on.
- A single class can create any number of objects.

- **Declaring objects -**

The syntax for declaring object is -

```
Class_Name Object_Name;
```

- Example

```
Fruit f1;
```

For the class **Fruit** the object **f1** can be created.

1.1.3 Classes

- A class can be defined as an entity in which data and functions are put together.
- The concept of class is similar to the concept of **structure** in C.
- **Syntax of class** is as given below

```
class name_of_class
{
    private :
        variables declarations;
        function declarations;
    public :
        variable declarations;
        function declarations;
}; do not forget semicolon
```

- Example

```
class rectangle
{
    private :
        int len, br;
    public :
        void get_data ( );
        void area( );
        void print_data ( );
};
```

- **Explanation**

- The class declared in above example is **rectangle**.
- The class name must be preceded by the keyword **class**.
- Inside the body of the class there are two keywords used **private** and **public**. These are called **access specifiers**.

Sr. No.	Class	Object
1.	For a single class there can be any number of objects. For example - If we define the class as River then Ganga, Yamuna, Narmada can be the objects of the class River.	There are many objects that can be created from one class. These objects make use of the methods and attributes defined by the belonging class.
2.	The scope of the class is persistent throughout the program.	The objects can be created and destroyed as per the requirements.
3.	The class can not be initialized with some property values.	We can assign some property values to the objects.
4.	A class has unique name.	Various objects having different names can be created for the same class.

1.1.4 Encapsulation

- Encapsulation is for the detailed implementation of a component which can be hidden from rest of the system.
- In C++ the data is encapsulated.
- **Definition :** Encapsulation means binding of data and method together in a **single entity called class**.
- The data inside that class is accessible by the function in the same class. It is normally not accessible from the outside of the component.

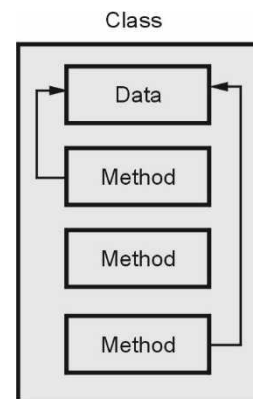


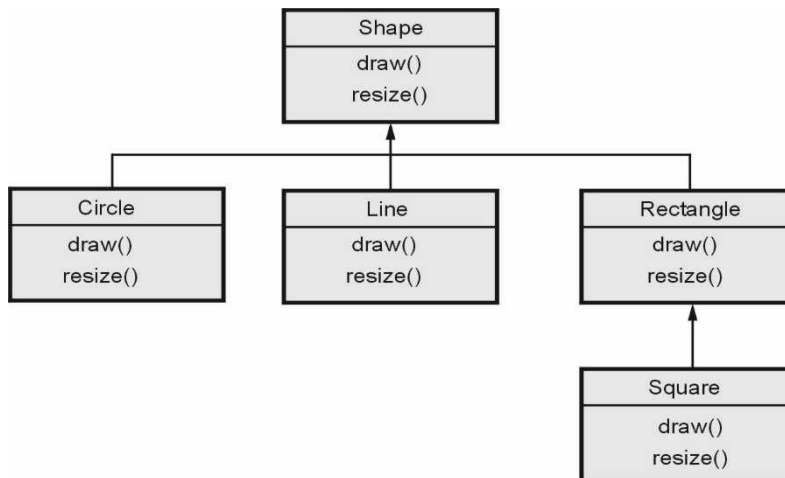
Fig. 1.1.1 Concept of encapsulation

Difference between Encapsulation and Abstraction

Sr. No.	Data encapsulation	Data abstraction
1.	It is a process of binding data members of a class to the member functions of that class.	It is the process of eliminating unimportant details of a class. In this process only important properties are highlighted.
2.	Data encapsulation depends upon object data type.	Data abstraction is independent upon object data type.
3.	It is used in software implementation phase.	It is used in software design phase.
4.	Data encapsulation can be achieved by inheritance.	Data abstraction is represented by using abstract classes.

1.1.5 Inheritance

- **Definition :** Inheritance is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.
- Inheritance support hierarchical structure.
- The old classes are referred as **base classes** and the new classes are referred as **derived classes**. That means the derived classes inherit the properties (data and functions) of base class.
- **Example :** Here the **Shape** is a base class from which the **Circle**, **Line** and **Rectangle** are the derived classes. These classes inherit the functionality **draw()** and **resize()**. Similarly the **Rectangle** is a base class for the derived class **Square**. Along with the derived properties the derived class can have its own properties. For example the class **Circle** may have the function like **backgrcolor()** for defining the back ground color.

**Fig. 1.1.2 Hierarchical structure of inheritance**

1.1.6 Polymorphism

- Polymorphism means **many structures**.
- Definition** : Polymorphism is the ability to take more than one form and refers to an operation exhibiting different behavior in different instances (situations).
- The behavior depends on the type of data used in the operation. It plays an important role in allowing objects with different internal structures to share the same external interface.
- Without polymorphism, one has to create separate module names for each method.
- For example the method **clean** is used to clean a **dish** object, one that cleans a **car** object, and one that cleans a **vegetable** object.
- With polymorphism, you create a single "clean" method and apply it for different objects.

Sr. No.	Inheritance	Polymorphism
1.	Inheritance is a property in which some of the properties and methods of base class can be derived by the derived class.	Polymorphism is ability for an object to used different forms. The name of the function remains the same but it can perform different tasks.
2.	Various types of inheritance can be single inheritance, multiple inheritance, multilevel inheritance and hybrid inheritance.	Various types of polymorphism are compile time polymorphism and run time polymorphism. In compile time polymorphism there are two types of overloading possible. - Functional overloading and operator overloading. In run time polymorphism there is a use of virtual function.

Review Questions

1. Explain what is OOPs and explain features of OOPs.

AU : Dec.-11, Marks 8

2. Discuss the following: (i) inheritance (ii) Polymorphism.

AU : May-12, Marks 16

3. Elaborate on the various object oriented concepts with necessary illustrations.

AU : May-14, Marks 16, Dec.-13, Marks 5

4. Explain the characteristics of OOPs.

AU : Dec.18, Marks 6

1.2 Benefits and Drawbacks of OOP

Benefits

Following are some advantages of object oriented programming -

1. Using **inheritance** the redundant code can be eliminated and the existing classes can be used.
2. The standard working **modules** can be created using object oriented programming. These modules can then communicate to each other to accomplish certain task.

3. Due to **data hiding** property, important data can be kept away from unauthorized access.
4. It is possible to create **multiple objects** for a given class.
5. For **upgrading the system** from small scale to large scale is possible due to object oriented feature.
6. Due to **data centered nature** of object oriented programming most of the details of the application model can be captured.
7. **Message passing technique** in object oriented programming allows the objects to communicate to the external systems.
8. **Partitioning the code** for simplicity, understanding and debugging is possible due to object oriented and modular approach.

Drawbacks

Following are some drawbacks of OOP -

1. The object oriented programming is **complex to implement**, because every entity in it is an object. We can access the methods and attributes of particular class using the object of that class.
2. If some of the members are declared as **private** then those **members** are **not accessible** by the object of another class. In such a case you have to make use of inheritance property.
3. In Object oriented programming, every thing must be arranged in the forms of **classes and modules**. For the lower level applications it is not desirable feature.

1.3 Applications of OOP

Various applications in which object oriented programming is preferred are -

- Business logic applications
- Knowledge based and expert systems
- Web based applications
- Real time systems
- Simulation and modeling
- Object oriented databases
- Applications in distributed environment
- CAD/CAM systems
- Office automation system
- Games programming

1.4 OOP in Java

AU : Dec.-18, 19, May-19, Marks 7

- Java was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems, Inc. in 1991. It took 18 months to complete the first working version of Java.
- This language was initially called as **Oak** but was renamed as Java in 1995.
- Java is purely an object oriented programming language as it supports various features like Classes, Objects, abstraction, inheritance and polymorphism.

1.4.1 Characteristics of Java

Java is getting popular because of its features. Following is a list of buzzwords that makes Java a popular programming language.

- Simple
- Secure
- Platform independent
- Object oriented
- Distributed
- Portable
- High performance
- Robust
- Multithreaded
- Interpreted
- Dynamic

Let us discuss these features in detail.

1. Java is simple and small programming language

- Java is a simple programming language. Even though you have no programming background you can learn this language comfortably.
- The programmers who worked on C or C++ can learn this language more efficiently because the syntax of Java resembles with C and C++.
- In fact Java is made more clear and adaptable than C++.

2. Java is robust and secure

- Following are reasons that make Java more secured than other programming languages -
 1. Java does not support the concept of pointers directly. This makes it impossible to accidentally reference memory that belongs to other programs or the kernel.

2. The output of Java compiler is not executable code but it is a bytecode. The instructions of bytecode is executed by the Java Virtual Machine(JVM). That means JVM converts the bytecode to machine readable form. JVM understand only the bytecode and therefore any infectious code can not be executed by JVM. No virus can infect bytecode. Hence it is difficult to trap the internet and network based applications by the hackers.
- 3 In programming languages like C or C++ the memory management is done explicitly by the user. That means user allocates or deallocates the memory. Whereas in java its automatically done using garbage collection. Thus user can not perform the memory management directly.
4. If an applet is executing in some browser then it is not allowed to access the file system of local machine.

3. Java is a platform independent and portable programming language

- Platform independence is the most exciting feature of Java program. That means programs in Java can be executed on variety of platforms. This feature is based on the goal - *write once, run anywhere, and at anytime forever*.
- Java supports portability in 2 ways - Java compiler generates the byte code which can be further used to obtain the corresponding machine code. Secondly the primitive data types used in Java are machine independent.

4. Java is known as object oriented programming language

- Java is popularly recognised as an object oriented programming language.
- It supports various object oriented features such as data encapsulation, inheritance, polymorphism and dynamic binding.
- Everything in Java is an object. The object oriented model of Java is simple and can be extended easily.

5. Java is multithreaded and interactive language

- Java supports multithreaded programming which allows a programmer to write such a program that can perform many tasks simultaneously.
- This ultimately helps the developer to develop more interactive code.

6. Java can be compiled and interpreted

- Normally programming languages can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted.
- First, Java compiler translates the Java source program into a special code called bytecode. Then Java interpreter interprets this bytecode to obtain the equivalent machine code.
- This machine code is then directly executed to obtain the output.

7. Java is known for its high performance, scalability, monitoring and manageability

- Due to the use of bytecode the Java has high performance. The use of multi-threading also helps to improve the performance of the Java.
- The J2SE helps to increase the scalability in Java. For monitoring and management Java has large number of Application Programming Interfaces(API).
- There are tools available for monitoring and tracking the information at the application level.

8. Java is a dynamic and extensible language

- This language is capable of dynamically linking new class libraries, methods and objects.
- Java also supports the functions written in C and C++. These functions are called **native methods**.

9. Java is a designed for distributed systems

- This feature is very much useful in networking environment.
- In Java, two different objects on different computers can communicate with each other.
- This can be achieved by **Remote Method Invocation(RMI)**. This feature is very much useful in Client-Server communication.

10. Java can be developed with ease

There are various features of Java such as Generics, static import, annotations and so on which help the Java programmer to create a **error free reusable code**.

Review Questions

1. Why is Java known as platform independent language ?
2. Explain how is java more secured than other languages ?
3. What makes java a robust language ? Explain.
4. Explain following features of java
a) Security b) Roboustness c) Platform independence.
5. Explain the features and characteristics of Java.
6. Discuss the three OOP principles in detail.
7. Explain the various features of java in detail.

AU : Dec.-18, Marks 7**AU : May-19, Marks 7****AU : Dec.-19, Marks 5****1.4.2 Difference between C, C++ and Java**

Sr.No.	C	C++	Java
1.	C is a language that needs to be compiled .	The C++ is a language that needs to be compiled .	The Java is a language that gets interpreted and compiled .
2.	C is platform dependent .	C++ is platform dependent .	Java is a platform independent .

3.	C does not support multi-threading programming.	C++ does not support multi-threading programming.	Java supports multi-threading .
4.	C does not have facility to create and implement the graphical user interface.	C++ does not have facility to create and implement the graphical user interface.	Using Java, one can design very interactive and user friendly graphical user interface .
5.	Database handling using C is very complex. The C does not allow the database connectivity.	Database handling using C++ is very complex. The C++ does not allow the database connectivity.	Java servlets can be created which can interact with the databases like MS-ACCESS, Oracle, My-SQL and so on.
6.	C code can not be embedded in any scripting language.	C++ code can not be embedded in any scripting language.	Java code can be embedded within a scripting language by means of applet programming.
7.	C does not have any exception handling mechanism.	C++ has exception handling mechanism using try-catch block.	Java has exception handling mechanism using try-catch and throw .
8.	C does not supports any object oriented programming concept. It is a procedure oriented programming language.	C++ supports multiple inheritance . It is an object oriented programming language .	Java does not support multiple inheritance however it makes use of interface. It is an object oriented programming language .
9.	In C we can use the pointers .	In C++ we can use the pointers .	In Java there is no concept of pointers .
10.	C does not support templates.	C++ supports templates .	Java does not support the concept of templates .
11.	In C there is no class. The structure is used instead of that.	In C++ we can write a program without a class.	In Java, there must be at least one class present.
12.	C is simple to use and implement.	C++ is simple to use and implement.	Java is safe and more reliable .
13.	C can be compiled with variety of compilers.	C++ can be compiled with variety of compilers.	Java can be compiled using an unique compiler.

Review Questions

1. Distinguish between C and Java.
2. Distinguish between C++ and Java.

1.5 The Java Environment

AU : Dec.-19, Marks 5

- The Java environment is made up of three things-development tools, classes and methods and java runtime environment.
- The java development tools constitute **Java Development Kit(JDK)**.
- The classes and methods are called **Application Programming Interface(API)** which forms a library called **Java Standard Library(JSL)**
- The java runtime environment is supported by **Java Virtual Machine (JVM)**.

1.5.1 Java Development Kit

- The Java Development Kit is nothing but a collection of tools that are used for development and runtime programs.
- It consists of -
 1. **javac** - The Java compiler which translates the source code to the bytecode form and stores it in a separate class file.
 2. **java** - The Java interpreter, which interprets the bytecode stored in the class file and executes the program to generate output.
 3. **javadoc** - For creating the HTML document for documentation from source code file.
 4. **javah** - It produces header files for the use of native methods.
 5. **jdb** - The Java debugger which helps to find the errors in the program.
 6. **appletviewer** - For executing the Java applet.
 7. **javap** - Java disassembler, which helps in converting byte code files into program description.
- Following are the steps that illustrate **execution process of the application program** -
 1. The user creates the Java source code in the text editor.
 2. The source code is compiled using the **javac** command.
 3. The **javadoc** tool can be used to create the HTML files that document the source program.
 4. On compiling the source program a class file gets generated which consists of the byte code.
 5. The developer may use **javah** tool for generating the required header files.
 6. The class file produced by **javac** can be interpreted using **java** in order to produce an executable.

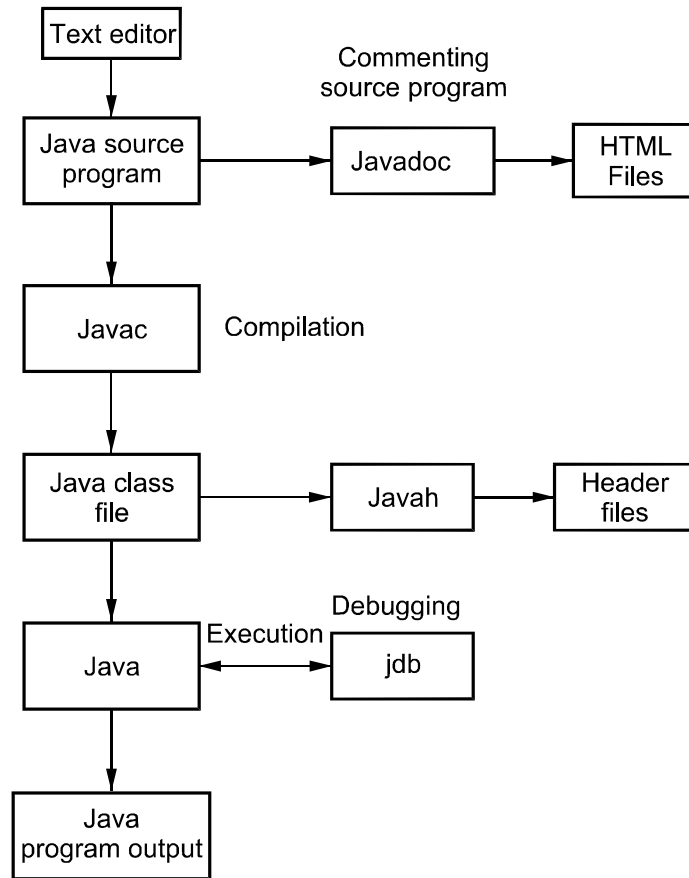


Fig. 1.5.1 Execution process of application program

1.5.2 Application Programming Interface

The Java API consists of large number of classes and methods. These classes and methods are grouped into package. Examples of some commonly used java packages are -

- **Input output package** : This package known as **java.io**. The collection of classes in this package helps in input and output operations.
- **Language support package** : This package is known as **java.lang**. The collection of classes and methods required for implementing basic features of Java.
- **Networking package** : This package is known as **java.net**. The classes and methods required for communicating with other computers through internet is supported by this package.
- **Applet package** : This package includes set of classes and methods that are required for creating applets.

1.5.3 Java Runtime Environment

The Java Runtime Environment(JRE) supports for execution of Java programs. It consists of following components -

- **Java Virtual Machine(JVM)** : The JVM takes the byte code as an input, reads it, interprets it and then executes it. The JVM can generate output corresponding to the underlying platform(operating system). Due to JVM the Java programs become portable.
- **Run Time Class Libraries** : These libraries consist of core class libraries that are required for the execution of Java program.
- **Graphical User Interface Toolkit** : In Java there are two types of GUI toolkits - AWT and swing.
- **Deployment Technologies** : The **Java-plugins** are available that help in the execution of Java applet on the web browser.

1.5.4 Architecture of JVM

JVM stands for Java Virtual Machine. It analyzes the bytecode, interprets the code and executes it.

JVM is divided into three main subsystems

- 1) ClassLoader subsystem
- 2) Runtime data area
- 3) Execution engine

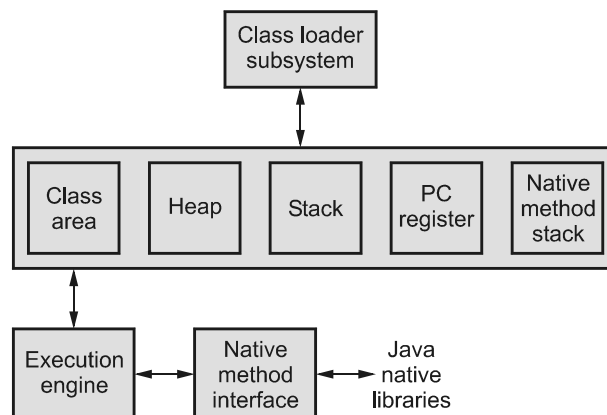


Fig. 1.5.2 Architecture of JVM

- 1) **ClassLoader subsystem** : Java's dynamic class loading functionality is handled by the ClassLoader subsystem. It loads, links the class file. Classes are loaded with the help of -
 - i) **Bootstrap class loader** : It is responsible for loading classes from the bootstrap classpath. Highest priority will be given to this loader.

- ii) **Extension class loader** : It is responsible for loading the classes which are inside the ext folder.
 - iii) **Application class loader** : Responsible for loading Application Level Classpath, path mentioned Environment Variable, etc.
- 2) **Runtime data area** : The Runtime data area is divided into five major components :
- i) **Method area** - All the class-level data will be stored here, including static variables. There is only one method area per JVM, and it is a shared resource.
 - ii) **Heap area** - All the Objects and their corresponding instance variables and arrays will be stored here. There is also one Heap area per JVM. Since the Method and Heap areas share memory for multiple threads, the data stored is not thread-safe.
 - iii) **Stack area** - For every thread, a separate runtime stack will be created. For every method call, one entry will be made in the stack memory which is called Stack Frame. All local variables will be created in the stack memory. The stack area is thread-safe since it is not a shared resource.
 - iv) **PC registers** - Each thread will have separate PC registers, to hold the address of current executing instruction once the instruction is executed the PC register will be updated with the next instruction.
 - v) **Native Method stacks** - Native method stack holds native method information. For every thread, a separate native method stack will be created.
- 3) **Execution engine** : The bytecode, which is assigned to the runtime data area, will be executed by the execution engine. The execution engine reads the bytecode and executes it piece by piece.
- i) **Interpreter** - The interpreter interprets the bytecode faster but executes slowly. The disadvantage of the interpreter is that when one method is called multiple times, every time a new interpretation is required.
 - ii) **JIT compiler** - The JIT compiler neutralizes the disadvantage of the interpreter. The execution engine will be using the help of the interpreter in converting byte code, but when it finds repeated code it uses the JIT compiler, which compiles the entire bytecode and changes it to native code. This native code will be used directly for repeated method calls, which improve the performance of the system.
- Java Native Interface** : Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc.

Review Question

1. What is JVM ? Explain the internal architecture of JVM with neat sketch.

AU : Dec.-19, Marks 5

1.6 Structure of Java Program

- The program structure for the Java program is as given in the following figure -

Documentation Section
package statements section
import statements section
Class Definition
<pre>Main Method class { public static void main(String[] args) { //main method definition } }</pre>

Documentation section : The documentation section provides the information about the source program. This section contains the information which is not compiled by the Java. Everything written in this section is written as comment.

Package section : It consists of the name of the package by using the keyword **package**. When we use the classes from this package in our program then it is necessary to write the package statement in the beginning of the program.

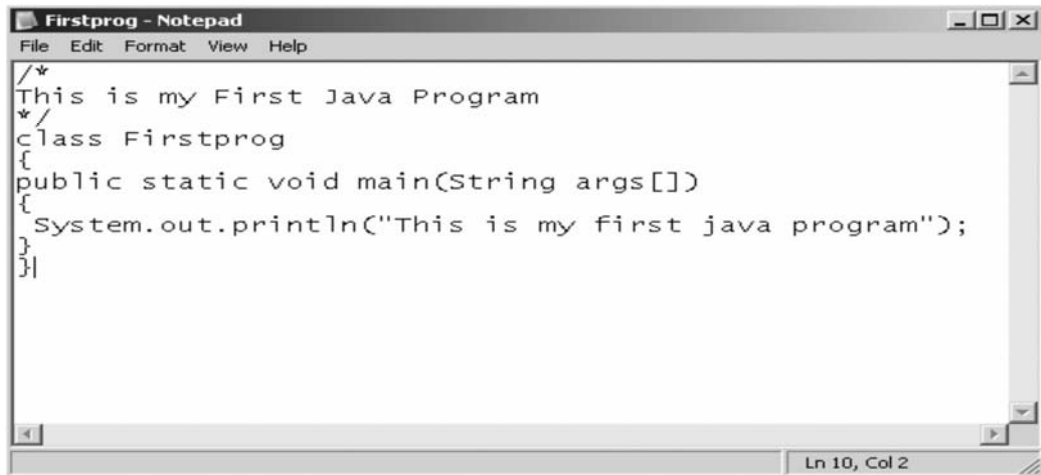
Import statement section : All the required java API can be imported by the import statement. There are some core packages present in the java. These packages include the classes and method required for java programming. These packages can be imported in the program in order to use the classes and methods of the program.

Class definition section : The class definition section contains the definition of the class. This class normally contains the data and the methods manipulating the data.

Main method class : This is called the main method class because it contains the main() function. This class can access the methods defined in other classes.

1.6.1 Writing Simple Java Program

- Any Java program can be written using simple **text editors** like Notepad or Wordpad. The file name should be same as the name of the class used in the corresponding Java program. Note that the name of the program is *Firstprog* and class name is *Firstprog*. The extension to the file name should be **.java**. Therefore we have saved our first program by the name *Firstprog.java*.

Java Program[Firstprog.java]

```
File Edit Format View Help
/*
This is my First Java Program
*/
class Firstprog
{
public static void main(String args[])
{
System.out.println("This is my first java program");
}
}
```

Ln 10, Col 2

The output of this program can be generated on the command prompt using the commands
javac filename.java
java filename

The sample output is as shown below for the program *Firstprog.java*



```
C:\> Command Prompt

D:\>javac Firstprog.java
D:\>java Firstprog
This is my first java program
D:\>
```

Program explanation

In our First Java program, on the first line we have written a comment statement as

```
/*  
This is my First Java Program  
*/
```

This is a multi-line comment. Even a single line comment like C++ is also allowed in Java.

Hence if the statement would be

```
//This is my First Java Program
```

Is perfectly allowed in Java. Then actual program starts with

```
class Firstprog
```

Here *class* is a keyword and *Firstprog* is a variable name of class. Note that **the name of the class and name of your Java program should be the same**. The class definition should be within the curly brackets. Then comes

```
public static void main(String args[])
```

This line is for a function **void main()**. The main is of type *public static void*.

The *public* is a access mode of the *main()* by which the class visibility can be defined. Typically *main* must be declared as *public*.

The parameter which is passed to main is *String args[]*. Here *String* is a class name and *args[]* is an array which receives the command line arguments when the program runs.

```
System.out.println("This is my first java program");
```

To print any message on the console *println* is a method in which the string "This is my first java program" is written. After *println* method, the newline is invoked. That means next output if any is on the new line. This *println* is invoked along with *System.out*. The *System* is a predefined class and *out* is an output stream which is related to console. The output as shown in above figure itself is a self explanatory. Also remember one fact while writing the Java programs that it is a **case sensitive** programming language like C or C++.

1.7 Fundamental Programming Structures in Java

AU : May-19, Marks 6

1.7.1 Java Tokens

The smallest individual and logical unit of the java statements are called tokens. In Java there are five types of tokens used. These tokens are -

1. Reserved keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

Reserved keywords

Keywords are the reserved words which are enlisted as below-

abstract	default	int	super
assert	double	interface	switch
boolean	else	long	this
byte	extends	native	throw
break	final	new	throws
case	for	package	transient
catch	float	private	try
char	goto	protected	void
class	if	public	volatile
const	implements	return	while
continue	import	short	true
do	instanceof	static	false
			null

Identifiers

Identifiers are the kind of tokens defined by the programmer. They are used for naming the classes, methods, objects, variables, interfaces and packages in the program.

Following are the rules to be followed for the identifiers-

1. The identifiers can be written using alphabets, digits, underscore and dollar sign.
2. They should not contain any other special character within them.
3. There should not be a space within them.
4. The identifier must not start with a digit, it should always start with alphabet.
5. The identifiers are case-sensitive. For example -

```
int a;  
int A;
```

In above code **a** and **A** are treated as two different identifiers.

6. The identifiers can be of any length.

Literals

Literals are the kind of variables that store the sequence of characters for representing the constant values. Five types of literals are -

1. Integer literal
2. Floating point literal
3. Boolean literal
4. Character literal
5. String literal

As the name suggests the corresponding data type constants can be stored within these literals.

Operators

Operators are the symbols used in the expression for evaluating them.

Separators

For dividing the group of code and arranging them systematically certain symbols are used, which are known as separators. Following table describes various separators.

Name of the Separator	Description
()	Parenthesis are used to - enclose the arguments passed to it to define precedence in the expression. For surrounding cast type
{ }	Curly brackets are used for - initialising array for defining the block
[]	Square brackets are used for - declaring array types dereferencing array values
;	Used to terminate the statement
,	Commas are used to separate the contents.
.	Period is used to separate the package name from subpackages.

1.7.2 Constants

Constants mean the fixed values that do not change during the execution of a program.

In Java various types of constants are used and these are as shown in following Fig. 1.7.1.

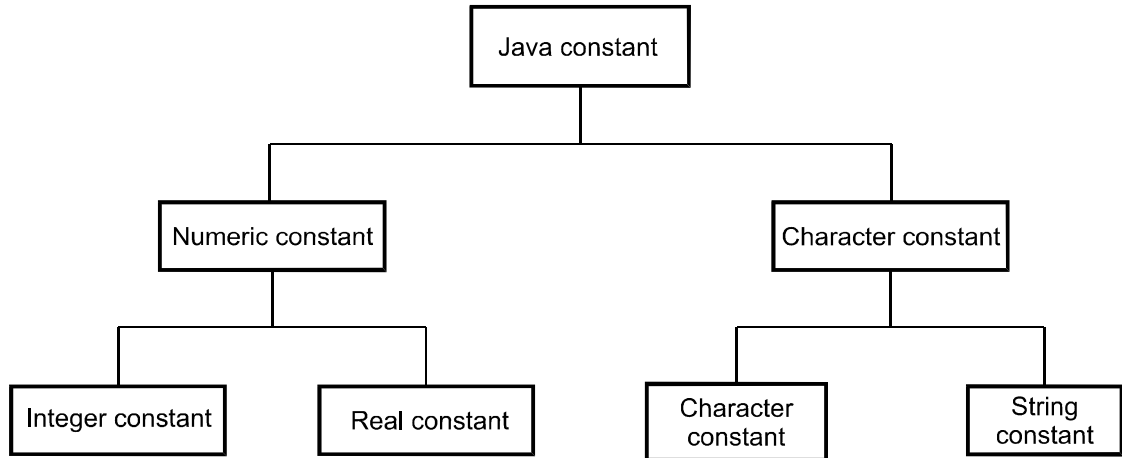


Fig. 1.7.1 Types of Constants

Numeric constants

Numeric constants are used for representing the numerical values. The numeric values can be integer or real type. Hence there exists the integer and real constants.

Integer constants

Integers mean sequence of digits. There are three types of integers -

Decimal integer : The decimal integers consist of a set of digits 0 through 9. These numbers can be unary numbers. That means these can be preceded by unary minus. For example

111, -200, 455

Octal integer : The octal integer constants consists of set of 0 to 7 having a zero at the beginning. For example -

033, 0431, 0633

Hexadecimal integer : Along with the 0 to 9 digits, it also consists of the letters A to F.

Real constants : The real constants contain the floating point parts. For example-

0.045, 0.567, 0.3211

Character constants : A single character constant is enclosed within a pair of single quotes. For example-

'a', 'T', '1'

String constant : A string is a collection of characters enclosed between double quotes. The characters can be alphabets, numbers, white spaces or special characters.

For example -

"India", "Best10", "s?ss"

1.7.3 Variables

A variable is an identifier that denotes the storage location.

Variable is a fundamental unit of storage in Java. The variables are used in combination with identifiers, data types, operators and some value for initialization. The variables must be declared before its use. The **syntax** of variable declaration will be -

`data_type name_of_variable [=initialization][, =initialization][,...];`

Following are some rules for variable declaration -

- The variable name should not with digits.
- No special character is allowed in identifier except underscore.
- There should not be any blank space with the identifier name.
- The identifier name should not be a keyword.
- The identifier name should be meaningful.

For Example :

```
int a,b;
char m='a';
byte k=12,p,t=22;
```

The variables have a scope which defines their visibility and a lifetime.

1.7.4 Data Types

Various data types used in Java are byte, short, int, long, char, float, double and boolean.

byte

This is in fact smallest integer type of data type. Its width is of 8-bits with the range -128 to 127. The variable can be declared as byte type as

```
byte i,j;
```

short

This data type is also used for defining the signed numerical variables with a width of 16 -bits and having a range from -32,768 to 32,767. The variable can be declared as short as

```
short a,b;
```

int

This is the most commonly used data type for defining the numerical data. The width of this data type is 32-bit having a range 2,147,483,648 to 2,147,483,647. The declaration can be

```
int p,q;
```

long

Sometimes when *int* is not sufficient for declaring some data then *long* is used. The range of long is really very long and it is -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. The declaration can be

```
long x,y;
```

float

To represent the real number(i.e. number that may have decimal point) float data type can be used. The width is 32-bit and range of this data type is 1.4e - 045 to 3.4e+038.

double

To represent the real numbers of large range the double data type is used. Its width is 64-bit having the range 4.9e-324 to 1.8e+308.

Char

This data type is used to represent the character type of data. The width of this data type is 16-bit and its range is 0 to 65,536.

Let us see one simple Java program which makes use of various data types.

boolean

Boolean is a simple data type which denotes a value to be either true or false.

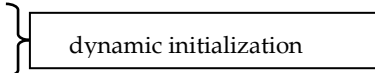
Java Program

Note that for displaying the value of variable, in the **println** statement the variable is appended to the message using + [In C we use comma in printf statement but in Java + is used for this purpose]

Java Program [DatatypeDemo.java]

```
/*  
This program introduces use of various data type  
in the program  
*/  
class DatatypeDemo  
{  
    public static void main(String args[])
```

```
{
    byte a=10;
    short b=10*128;
    int c=10000* 128;
    long d=10000*1000*128;
    double e=99.9998;
    char f='a';
    boolean g=true;
    boolean h=false;
    System.out.println("The value of a=: "+a);
    System.out.println("The value of b=: "+b);
    System.out.println("The value of c=: "+c);
    System.out.println("The value of d=: "+d);
    System.out.println("The value of e=: "+e);
    System.out.println("The value of f=: "+f);
    f++;
    System.out.println("The value of f after increment=: "+f);
    System.out.println("The value of g=: "+g);
    System.out.println("The value of h=: "+h);
}
```



Output

```
The value of a=: 10
The value of b=: 1280
The value of c=: 1280000
The value of d=: 1280000000
The value of e=: 99.9998
The value of f=: a
The value of f after increment=: b
The value of g=: true
The value of h=: false
```

Dynamic initialization

- Java is a flexible programming language which allows the dynamic initialization of variables. In other words, at the time of declaration one can initialize the variables(*note that in the above program we have done dynamic initialization*). In Java we can declare the variable at any place before it is used.

Review Questions

1. What is a data type ? Explain various data types available in Java.
2. What are literals ? Explain the types of literals supported by java.

AU : May-19, Marks 6

1.7.5 Operators

Various operators that are used in Java are enlisted in following table -

Type	Operator	Meaning	Example
Arithmetic	+	Addition or unary plus	c=a+b
	-	Subtraction or unary minus	d= -a
	*	Multiplication	c=a*b
	/	Division	c=a/b
	%	Mod	c=a%b
Relational	<	Less than	a<4
	>	Greater than	b>10
	<=	Less than equal to	b<=10
	>=	Greater than equal to	a>=5
	==	Equal to	x==100
	!=	Not equal to	m!=8
Logical	&&	And operator	0&&1
		Or operator	0 1
Assignment	=	is assigned to	a=5
Increment	++	Increment by one	++i or i++
Decrement	--	Decrement by one	-- k or k--

Arithmetic Operators

The arithmetic operators are used to perform basic arithmetic operations. The operands used for these operators must be of numeric type. The Boolean type operands can not be used with arithmetic operators.

Java Program

```

////////////////////////////////////
//Program for demonstrating arithmetic operators
////////////////////////////////////

```

```
class ArithOperDemo
{
    public static void main(String[] args)
    {
        System.out.println("\n Performing arithmetic operations ");
        int a=10,b=20,c;
        System.out.println("a= "+a);
        System.out.println("b= "+b);
        c=a+b;
        System.out.println("\n Addition of two numbers is "+c);
        c=a-b;
        System.out.println("\n Subtraction of two numbers is "+c);
        c=a*b;
        System.out.println("\n Multiplication of two numbers is "+c);
        c=a/b;
        System.out.println("\n Division of two numbers is "+c);
    }
}
```

Output

Performing arithmetic operations

a= 10

b= 20

Addition of two numbers is 30

Subtraction of two numbers is -10

Multiplication of two numbers is 200

Division of two numbers is 0

Relational Operators

The relational operators typically used to denote some condition. These operators establish the relation among the operators. The <, >, <=, >= are the relational operators. The result of these operators is a Boolean value.

Java Program

```
////////////////////////////////////
//Program for demonstrating relational operators
////////////////////////////////////
import java.io.*;
import java.lang.*;
import java.util.*;
public class RelOper
```

```
{
    public static void main(String args[])
    {
        int a,b,c;
        a=10;
        b=20;
        if(a>b)
        {
            System.out.println("a is largest");
        }
        else
        {
            System.out.println("b is largest");
        }
    }
}
```

Logical Operators

The logical operators are used to combine two operators. These two operands are Boolean operators.

Java Program

```
////////////////////////////////////
//Program for demonstrating logical operators
////////////////////////////////////
import java.io.*;
import java.lang.*;
public class LogicalOper
{
    public static void main(String args[])throws IOException
    {
        boolean oper1,oper2;
        oper1=true;
        oper2=false;
        boolean ans1,ans2;
        ans1=oper1&oper2;
        ans2=oper1|oper2;
        System.out.println("The oper1 is: "+oper1);
        System.out.println("The oper2 is: "+oper2);
        System.out.println("The oper1&oper2 is: "+ans1);
        System.out.println("The oper1|oper2 is: "+ans2);
    }
}
```

Output

The oper1 is: true

The oper2 is: false

The oper1&oper2 is: false

The oper1|oper2 is: true

Special Operators

- There are two special operators used in Java-**instanceof** and **dot** operators.
- For determining whether the object belongs to particular class or not- an **instanceof** operator is used. For example-
- Ganga instanceof River if the object Ganga is an object of class River then it returns true otherwise false.
- The **dot** operator is used to access the instance variable and methods of class objects.

For example -

```
Customer.name //for accessing the name of the customer  
Customer.ID   //for accessing the ID of the customer
```

Conditional Operator

The conditional operator is “?” The syntax of conditional operator is

Condition?expression1:expression2

Where expression1 denotes the true condition and expression2 denotes false condition.

For example :

```
a>b?true:false
```

This means that if a is greater than b then the expression will return the value true otherwise it will return false.

1.7.6 Expressions

Expression is a combination of operators and operands in specific manner. For example

```
c=a+b*c; // Expression with arithmetic operators
```

```
(a>b)&&(b<c)// Expression with logical operators and logical operators
```

1.8 Control Statements

AU : May-19, Marks 7

Programmers can take decisions in their program with the help of control statements. Various control statements that can be used in java are –

1. if statement
2. if else statement
3. while statement
4. do...while statement
5. switch case statement
6. for loop

Let us discuss each of the control statement in details -

1. if statement

The if statement is of two types

- i) **Simple if statement** : The if statement in which only one statement is followed by that is statement.

Syntax

```
if(apply some condition)
    statement
```

For example

```
if(a>b)
    System.out.println("a is Biiiiig!");
```

- ii) **Compound if statement** : If there are more than one statement that can be executed when if condition is true. Then it is called compound if statement. All these executable statements are placed in curly brackets.

Syntax

```
if(apply some condition)
{
    statement 1
    statement 2
    .
    .
    .
    statement n
}
```

2. if...else statement

The syntax for if...else statement will be -

```
if(condition)
    statement
else
    statement
```

For example

```
if(a>b)
    System.out.println("a is big")
else
    System.out.println("b :big brother")
The if...else statement can be of compound type even. For example
if(raining==true)
{
    System.out.println("I won't go out");
    System.out.println("I will watch T.V. Serial");
    System.out.println("Also will enjoy coffee");
}
else

{
    System.out.println("I will go out");
    System.out.println("And will meet my friend");
    System.out.println("we will go for a movie");
    System.out.println("Any how I will enjoy my life");
}
```

if...else if statement

The syntax of if...else if statement is

```
if(is condition true?)
    statement
else if(another condition)
    statement
else if(another condition)
    statement
else
    statement
```

For example

```
if(age==1)
    System.out.println("You are an infant");
else if(age==10)
    System.out.println("You are a kid");
else if(age==20)
    System.out.println("You are grown up now");
else
    System.out.println("You are an adult");
```

Let us now see Java programs using this type of control statement-

Java Program [ifelsedemo.java]

```
/*
This program illustrates the use of
if else statement
*/

class ifelsedemo
{
    public static void main(String[] args)
    {
        int x=111,y=120,z=30;
        if(x>y)
        {
            if(x>z)
                System.out.println("The x is greatest");
            else
                System.out.println("The z is greatest");
        }
        else
        {
            if(y>z)
                System.out.println("The y is greatest");
            else
                System.out.println("The z is greatest");
        }
    }
}
```

Output

The y is greatest

Java Program [ifdemo.java]

```
/*This program illustrates the use of
if...else if statement
*/

class ifdemo
{
    public static void main(String args[])
    {
        int basic=20000;
        double gross,bonus;
        if(basic<10000)
        {
            bonus=0.75*basic;
            gross=basic+bonus;
            System.out.println("Your Salary is = "+gross);
        }
    }
}
```

```
}
else if(basic>10000&&basic<=20000)
{
    bonus=0.50*basic;
    gross=basic+bonus;
    System.out.println("Your Salary is= "+gross);
}
else if(basic>20000&&basic<=50000)
{
    bonus=0.25*basic;
    gross=basic+bonus;
    System.out.println("Your Salary is= "+gross);
}
else
{
    bonus=0.0;
    gross=basic+bonus;
    System.out.println("Your Salary is= "+gross);
}
}
```

Output

Your Salary is= 30000.0

3. while statement

This is another form of while statement which is used to have iteration of the statement for the any number of times. The syntax is

```
while(condition)
{
    statement 1;
    statement 2;
    statement 3;
    ...
    statement n;
}
```

For example

```
int count=1;
while(count<=5)
{
    System.out.println("I am on line number "+count);
    count++;
}
```

Let us see a simple Java program which makes the use of while construct.

Java Program [whiledemo.java]

```
/*
This is java program which illustrates
while statement
*/
class whiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        while(count<=5)
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }
    }
}
```

Output

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

4. do... while statement

- This is similar to while statement but the only **difference** between the two is that in case of do...while statement the statements inside the do...while must be executed at least once.
- This means that the statement inside the do...while body gets **executed first** and then the while condition is checked for next execution of the statement, whereas in the while statement first of all the condition given in the while is checked first and then the statements inside the while body get executed when the condition is true.

- **Syntax**

```
do
{
    statement 1;
    statement 2;
    ...
    statement n;
}while(condition);
```

For example

```
int count=1;
do
{
    System.out.println("I am on the first line of do-while");
    System.out.println("I am on the second line of do-while");
    System.out.println("I am on the third line of do-while");
    System.out.println("I am on the forth line of do-while");
    System.out.println("I am on the fifth line of do-while");
    count ++;
}while(count<=5);
```

Java Program [dowhiledemo.java]

```
/*
This is java program which illustrates
do...while statement
*/
class dowhiledemo
{
    public static void main(String args[])
    {
        int count=1,i=0;
        do
        {
            i=i+1;
            System.out.println("The value of i= "+i);
            count++;
        }while(count<=5);
    }
}
```

Output

```
The value of i= 1
The value of i= 2
The value of i= 3
The value of i= 4
The value of i= 5
```

5. switch statement

You can compare the switch case statement with a Menu-Card in the hotel. You have to select the menu then only the order will be served to you.

Here is a sample program which makes use of switch case statement -

Java Program [switchcasedemo.java]

```
/*
This is a sample java program for explaining
Use of switch case
*/
class switchcasedemo
{
    public static void main(String args[])
    throws java.io.IOException
    {
        char choice;
        System.out.println("\tProgram for switch case demo");
        System.out.println("Main Menu");
        System.out.println("1. A");
        System.out.println("2. B");
        System.out.println("3. C");
        System.out.println("4. None of the above");
        System.out.println("Enter your choice");
        choice=(char)System.in.read();
        switch(choice)
        {
            case '1':System.out.println("You have selected A");
                break;
            case '2':System.out.println("You have selected B");
                break;
            case '3':System.out.println("You have selected C");
                break;
            default:System.out.println("None of the above choices made");
        }
    }
}
```

Output

```
    Program for switch case demo
Main Menu
1. A
2. B
3. C
4. None of the above
Enter your choice
2
You have selected B
```

Note that in above program we have written main() in somewhat different manner as -
public static void main(String args[])

throws java.io.IOException

This is **IOException** which must be thrown for the statement **System.in.read()**. Just be patient, we will discuss the concept of Exception shortly! The **System.in.read()** is required for reading the input from the console[**note that it is parallel to scanf statement in C**]. Thus using **System.in.read()** user can enter his choice. Also note that whenever **System.in.read()** is used it is necessary to write the **main()with IOException** in order to handle the input/output error.

6. for loop

for is a keyword used to apply loops in the program. Like other control statements for loop can be categorized in simple for loop and compound for loop.

Simple for loop :

```
for (statement 1;statement 2;statement 3)
    execute this statement;
```

Compound for loop :

```
for(statement 1;statement 2; statement 3)
{
    execute this statement;
    execute this statement;
    execute this statement;
    that's all;
}
```

Here

Statement 1 is always for initialization of conditional variables,

Statement 2 is always for terminating condition of the for loop,

Statement 3 is for representing the stepping for the next condition.

For example :

```
for(int i=1;i<=5;i++)
{
    System.out.println("Java is an interesting language");
    System.out.println("Java is a wonderful language");
    System.out.println("And simplicity is its beauty");
}
```

Let us see a simple Java program which makes use of for loop.

Java Program [forloop.java]

```
/*
This program shows the use of for loop
```

```
*/  
class forloop  
{  
    public static void main(String args[])  
    {  
        for(int i=0;i<=5;i++)  
            System.out.println("The value of i: "+i);  
    }  
}
```

Output

The value of i: 0
The value of i: 1
The value of i: 2
The value of i: 3
The value of i: 4
The value of i: 5

Ex. 1.8.1 : Write a Java program to find factorial of a given number.

Sol. :

Method 1 : Factorial program without recursion i.e. using control statement

```
class FactorialProgram  
{  
    public static void main(String args[])  
    {  
        int i,fact=1;  
        int number=5;//The factorial of 5 is calculated  
        i=1;  
        while(i<number)  
        {  
            fact=fact*i;  
            i++;  
        }  
        System.out.println("Factorial of "+number+" is: "+fact);  
    }  
}
```

Output

Factorial of 5 is: 24

Method 2 : Factorial program using recursion

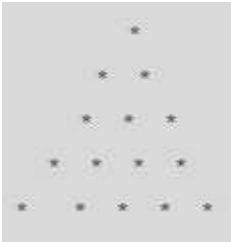
```
class FactorialProgramRec  
{  
    static int factorial(int n)  
    {  
        if (n == 0)
```

```
        return 1;
    else
        return(n * factorial(n-1)); //recursive call
    }
    public static void main(String args[])
    {
        int i,fact=1;
        int number=5;//Factorial of this number is calculated
        fact = factorial(number);
        System.out.println("Factorial of "+number+" is: "+fact);
    }
}
```

Output

Factorial of 5 is: 120

Ex. 1.8.2 : Write a Java program to display following pattern.



Sol. :

```
class TriangleDisplay
{
    public static void main(String[] args)
    {
        int i,j,k;
        for(i=1; i<=5; i++)
        {
            for(j=4; j>=i; j--)
            {
                System.out.print(" ");
            }
            for(k=1; k<=(2*i-1); k++)
            {
                System.out.print("*");
            }
            System.out.println("");
        }
    }
}
```

Output

```
*
***
*****
*****
*****
```

Ex. 1.8.3 : Write a Java program to display following number pattern

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Sol. :

```
class NumberPatternDisplay
{
    public static void main(String[] args)
    {
        int i,j;
        for(i=1; i<=5; i++)
        {
            for(j=1; j<=i; j++)
            {
                System.out.print(j+" ");
            }
            System.out.println("");
        }
    }
}
```

Output

```
Command Prompt

D:\>javac NumberPatternDisplay.java

D:\>java NumberPatternDisplay
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

D:\>
```

Ex. 1.8.4 : Write a program for following series $1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^n$

Sol. :

Java Program

```
////////////////////////////////////  
//Program for computing  $1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^n$   
////////////////////////////////////  
import java.io.*;  
import java.lang.*;  
import java.math.*;  
public class Series1  
{  
    public static void main(String args[])throws IOException  
    {  
        int n;  
        double val;  
        double sum=1;  
        n=5;  
        System.out.println("\n\t\tProgram for displaying the sum of series");  
        for(int i=1;i<n;i++)  
        {  
            val=1/(Math.pow(2,i));  
            sum=sum+val;  
        }  
        System.out.println("\nSum of the series is "+sum);  
    }  
}
```

Output

Program for displaying the sum of series
Sum of the series is 1.9375

Ex. 1.8.5 : Write a Java application that computes the value of ex by using the following series. $ex = 1 + x/1! + x^2/2! + x^3/3! + \dots$

Sol. :

```
////////////////////////////////////  
//Program for computing sum of series  
////////////////////////////////////  
import java.io.*;  
import java.lang.*;  
import java.math.*;  
public class Series2  
{  
    public static int fact(int n)
```

```
{
    int x,y;
    if(n<0)
    {
        System.out.println("The negative parameter in the factorial function");
        return -1;
    }
    if(n==0)
        return 1;
    x=n-1;
    y=fact(x);    /*recursive call*/
    return(n*y);
}
public static void main(String args[])throws IOException
{
    int n;
    float val;
    float sum=1;
    int x=2;
    n=3;
    System.out.println("\n\t\tProgram for displaying the sum of series");
    for(int i=1;i<=n;i++)
    {
        val=(float)(Math.pow(x,i))/fact(i);
        sum=sum+val;
    }
    System.out.println("\nSum of the series is "+sum);
}
}
```

Output

Program for displaying the sum of series
Sum of the series is 6.3333335

Ex. 1.8.6 : Write a Java program that reverses the digits of given number.

Sol. :

```
////////////////////////////////////
//Program for reversing digits of given number
////////////////////////////////////
import java.io.*;
public class Reversenum
{
    public static void main(String[] args)
    {
        int num=1234;
```

```
int rev_num=0;
while(num!=0)
{
    rev_num=(rev_num*10)+(num%10);
    num=num/10;
}
System.out.println(rev_num);
}
}
```

Output

4321

Ex.1.8.7 : Write a Java code using do-while loop that counts down to 1 from 10 printing exactly ten lines of “hello”.

AU : May-19, Marks 6

Sol. :

```
class test
{
    public static void main(String args[])
    {
        int count = 10;
        do
        {
            System.out.println("Hello");
            count--;
        }while(count>=1);
    }
}
```

Output

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello

Review Question

1. Explain the selection statements in Java using suitable examples.

AU : May-19, Marks 7

1.9 The break and continue Statement

Sometimes when we apply loops we need to come out of the loop on occurrence of particular condition. This can be achieved by break statement. Following program illustrates the use of break in the for loop.

Java Program [breakdemo.java]

```
/*
This program shows the use of break statement
*/
class breakdemo
{
    public static void main(String args[])
    {
        for(int i=1;i<=20;i++)
        {
            if(i%10==0)
            {
                System.out.println("number "+i+" is divisible by 10");
                break;//come out of for loop
            }
            else
                System.out.println("number "+i+" is not divisible by 10");
        }
    }
}
```

Output

```
The number 1 is not divisible by 10
The number 2 is not divisible by 10
The number 3 is not divisible by 10
The number 4 is not divisible by 10
The number 5 is not divisible by 10
The number 6 is not divisible by 10
The number 7 is not divisible by 10
The number 8 is not divisible by 10
The number 9 is not divisible by 10
The number 10 is divisible by 10
```

Program Explanation : Note that in the above program the numbers after 10 will not be considered at all. This is because when we reach at 10 then if condition becomes true and we encounter break statement. This forces us to come out of for loop. Just look at the output of corresponding program!!!

Similarly we can use the return statement which is useful for returning the control from the current block.

Difference between break and continue

Sr.No.	break	continue
1.	The break is used to terminate the execute the loop.	The continue statement is not used to terminate the execution
2.	It breaks iteration.	It skips the iteration.
3.	Break is used in loops and in switch.	Continue is used only in loop.
4.	When this break is executed, the control will come out of the loop.	When continue is executed the control will not come out of the loop, in fact it will jump to next iteration of loop.
5.	<p>Example:</p> <pre>public class breakdemo { public static void main(String[] args) { for (int i = 1; i <= 20; i++) { if (i % 2 == 0) { break; // skip the even //numbers } System.out.println(i + " "); //will display only odd numbers } } }</pre> <p>Output</p> <p>1</p>	<p>Example:</p> <pre>public class continuedemo { public static void main(String[] args) { for (int i = 1; i <= 20; i++) { if (i % 2 == 0) { continue; // skip the even //numbers } //will display only odd numbers System.out.println(i + " "); } } }</pre> <p>Output</p> <p>1 3 5 7 9 11 13 15 17</p>

1.10 Defining Classes in Java

AU : IT : May-12, CSE : Dec.-10, 14, Marks 5

Each class is a collection of data and the functions that manipulate the data. The data components of the class are called data fields and the function components of the class are called member functions or methods.

- Thus the state of an object is represented using **data fields** (data members) and behaviour of an object is represented by set of **methods** (member functions). The class template can be represented by following Fig. 1.10.1.

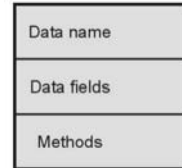
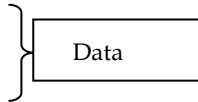


Fig. 1.10.1 Class template

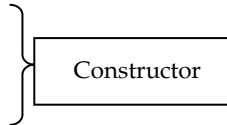
- The **Java Class** can written as follows

```
class Customer {
```

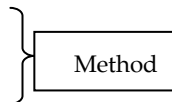
```
int ID;
int Age;
String Name;
```



```
Customer() {
}
Customer(int ID) {
}
```



```
double withdraw_money() {
...
...
}
```

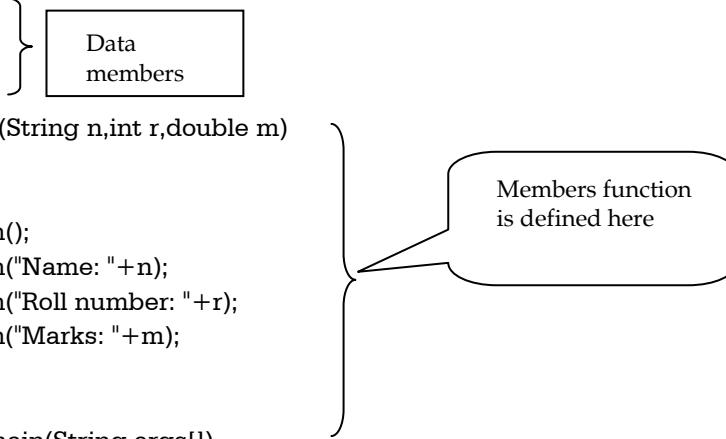


- Encapsulation means binding of data and method together in a single entity called class. Thus a class is used to achieve an **encapsulation** property.
- This class is not having the main function. The class that contains main function is called **main class**.

- Following is a simple Java program that illustrates the use of class

```
////////////////////////////////////
//Program for demonstrating the concept of class
////////////////////////////////////
import java.io.*;
import java.lang.*;
import java.math.*;
public class MyClass
```

```
{
String name;
int roll;
double marks;
public void display(String n,int r,double m)
{
    System.out.println();
    System.out.println("Name: "+n);
    System.out.println("Roll number: "+r);
    System.out.println("Marks: "+m);
}
public static void main(String args[])
{
    int a,b;
    MyClass obj1=new MyClass();
    MyClass obj2=new MyClass();
    MyClass obj3=new MyClass();
    obj1.display("Amar",10,76.65);
    obj2.display("Akbar",20,87.33);
    obj3.display("Anthony",30,96.07);
}
}
```



The diagram illustrates the structure of the provided Java code. A box labeled "Data members" is connected by a bracket to the variables `String name;`, `int roll;`, and `double marks;`. Another box labeled "Members function is defined here" is connected by a bracket to the `display` method and the `main` method.

Output

Name: Amar
Roll number: 10
Marks: 76.65

Name: Akbar
Roll number: 20
Marks: 87.33

Name: Anthony
Roll number: 30
Marks: 96.07

Program Explanation

- In above program we have declared one simple class. This class displays small database of the students. The data members of this class are name, roll and marks. There is one member function named **display** which is for displaying the data members. In the **main** function we have created three objects **obj1**, **obj2** and **obj3**. These objects represent the real world entities. These entities are actually the instances of the class. The class representation of the above program can be as follows -

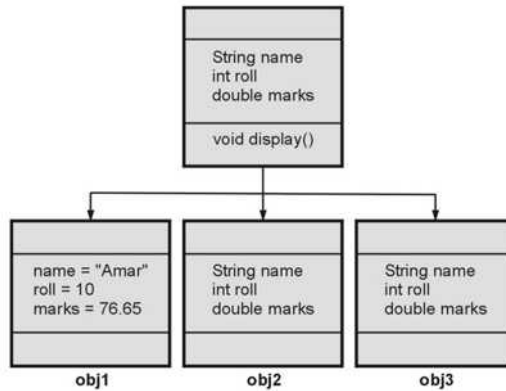


Fig. 1.10.2

1.10.1 Methods Defined in Class

Methods are nothing but the functions defined by the particular class.

```
class classname
```

```
{
```

```
    declaration of member  
    variables;
```

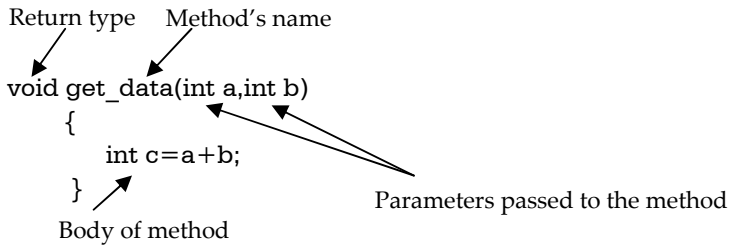
```
    definition of  
    method
```

```
    {  
        ...  
        ...  
    }
```

```
}
```

There are four parts of the method -

- Return type of the method
- name of the method
- parameter passed to the method
- body of the method

For example**1.10.1.1 Parameter Passing**

- Parameter can be passed to the function by two ways -
 - Call by value
 - Call by reference
- In the **call by value** method the value of the actual argument is assigned to the formal parameter. If any change is made in the formal parameter in the subroutine definition then that change does not reflect the actual parameters.
- Following Java program shows the use of parameter passing by value -

Java Program

```

////////////////////////////////////
//Program implementing the parameter passing by value
////////////////////////////////////
public class ParameterByVal
{
    void Fun(int a,int b)
    {
        a=a+5;
        b=b+5;
    }
    public static void main(String args[])
    {
        ParameterByVal obj1=new ParameterByVal();
        int a,b;
        a=10;b=20;
        System.out.println("The values of a and b before function call");
        System.out.println("a= "+a);
        System.out.println("b= "+b);
        obj1.Fun(a,b);
        System.out.println("The values of a and b after function call");
    }
}
  
```

```
System.out.println("a= "+a);
System.out.println("b= "+b);

}
}
```

Output

The values of a and b before function call

a= 10

b= 20

The values of a and b after function call

a= 10

b= 20

Program Explanation

- The above Java program makes use of the parameter passing by value. By this approach of parameter passing we are passing the actual values of variables a and b.
- In the function **Fun** we are changing the values of **a** and **b** by adding 5 to them. But these incremented values will remain in the function definition body only. After returning from this function these values will not get preserved. Hence we get the same values of a and b before and after the function call.
- On the contrary the parameter passing by reference allows to change the values after the function call. But use of variables as a parameter does not allow to pass the parameter by reference.
- For passing the **parameter by reference** we **pass the object** of the class as a parameter.
- Creating a variable of class type means creating reference to the object. If any changes are made in the object made inside the method then those changes get preserved and we can get the changed values after the function call.

How to Pass Objects to the Method?

The object can be passed to a method as an argument. Using dot operator the object's value can be accessed. Such a method can be represented syntactically as -

```
Data_Type name_of_method(object_name)
{
    //body of method
}
```

Following is a sample Java program in which the method **area** is defined. The parameter passed to this method is an **object**.

Java Program[ObjDemo.java]

```
public class ObjDemo {
    int height;
    int width;
    ObjDemo(int h,int w)
    {
        height=h;
        width=w;
    }
    void area(ObjDemo o)
    {
        int result=(height + o.height)*(width + o.width);
        System.out.println("The area is "+result);
    }
}
class Demo {
    public static void main(String args[])
    {
        ObjDemo obj1=new ObjDemo(2,3);
        ObjDemo obj2=new ObjDemo(10,20);
        obj1.area(obj2);
    }
}
```

height=2 and o.height=10
width=3 and o.width=20
Hence,
 $result=(2+10)*(3+20)$
 $=12*23$
 $=276$

Method with object as parameter

Output

F:\test>javac ObjDemo.java

F:\test>java Demo

The area is 276

Program Explanation

- The method, **area** has object **obj2** as argument. And there is another object named **obj1** using which the method **area** is invoked. Inside the method **area**, the height and width values of invoking object are added with the height and width values of the object to be passed.
- When an object needs to be passed as a parameter to the function then constructor is built. Hence we have to define **constructor** in which the values of the object are used for initialization.

How to Return an Object from a Method?

- We can return an object from a method. The data type such method is a class type.

Java Program[ObjRetDemo.java]

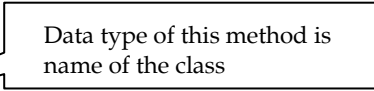
```

public class ObjRetDemo {
    int a;
    ObjRetDemo(int val)
    {
        a=val;
    }
    ObjRetDemo fun()
    {
        ObjRetDemo temp=new ObjRetDemo(a+5); //created a new object
        return temp; //returning the object from this method
    }
}

class ObjRet {
    public static void main(String args[])
    {
        ObjRetDemo obj2=new ObjRetDemo(20);
        ObjRetDemo obj1;
        obj1=obj2.fun();//obj1 gets the value from object temp
        System.out.println("The returned value is = "+obj1.a);

    }
}

```


Output

F:\test>javac ObjRetDemo.java

F:\test>java ObjRet

The returned value is = 25

Ex. 1.10.1 : Write Java program for computing Fibonacci series.

AU : IT : May-12

Sol. :

/******

Program for computing the number in the fibonacci series at certain location. For example the eighth number in fibonacci series will be 21. The fibonacci series is 1 1 2 3 5 8 13 21 34...

*****/

```

import java.io.*;
import java.util.*;
class Fibonacci
{
    public int fib(int n)
    {

```

```

    int x,y;
    if(n<=1)
        return n;
    x=fib(n-1);
    y=fib(n-2);
    return (x+y);
}
} //end of class
class FibonacciDemo
{
    public static void main(String[] args) throws IOException
    {
        Fibonacci f=new Fibonacci();
        int n=8;
        System.out.println("\nThe number at "+n+" is "+f.fib(n));
    }
}

```

Output

D:\>javac FibonacciDemo.java

D:\>java FibonacciDemo

The number at 8 is 21

Review Questions

1. What is class ? How do you define a class in Java ?
2. What is method ? How is method defined ? Give example.

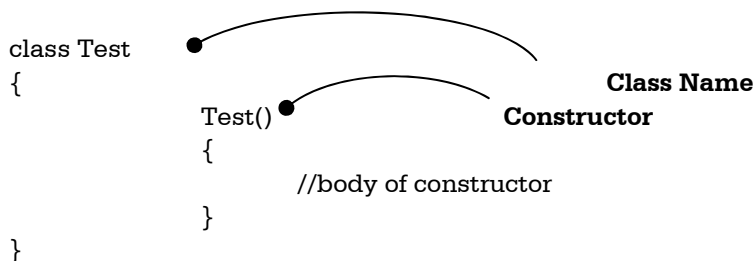
AU : CSE : Dec.-10, Marks 4

AU : Dec.-14, 18, Marks 6

1.11 Constructors

AU : May-14, Dec.-14, 18, Marks 16

- **Definition** : The constructor is a specialized method for **initializing objects**.
- Name of the constructor is same as that of its class name. In other words, the name of the constructor and class name is same.



- Whenever an object of its associated class is created, the constructor is invoked automatically.
- The constructor is called constructor because it creates values for data fields of class.
- Example -

Java Program[Rectangle1.java]

```
public class Rectangle1 {
    int height;
    int width;
    Rectangle1()
    {
        System.out.println(" Simple constructor: values are initialised...");
        height=10;
        width=20;
    }
    Rectangle1(int h,int w)
    {
        System.out.println(" Parameterised constructor: values are initialised...");
        height=h;
        width=w;
    }
    void area()
    {
        System.out.println("Now, The function is called...");
        int result=height*width;
        System.out.println("The area is "+result);
    }
}
```

Data fields

Simple constructor

Parameterised constructor

Method defined

Object created and simple constructor is invoked

Object created and parameterised constructor is invoked

```
class Constr_Demo {
    public static void main(String args[])
    {
        Rectangle1 obj=new Rectangle1();
        obj.area();//call the to method
        Rectangle1 obj1=new Rectangle1(11,20);
        obj1.area();//call the to method
    }
}
```

Output

```
F:\test>javac Rectangle1.java
F:\test>java Constr_Demo
Simple constructor: values are initialised...
```

Note the method of running the program

Now, The function is called...

The area is 200

Parameterised constructor: values are initialised...

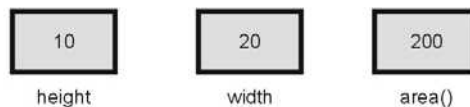
Now, The function is called...

The area is 220

F:\test>

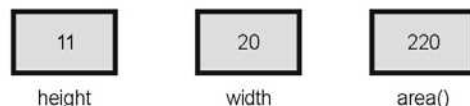
Program Explanation

In above program, there are two classes - **Rectangle1** and **Constr_Demo**. In the **Rectangle1** class, data fields, method and constructor is defined. In the **Constr_Demo** class the objects are created. When an object **obj** gets created, then the simple constructor **Rectangle1()** gets invoked and the data fields height and width gets initialized



Hence the **area** function will compute the $\text{area} = 10 * 20 = 200$.

When an object **obj1** gets created, then the parameterised constructor **Rectangle1(11, 20)** gets invoked and the data fields height and width gets initialised.



Hence the **area** function will compute the $\text{area} = 11 * 20 = 220$.

The class **Constr_Demo** is called **main class** because it consists of **main** function.

Ex. 1.11.1 : Differentiate between constructor and method.

Sol. :

Sr. No.	Constructor	Method
1.	The name of the constructor must be same as the class name.	The name of the method should not be the class name. It can be any with alphanumeric or alphabetic characters with restricted character length. The name must not start with digit or special symbols. Only underscore is allowed.
2.	It does not return anything hence no return type.	It can return and hence it has a return type. If method does not return anything then the return type is void.

3.	The purpose of constructor is to initialize the data members of the class.	The method is defined to execute the core logic of the class.
4.	The constructor is invoked implicitly at the time of object creation.	The method must be called explicitly using the object name and dot operator.

1.11.1 Properties of Constructor

1. Name of constructor must be the same as the name of the class for which it is being used.
2. The constructor must be declared in the **public** mode.
3. The constructor gets invoked automatically when an object gets created.
4. The constructor should not have any return type. Even a **void** data type should not be written for the constructor.
5. The constructor cannot be used as a member of union or structure.
6. The constructors can have default arguments.
7. The constructor **cannot be inherited**. But the derived class can invoke the constructor of base class.
8. Constructor can make use of **new** or **delete** operators for allocating or releasing memory respectively.
9. Constructor can not be **virtual**.
10. Multiple constructors can be used by the same class.
11. When we declare the constructor explicitly then we must declare the object of that class.

Ex. 1.11.2 : There is no destructor in Java. Justify

Sol. : There are two purposes of destructor - 1. Freeing up the memory allocated for the objects. 2. Cleaning up resources like closing the open file stream. The garbage collection mechanism of Java takes care of these two tasks automatically. Hence there is no need for having destructor in Java.

Ex. 1.11.3 : Define the Rectangle class that contains :

Two double fields x and y that specify the center of the rectangle, the data field width and height. A no-arg constructor that creates the default rectangle with (0,0) for (x,y) and 1 for both width and height.

A parameterized constructor creates a rectangle with the specified x, y, height and width.

A method getArea() that returns the area of the rectangle.

A method getPerimeter() that returns the perimeter of the rectangle.

A method contains(double x, double y) that returns true if the specified point (x, y) is inside this rectangle.

Write a test program that creates two rectangle objects. One with default values and other with user specified values. Test all the methods of the class for both the objects.

Sol. :

```
class Rectangle
{
    double Center_X,Center_Y,width,height;
    double X_Left,X_Right,Y_UP,Y_Down;
    Rectangle() // No Argument constructor
    {
        Center_X=Center_Y=0;
        width=height=1;
    }
    Rectangle(double x,double y,double h,double w)           // Parameterized constructor
    {
        Center_X=x;
        Center_Y=y;
        height=h;
        width=w;
    }
    double getArea()
    {
        return (height*width);
    }
    double getPerimeter()
    {
        return ( 2*(height+width) );
    }
    boolean contains(double x,double y)
    {
        X_Left=( Center_X - (width/2) );
        X_Right=( Center_X + (width/2) );
        Y_UP=(Center_Y + (height/2));
        Y_Down=(Center_Y - (height/2));
        if( (x > X_Left && x < X_Right) && (y < Y_UP && y > Y_Down))
            return true;    //point is inside rect.
        else
            return false;  //point is outside rect.
    }
}

class testclass
{
    public static void main(String args[])
    {
    }
```

```
{
    Rectangle obj1 = new Rectangle();
    Rectangle obj2 = new Rectangle(300,300,100,200);
    System.out.println();
    System.out.println("\t\t Rectangle 1" );
    System.out.println(" Area "+obj1.getArea());
    System.out.println(" Perimeter "+obj1.getPerimeter());
    if(obj1.contains(10,20))
        System.out.println(" Contains (10,20) is inside the Rectangle ");
    else
        System.out.println(" Contains (10,20) is outside the Rectangle ");
    System.out.println("-----");
    System.out.println();
    System.out.println("\t\t Rectangle 2" );
    System.out.println(" Area "+obj2.getArea());
    System.out.println(" Perimeter "+obj2.getPerimeter());
    if(obj2.contains(250,310))
        System.out.println(" Contains (250,310) is inside the Rectangle ");
    else
        System.out.println(" Contains (250,310) is outside the Rectangle ");

}
}
```

Ex. 1.11.4 : State whether any error exists in the following code. If so, correct the error and give output.

```
class Test {
public static void main(String args[]) {
A a = new A();
a.print();
}
}
class A {
String s;
A(String s) {
this.s=s;
}
public void print() {
System.out.println(s);
}
}
```

Sol. : The error at the line where the instance a for the class A is created. Some string must be passed as an argument to the constructor. The corrected code is as given below -

```
class Test {  
    public static void main(String args[]) {  
        A a = new A("JAVA");  
        a.print();  
    }  
}  
  
class A {  
    String s;  
    A(String s) {  
        this.s=s;  
    }  
    public void print() {  
        System.out.println(s);  
    }  
}
```

Ex. 1.11.5 : Declare a class called coordinate to represent 3 dimensional cartesian coordinates (x, y and z). Define following methods : - constructor

- **display, to print values of members**
- **add_coordinates, to add three such coordinate objects to produce a resultant coordinate object**

Generate and handle exception if x, y and z coordinates of the result are zero.

- **main, to show use of above methods.**

Sol. :

```
import java.lang.Exception;  
  
class Cartesian extends Exception  
{  
    double result;  
    double x,y,z;  
    Cartesian()  
    {  
  
    }  
    Cartesian(double x,double y,double z)  
    {  
        this.x=x;  
        this.y=y;  
        this.z=z;  
    }  
  
    void display()  
    {
```

```
        System.out.println("value of x = "+x);
        System.out.println("value of y = "+y);
        System.out.println("value of z = "+z);
    }
    Cartesian add_coordinates(Cartesian obj1,Cartesian obj2,Cartesian obj3) throws
    Cartesian
    {
        Cartesian obj4=new Cartesian();
        obj4.x=obj1.x+obj2.x+obj3.x;
        obj4.y=obj1.y+obj2.y+obj3.y;
        obj4.z=obj1.z+obj2.z+obj3.z;
        if((obj4.x == 0) || (obj4.y == 0) || (obj4.z == 0))
        {
            throw new Cartesian();
        }
        return obj4;
    }

    public static void main(String [] args)
    {
        Cartesian obj=new Cartesian();
        Cartesian obj1=new Cartesian(10,20,30);
        Cartesian obj2=new Cartesian(11,22,33);
        Cartesian obj3=new Cartesian(1,2,3);
        try
        {
            obj=obj.add_coordinates(obj1,obj2,obj3);
            obj.display();
        }
        catch(Cartesian j)
        {
            System.out.println("Exception for Zero Value!!!");
        }
    }
}
```

Output



```
Command Prompt
D:\>javac Cartesian.java
D:\>java Cartesian
value of x = 22.0
value of y = 44.0
value of z = 66.0
D:\>
```

Ex. 1.11.6 : Implement a class Student. A Student has a name and total quiz score. Supply an appropriate constructor and methods getName(), addQuiz(int score), getTotalScore() and getAverageScore(). To Compute the latter, you also need to store number of quizzes that the student took.

Sol. :

```
class Student
{
    private String name;
    private int total_quizScore;
    private int quizCount;
    Student(String n)
    {
        name = n;
        total_quizScore = 0;
        quizCount = 0;
    }
    public String getName()
    {
        return name;
    }
    public void addQuiz(int score)
    {
        total_quizScore = total_quizScore + score;
        quizCount = quizCount + 1;
    }
    public int getTotalScore()
    {
        return total_quizScore;
    }
    public double getAverageScore()
    {
        return total_quizScore / quizCount;
    }
}
class StudentDemo
{
    public static void main(String args[])
    {
        Student obj = new Student("AAA");
        int no_of_quizzes = 5;
        int Score[] = { 50,70,60,55,68 };
        for (int i = 0; i<no_of_quizzes; i++)
```

```
        obj.addQuiz(Score[i]);
    System.out.print("\n Student's Name: " + obj.getName());
    System.out.print("\n Total number of Quizzes: " + no_of_quizzes);
    System.out.print("\n Student's Total Score: " + obj.getTotalScore());
    System.out.print("\n Student's Average Score: " + obj.getAverageScore());
    }
}
```

Output

```
Student's Name: AAA
Total number of Quizzes: 5
Student's Total Score: 303
Student's Average Score: 60.0
```

Ex. 1.11.7 : Write a program to perform the following functions using classes, objects, constructors and destructors where essential. (i) Get as input the marks of 5 students in 5 subjects. (ii) Calculate the total and average. (iii) Print the formatted result on the screen.

AU : May-14, Marks(4+8+4)

Sol. :

```
import java.util.*;
import java.lang.*;
class student
{
    String name;
    int roll_no;
    int sub1,sub2,sub3,sub4,sub5;
    int total;
    float avg;
    student()
    {
        roll_no=0;
        sub1=sub2=sub3=sub4=sub5=0;
        total=0;
        avg=0;
    }
    void getdata()throws NullPointerException
    {
        Scanner sc=new Scanner(System.in);
        System.out.print ("\nEnter Name of Student:");
        name = sc.next();
        System.out.print ("\nEnter Roll No. of Student: ");
        roll_no = sc.nextInt();
        System.out.println ("\nEnter marks of 1st subject: ");
        sub1 = sc.nextInt();
        System.out.println ("\nEnter marks of 2nd subject:");
```

```
        sub2 = sc.nextInt();
        System.out.println ("\nEnter marks of 3rd subject:");
        sub3 = sc.nextInt();
        System.out.println ("\nEnter marks of 4th subject:");
        sub4 = sc.nextInt();
        System.out.println ("\nEnter marks of 5th subject: ");
        sub5 = sc.nextInt();
    }
    void show()
    {
        total=sub1+sub2+sub3+sub4+sub5;
        avg=total/5;

        System.out.println(roll_no+"\t"+name+"\t"+sub1+"\t"+sub2+"\t"+sub3+"\t"+sub4
        +"\t"+sub5+"\t"+total+"\t"+avg+"%");
    }
}
class MainClass
{
    public static void main(String args[])throws NullPointerException
    {
        int i;
        student[] s=new student[5];
        System.out.println("Enter the marks of five students: ");
        for(i=0;i<5;i++)
        {
            s[i]=new student();
            s[i].getdata();
        }

        System.out.println("\nRoll\tName\tSub1\tSub2\tSub3\tSub4\tSub5\tTotal\tAverage"
);
        for(i=0;i<5;i++)
            s[i].show();
    }
}
```

Output

```
Enter the marks of five students:
Enter Name of Student: AAA
Enter Roll No. of Student: 10
Enter marks of 1st subject: 67
Enter marks of 2nd subject: 88
Enter marks of 3rd subject: 90
Enter marks of 4th subject: 55
Enter marks of 5th subject: 66
```

Enter Name of Student: BBB
Enter Roll No. of Student: 20
Enter marks of 1st subject: 99
Enter marks of 2nd subject: 98
Enter marks of 3rd subject: 97
Enter marks of 4th subject: 95
Enter marks of 5th subject: 92
Enter Name of Student: CCC
Enter Roll No. of Student: 30
Enter marks of 1st subject: 64
Enter marks of 2nd subject: 78
Enter marks of 3rd subject: 56
Enter marks of 4th subject: 92
Enter marks of 5th subject: 86
Enter Name of Student: DDD
Enter Roll No. of Student: 40
Enter marks of 1st subject: 76
Enter marks of 2nd subject: 87
Enter marks of 3rd subject: 98
Enter marks of 4th subject: 67
Enter marks of 5th subject: 89
Enter Name of Student: EEE
Enter Roll No. of Student: 50
Enter marks of 1st subject: 67
Enter marks of 2nd subject: 45
Enter marks of 3rd subject: 43
Enter marks of 4th subject: 38
Enter marks of 5th subject: 56

Roll	Name	Sub1	Subj2	Sub3	Sub4	Sub5	Total	Average
10	AAA	67	88	90	55	66	366	73.0%
20	BBB	99	98	97	95	92	481	96.0%
30	CCC	64	78	56	92	86	376	75.0%
40	DDD	76	87	98	67	89	417	83.0%
50	EEE	67	45	43	38	56	249	49.0%

Ex 1.11.8 : Write a Java program to create a student examination database system that prints the mark sheet of students. Input student name, marks in 6 subjects. These marks should be between 0 to 100.

If the average of marks ≥ 80 then prints Grade 'A'.

If the average is < 80 and ≥ 60 then prints Grade 'B'.

If the average is < 60 and ≥ 40 then prints Grade 'C'.

Else prints Grade 'D'.

AU : Dec-18, Marks 13

Sol. :

```
import java.util.*;
import java.lang.*;
class student
{
    String name;
    int roll_no;
    int[] sub=new int[6];
    int total;
    float avg;
    char grade;
    student()
    {
        roll_no=0;
        total=0;
        avg=0;
        grade=' ';
    }
    void getdata()throws NullPointerException
    {
        Scanner sc=new Scanner(System.in);
        System.out.print ("\nEnter Name of Student:");
        name = sc.next();
        System.out.print ("\nEnter Roll No. of Student: ");
        roll_no = sc.nextInt();
        for(int j=0;j<6;j++)
        {
            System.out.println ("\nEnter marks of subject: "+(j+1));
            sub[j] = sc.nextInt();
        }
    }
    void calculate()
    {
        for(int j=0;j<6;j++)
        {
            total=total+sub[j];
        }
        avg=total/6;
        if (avg>=80)
            grade='A';
        else if(avg<80 && avg >= 60)
            grade='B';
    }
}
```

```
        else if(avg<60 && avg >= 40)
            grade='C';
        else
            grade='D';
    }
    void show()
    {
        System.out.print("\n"+roll_no+"\t"+name);
        for(int j=0;j<6;j++)
            System.out.print("\t"+sub[j]);
        System.out.print("\t"+total+"\t"+avg+"%\t"+grade);
    }
}
class MainClass
{
    public static void main(String args[])throws NullPointerException
    {
        int i;
        student[] s=new student[5];
        System.out.println("Enter the marks of five students: ");
        for(i=0;i<5;i++)
        {
            s[i]=new student();
            s[i].getdata();
            s[i].calculate();
        }
        System.out.println("\nRoll\tName\tSub1\tSub2\tSub3\tSub4\tSub5\tSub6\tTotal\tAverage\tGrade");
        for(i=0;i<5;i++)
            s[i].show();
    }
}
```

Output

```
Enter the marks of five students:
Enter Name of Student:AAA
Enter Roll No. of Student: 10
Enter marks of subject: 1
55
Enter marks of subject: 2
44
Enter marks of subject: 3
55
```

Enter marks of subject: 4
44
Enter marks of subject: 5
55
Enter marks of subject: 6
44
Enter Name of Student:BBB
Enter Roll No. of Student: 20
Enter marks of subject: 1
11
Enter marks of subject: 2
22
Enter marks of subject: 3
11
Enter marks of subject: 4
23
Enter marks of subject: 5
12
Enter marks of subject: 6
21
Enter Name of Student:CCC
Enter Roll No. of Student: 30
Enter marks of subject: 1
88
Enter marks of subject: 2
99
Enter marks of subject: 3
89
Enter marks of subject: 4
87
Enter marks of subject: 5
86
Enter marks of subject: 6
89
Enter Name of Student:DDD
Enter Roll No. of Student: 40
Enter marks of subject: 1
54
Enter marks of subject: 2
56
Enter marks of subject: 3
55
Enter marks of subject: 4
66
Enter marks of subject: 5
53
Enter marks of subject: 6
50
Enter Name of Student:EEE
Enter Roll No. of Student: 50

Enter marks of subject: 1

77

Enter marks of subject: 2

70

Enter marks of subject: 3

71

Enter marks of subject: 4

72

Enter marks of subject: 5

60

Enter marks of subject: 6

65

Roll	Name	Sub1	Subj2	Sub3	Sub4	Sub5	Sub6	Total	Average	Grade
10	AAA	55	44	55	44	55	44	297	49.0%	C
20	BBB	11	22	11	23	12	21	100	16.0%	D
30	CCC	88	99	89	87	86	89	538	89.0%	A
40	DDD	54	56	55	66	53	50	334	55.0%	C
50	EEE	77	70	71	72	60	65	415	69.0%	B

Review Questions

1. What is constructor ? What is the use of new method ?
2. How the objects are constructed ? Explain the constructor overloading with an example.
3. Write detailed discussion on constructors.
4. Discuss the usage of constructor with an example using Java.
5. What is constructor ? Explain with an example in detail.
6. What is constructor in java ? Why constructor does not have return type in java ? Explain it with proper example.
7. What is the difference between constructor and method ? Discuss with example

AU : Dec.-14, Marks 6

1.12 Access Specifiers

AU : IT : Dec.-10, 11, CSE : Dec.-12, Marks 8

Access modifiers control access to data fields, methods, and classes. There are three modifiers used in Java -

- public
- private
- default modifier

public allows classes, methods and data fields accessible from any class

private allows classes, methods and data fields accessible only from within the own class.

If public or private is not used then by default the classes, methods, data fields are assessable by any class in the same package. This is called **package-private** or **package-access**. A package is essentially grouping of classes.

For example :

```
package Test;
public class class1
{
    public int a;
    int b;
    private int c;
    public void fun1() {
    }
    void fun2() {
    }
    private void fun3() {
    }
}
public class class2
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;//allowed
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//allowed
        obj.fun3();//error:cannot access
    }
}
```

```
package another_Test
public class class3
{
    void My_method() {
        class1 obj=new class1();
        obj.a;//allowed
        obj.b;// error:cannot access
        obj.c;//error:cannot access
        obj.fun1();//allowed
        obj.fun2();//error:cannot access
        obj.fun3();//error:cannot access
    }
}
```

In above example,

- We have created two packages are created namely - **Test** and **another_Test**.
- Inside the package **Test** there are two classes defined - **class1** and **class2**
- Inside the package **another_Test** there is only one class defined and i.e. **class3**.
- There are three data fields - **a**, **b** and **c**. The data field **a** is declared as public, **b** is defined as default and **c** is defined as private.
- The variable **a** and method **fun1()** both are accessible from the classes **class2** and **class3**(even if it is in another package). This is because they are declared as **public**.
- The variable **b** and method **fun2()** both are accessible from **class2** because **class2** lies in the same package. But they are not accessible from **class3** because **class3** is defined in another package.

- The variable **c** and method **fun30()** both are not accessible from any of the class, because they are declared as private.

Protected mode is another access specifier which is used in inheritance. The **protected** mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.

The effect of access specifiers for class, subclass or package is enlisted below -

Specifier	class	subclass	package
private	Yes	-	-
protected	Yes	Yes	Yes
public	Yes	Yes	Yes

For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it. Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass can not access it.

Review Questions

1. Write short note on access specifiers in Java.
2. What are access specifiers ? Discuss them in context of Java.

AU : IT : Dec.-10, Marks 6

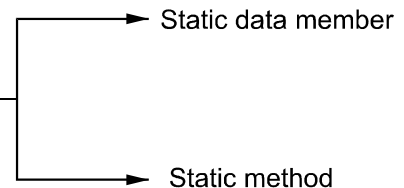
AU : IT : Dec.-11, Marks 8, CSE : Dec.-12

1.13 Static Members

AU : CSE : Dec.-10, IT : May-11, 12, 13, Marks 8

The static members can be **static data member** or **static method**.

The static members are those Static members members which can be accessed without using object. Following program illustrates the use of static members.



Java Program[StaticProg.java]

```

/*****

```

Program to introduce the use of the static method and static variables

```

*****/

```

```

class StaticProg

```

```

{
    static int a=10;

```

```
static void fun(int b)
{

    System.out.println("b= "+b);
    System.out.println("a= "+a);

}
}
class AnotherClass
{
    public static void main(String[] args)
    {
        System.out.println("a= "+StaticProg.a);
        StaticProg.fun(20);

    }
}
```

Output

```
a= 10
b= 20
a= 10
```

Program Explanation

In above program, we have declared one static variable **a** and a static member function **fun()**. These static members are declared in one class **StaticProg**. Then we have written one more class in which the **main** method is defined. This class is named as **AnotherClass**. From this class the static members of class **StaticProg** are accessed *without using any object*. Note that we have simple used a syntax

ClassName.staticMember

Hence by using **StaticProg.a** and **StaticProg.fun** we can access static members

Points to remember about static members

- In Java the **main** is always static method.
- The static methods must can access only static data.
- The static method can call only the static method and can not call a non static method.
- The static method can not refer to **this** pointer.
- The static method can not refer to **super** method.

Review Questions

1. Give the syntax for the static method and its initialization.

AU : CSE : Dec.-10, Marks 4

2. Discuss the working and meaning of the “static” modifier with suitable example.

AU : IT : May-11, Marks 8

3. Explain about static variables and static methods used in java, with examples.

AU : IT : May-12, Marks 8

4. Why do we need static members and how to access them ? Explain it with clear example.

AU : IT: May-13, Marks 8

1.14 Arrays

AU : May-14, Marks 16

- **Definition :** Array is a collection of similar type of elements.
- Using arrays the elements that are having the same data type can be grouped together.
- If we **use bunch of variables** instead of arrays, then we have to use large number of variables.
- Declaring and handling large number of variables is very inconvenient for the developers.
- There are two types of arrays -
 - One dimensional arrays
 - Two dimensional arrays

1.14.1 One Dimensional Array

- Array is a collection of similar type of elements. Thus grouping of similar kind of elements is possible using arrays.
- Typically arrays are written along with the size of them.

- The **syntax** of declaring array is -

`data_type array_name[];`

and to allocate the memory -

`array_name=new data_type[size];`

where *array_name* represent name of the array, **new** is a keyword used to allocate the memory for arrays, *data_type* specifies the data type of array elements and *size* represent the size of an array. For example:

`a=new int[10];`

[Note that in C/C++ this declaration is as good as `int a[10]`]

- After this declaration the array **a** will be created as follows

Array a[10]

0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---

- That means after the above mentioned declaration of array, all the elements of array will get initialized to zero. Note that, always, while declaring the array in Java, we make use of the keyword **new**, and thus we actually make use of dynamic memory allocation.
- Therefore arrays are **allocated dynamically** in Java. Let us discuss on simple program based on arrays.

Java Program [SampleArray.Java]

```
/*  
This is a Java program which makes use of arrays  
*/  
class SampleArray  
{  
    public static void main(String args[])  
    {  
        int a[];  
        a=new int[10];  
        System.out.println("\tStoring the numbers in array");  
        a[0]=1;  
        a[1]=2;  
        a[2]=3;  
        a[3]=4;  
        a[4]=5;  
        a[5]=6;  
        a[6]=7;  
        a[7]=8;  
        a[8]=9;  
        a[9]=10;  
        System.out.println("The element at a[5] is: "+a[5]);  
    }  
}
```

Output

Storing the numbers in array
The element at a[5] is : 6

Program explanation

In above program we have created an array of 10 elements and stored some numbers in that array using the array index. The array that we have created in above program virtually should look like this -

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	4	5	6	7	8	9	10

The **System.out.println** statement is for printing the value present at a[5]. We can modify the above program a little bit and i.e instead of writing

```
int a[];  
a=new int[10];
```

these two separate statements we can combine them together and write it as -

```
int a[]=new int[10];
```

That means the declaration and initialization is done simultaneously.

Another way of initialization of array is

```
int a[] = {1,2,3,4,5,6,7,8,9,10};
```

That means, as many number of elements that are present in the curly brackets, that will be the size of an array. In other words there are total 10 elements in the array and hence size of array will be 10.

1.14.2 Two Dimensional Arrays

- The two dimensional arrays are the arrays in which elements are stored in rows as well as in columns.
- For example

		Columns		
		0	1	2
Rows	0	10	20	30
	1	40	50	60
	2	70	80	90

- The two dimensional array can be declared and initialized as follows

Syntax

```
data_type array_name=new data_type[size];
```

For example

```
int a=new int[10];
```

Let us straight away write a Java program that is using two dimensional arrays -

Java Program [Sample2DArray.java]

```
/*  
This is a Java program which makes use of 2D arrays  
*/  
class Sample2DArray
```

```
{
public static void main(String args[])
{
    int a[][]=new int[3][3];
    int k=0;
    System.out.println("\tStoring the numbers in array");
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            a[i][j]=k+10;
            k=k+10;
        }
    }
    System.out.println("You have stored...");
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            System.out.print(" "+a[i][j]);
        }
        System.out.println();
    }
}
```

Output

```
Storing the numbers in array
You have stored...
10 20 30
40 50 60
70 80 90
```

The typical application of two dimensional arrays is performing matrix operations. Let have some simple Java program in which various matrix operations are performed.

Ex. 1.14.1 : Write a Java program to perform matrix addition , subtraction and multiplication.

Java Program[Matrix.java]

```
/*
This is a Java program which performs matrix operations
*/
class Matrix
{
    public static void main(String args[])
    throws java.io.IOException
```

```
{
int a[][]={
    {10,20,30},
    {40,50,60},
    {70,80,90}
};
int b[][]={
    {1,2,3},
    {4,5,6},
    {7,8,9}
};
int c[][]=new int [3][3];
char choice;
System.out.println("\n Main Menu");
System.out.println("1. Addition");
System.out.println("2. Subtraction");
System.out.println("3. Multiplication");
System.out.println("Enter your choice");
choice=(char)System.in.read();
switch(choice)
{
case '1':for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            c[i][j]=a[i][j]+b[i][j];
        }
    }
    System.out.println("The Addition is...");
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            System.out.print(" "+c[i][j]);
        }
        System.out.println();
    }
    break;
case '2':for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            c[i][j]=a[i][j]-b[i][j];
        }
    }
    System.out.println("The Subtraction is...");
```

```
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(" "+c[i][j]);
            }
            System.out.println();
        }
        break;
case '3':for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        c[i][j]=0;
        for(int k=0;k<3;k++)
        {
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
        }
    }
}
System.out.println("The Multiplication is...");
for(int i=0;i<3;i++)
{
    for(int j=0;j<3;j++)
    {
        System.out.print(" "+c[i][j]);
    }
    System.out.println();
}
break;
}
}
}
```

Output (Run1)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
1
The Addition is...
11 22 33
44 55 66
77 88 99
```

Output (Run2)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
2
The Subtraction is...
9 18 27
36 45 54
63 72 81
```

Output (Run3)

```
Main Menu
1. Addition
2. Subtraction
3. Multiplication
Enter your choice
3
The Multiplication is...
300 360 420
660 810 960
1020 1260 1500
```

Ex. 1.14.2 : Write a Java Program to implement Linear Search.**Sol. :**

```
public class LinearSearch
{
    public static void main(String args[]) {
        int[] a={1,5,-2,8,7,11,40,32};
        System.out.println("The element is at location "+Find(a,7));
        System.out.println("The element is at location "+Find(a,11));
    }
    public static int Find(int[] a,int key)
    {
        int i;
        for(i=0;i<a.length;i++)
        {
            if(a[i]==key)
                return i+1;
        }
        return -1;
    }
}
```

Output

The element is at location 5
The element is at location 6

arraycopy Command

Using array copy we can copy entire an to another array. The arraycopy command is applicable to one dimensional array. The syntax of arraycopy command is -

```
public static void arraycopy(Object src,
                             int srcPos,
                             Object dest,
                             int destPos,
                             int length)
```

But it can be extended to copy two dimensional array as well. Following example illustrates this idea.

Ex. 1.14.3 : Write a program for transposition of matrix using array copy command.

Sol. :

```
Java Program[ArrayTranspose.java]
public class ArrayTranspose
{
    public static void main(String args[])
    {
        int A[][] = {{1,2,3},{4,5,6},{7,8,9}};
        int B[][] = new int[3][3];
        int i,j,From[],To[];
        From = new int[9];
        To = new int[9];
        int k=0;
        System.out.println("The original matrix is...");
        for(i=0;i<3;i++)
        {
            for(j=0;j<3;j++)
                System.out.print(A[i][j]+" ");
            System.out.println();
        }
        for(i=0,k=0;i<3;i++)
            for(j=0;j<3;j++)
                From[k++] = A[i][j]; //making it 1-D
        for(i=0,j=0,k=0;i<9;i++)
        {
            System.arraycopy(From,j,To,i,1); //copying it to another 1-D array
            j=j+3;
            if(j>=9) //first row completion
                j=++k;
        }
    }
}
```

```
System.out.println("The Transposed Matrix is...");
for(i=0,k=0;i<3;i++)
{
    for(j=0;j<3;j++)
    {
        B[i][j]=To[k];
        k=k+1;
        System.out.print(B[i][j]+" ");
    }
    System.out.println();
}
}
```

Output

```
The original matrix is...
1 2 3
4 5 6
7 8 9
The Transposed Matrix is...
1 4 7
2 5 8
3 6 9
```

Ex. 1.14.4 : Write a program to find the transpose of a given matrix.

Sol. :

```
import java.util.Scanner;
class Transpose
{
    public static void main(String args[])

    {
        int m, n, i, j;
        Scanner in = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns of matrix");
        m = in.nextInt();
        n = in.nextInt();

        int A[][] = new int[m][n]; //original matrix
        int B[][] = new int[n][m]; //transpose matrix
        System.out.println("Enter the elements of matrix");
        for (i = 0; i < m; i++)
            for (j = 0; j < n; j++)
                A[i][j] = in.nextInt(); //enter matrix elements throu keyboard
        for (i=0;i<m;i++)

        {
```

```
        for ( j = 0 ; j < n ; j++ )
            B[j][i] = A[i][j];
    }
    System.out.println("Transpose of matrix is ");

    for (i=0;i< n;i++ )
    {
        for (j=0;j<m;j++ )
            System.out.print(" " +B[i][j]);

        System.out.print("\n");
    }
}
```

Output

Enter the number of rows and columns of matrix

3 4

Enter the elements of matrix

1 2 3 4

5 6 7 8

9 10 11 12

Transpose of matrix is

1 5 9

2 6 10

3 7 11

4 8 12

Ex. 1.14.5 : Write a program to perform the following functions on a given matrix (i) Find the row and column sum (ii) Interchange the rows and columns

AU : May-14, Marks 16

Sol. :

(i)

class RowColSum

```
{
    public static void main(String args[])throws java.io.IOException
    {
        int a[][]={
            {10,20,30},
            {40,50,60},
            {70,80,90}
        };

        int rsum[]=new int[3];
        int csum[]=new int[3];
        int i,j;
```

```

/* Sum of Rows */
for(i=0;i<3;i++)
{
    rsum[i]=0;
    for(j=0;j<3;j++)
        rsum[i]=rsum[i]+a[i][j];
}
/* Sum of Column */
for(i=0;i<3;i++)
{
    csum[i]=0;
    for(j=0;j<3;j++)
        csum[i]=csum[i]+a[j][i];
}
System.out.println("The sum or rows and columns of the matrix is :");
for(i=0;i<3;i++)
{
    for(j=0;j<3;j++)
        System.out.print(" "+a[i][j]);/*displaying each row*/
    System.out.print(" = "+rsum[i]);/*displaying row sum*/
    System.out.println("");
}
System.out.println("-----");
for(j=0;j<3;j++)
{
    System.out.print(" "+csum[j]);/*displaying col sum*/
}
}
}

```

Output

The sum or rows and columns of the matrix is :

10 20 30 = 60

40 50 60 = 150

70 80 90 = 240

120 180

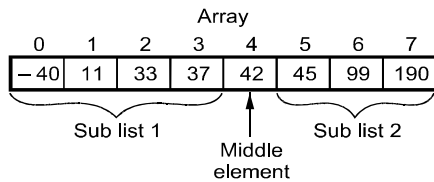
(ii) Refer Example 1.14.4.

Ex. 1.14.6 : Explain binary search method with necessary illustration and write a Java program to implement it.

Sol. : The necessity of this method is that all the elements should be sorted. So let us take an array of sorted elements.

array

0	1	2	3	4	5	6	7
- 40	11	33	37	42	45	99	100



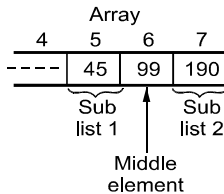
Step 1 : Now the key element which is to be searched is = 99 key = 99.

Step 2 : Find the middle element of the array. Compare it with the key

if middle $\stackrel{?}{=}$ key

i.e. if 42 $\stackrel{?}{=}$ 99

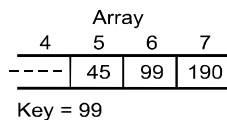
if 42 < 99 search the sublist 2



Now handle only sublist 2. Again divide it, find mid of sublist 2

if middle $\stackrel{?}{=}$ key

i.e. if 99 $\stackrel{?}{=}$ 99



So match is found at 7th position of array i.e. at array [6]

Thus by binary search method we can find the element 99 present in the given list at array [6]th location.

Java Program[BinSearch.java]

```
public class BinSearch {
    public static void main(String args[]) {
        int[] a = {10,20,30,40,50,60};
        int key = 40;
        Find(a,0,5,key);
    }
    public static void Find(int[] a,int low,int high,int key)
    {
        int mid;
        if(low > high)
        {
            System.out.println("Error!!The element is not present in the list");
            return;
        }
    }
}
```

```
mid=(low+high)/2;
if(key==a[mid])
    System.out.println("\n The element is present at location "+(mid+1));
else if(key<a[mid])
    Find(a,low,mid-1,key);
else if(key>a[mid])
    Find(a,mid+1,high,key);
}
}
```

Output

```
F:\test>javac BinSearch.java
```

```
F:\test>java BinSearch
```

```
The element is present at location 4
```

Ex. 1.14.7 : Develop a Java program to find a smallest number in the given array by creating a one dimensional array and two dimensional array using new operator.

AU : Dec.- 19, Marks 15

Sol. :

```
import java.util.*;
public class SmallestElement
{
    public static void main(String[] args)
    {
        int rows;
        int columns;
        Scanner scanner = new Scanner (System.in);
        System.out.println("\t\t ***** TWO DIMENSIONAL ARRAY *****");
        System.out.println("Enter number of rows: ");
        rows = scanner.nextInt();
        System.out.println("Enter number of columns: ");
        columns = scanner.nextInt();
        //creating two dimensional array using new operator
        int[][] matrix = new int [rows][columns];
        //storing the elements in the two dimensional array
        System.out.println("Enter matrix numbers: ");
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                matrix[i][j] = scanner.nextInt();
            }
        }
    }
}
```

```
// Displaying entered matrix
System.out.println("Matrix as entered");
for (int i = 0; i < matrix.length; i++)
{
    System.out.println();
    for (int j = 0; j < matrix[i].length; j++)
    {
        System.out.print(matrix[i][j] + " ");
    }
}
System.out.println();
//finding smallest element from 2D array
findMin(matrix);
int size;
System.out.println("\t\t ***** ONE DIMENSIONAL ARRAY *****");
System.out.println("Enter total number of elements: ");
size = scanner.nextInt();
//creating one dimensional array using new operator
int[] a = new int[size];
//storing the elements in one dimensional array
System.out.println("Enter the elements: ");
for(int i = 0; i < size; i++)
{
    a[i] = scanner.nextInt();
}
//displaying one dimensional array
System.out.println("The one dimensional array which you have entered is");
for(int i = 0; i < size; i++)
{
    System.out.println(a[i] + " ");
}
System.out.println();
//finding smallest element from array
findSmall(a);
}
private static void findMin(int[][] matrix)
{
    int minNum = matrix[0][0];
    for (int i = 0; i < matrix.length; i++)
    {
        for (int j = 0; j < matrix[i].length; j++)
        {
            if(minNum > matrix[i][j])
            {
                minNum = matrix[i][j];
            }
        }
    }
}
```

```
        }  
    }  
    }  
    System.out.println("Smallest number: " + minNum);  
}  
private static void findSmall(int[] a)  
{  
    int smallNum = a[0];  
    for(int i = 0; i < a.length; i++)  
    {  
        if(smallNum > a[i])  
        {  
            smallNum = a[i];  
        }  
    }  
    System.out.println("Smallest number: " + smallNum);  
}  
}
```

Output

***** TWO DIMENSIONAL ARRAY *****

Enter number of rows:

3

Enter number of columns:

3

Enter matrix numbers:

5 2 4

3 1 6

9 8 7

Matrix as entered

5 2 4

3 1 6

9 8 7

Smallest number: 1

***** ONE DIMENSIONAL ARRAY *****

Enter total number of elements:

5

Enter the elements:

30 10 20 50 40

The one dimensional array which you have entered is

30

10

20

50

40

Smallest number : 10

Ex. 1.14.8 : Define Java classes of your own without using any library classes to represent linked lists of integers. Provide it with methods that can be used to reverse a list and to append two lists.

AU : May 19, Marks 15

Sol. :

```
public class SinglyLinkedList{
    Node head;

    static class Node {
        private int data;
        private Node next;

        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }

    public void insert(Node node) {
        if (head == null) {
            head = node;
        } else {
            Node temp = head;
            while (temp.next != null)
                temp = temp.next;
            temp.next = node;
        }
    }

    public void display(Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.format("%d ", temp.data);
            temp = temp.next;
        }
        System.out.println();
    }

    // Reverse linkedlist using this function
    public static Node reverse(Node currentNode)
    {
        // For first node, previousNode will be null
        Node previousNode=null;
        Node nextNode;
        while(currentNode!=null)
```

```
        {
            nextNode=currentNode.next;
            // reversing the link
            currentNode.next=previousNode;
            // moving ahead currentNode and previousNode by 1 node
            previousNode=currentNode;
            currentNode=nextNode;
        }
        return previousNode;
    }
    public static Node Append(Node head1,Node head2) {
        Node previousNode =null;
        Node temp = head1;
        while (temp != null) {
            previousNode = temp;
            temp = temp.next;//reaching to last node of first list
        }

        previousNode.next = head2;//appending second list
        return head1;
    }
}
```

```
public static void main(String[] args) {
    SinglyLinkedList list = new SinglyLinkedList();
    // Creating a linked list
    Node head=new Node(10);
    list.insert(head);
    list.insert(new Node(20));
    list.insert(new Node(30));
    list.insert(new Node(40));
    list.insert(new Node(50));

    list.display(head);
    //Reversing LinkedList
    Node reverseHead=reverse(head);
    System.out.println("After reversing");
    list.display(reverseHead);
    //Creating First LinkedList
    Node head1=new Node(1);
    list.insert(head1);
    list.insert(new Node(2));
    list.insert(new Node(3));
    list.insert(new Node(4));
    list.insert(new Node(5));
}
```

```
        System.out.println("The first linked list is");
        list.display(head1);
        //Creating Second LinkedList
        Node head2=new Node(6);
        list.insert(head2);
        list.insert(new Node(7));
        list.insert(new Node(8));
        list.insert(new Node(9));
        list.insert(new Node(10));
        System.out.println("The second linked list is");
        list.display(head2);
        System.out.println("The concatenated linked list is");
        Node head3 = Append(head1,head2);
        list.display(head3);
    }
```

1.15 Packages

AU : IT : May-12, 13, 14, Dec.-11, 12, CSE : Dec.-13, Marks 16

1.15.1 Defining Package

- **Purpose** : In Java, packages are used to achieve the **code reusability**. That means, the classes from other programs can be easily used by a class without physically copying it to current location.
- **Definition** : Package is a mechanism in which variety of classes and interfaces can be grouped together.
- **Importance** : Following are the benefits of organizing classes into packages-
 1. The classes defined in the packages of other program can be easily **reused**.
 2. Two classes from two different packages can have the **same name**. By using the package name the particular class can be referred.
 3. Packages provide the complete **separation between** the two phases- **design and coding**. In the design phase, we can design the classes and decide their relationship and then during the coding phase we can develop the Java code for corresponding classes and can group them in several packages according to their relationship with each other.
 4. Using packages it is possible to **hide the classes**. This feature prevents other programs to access the classes that are developed for internal purpose only.
- Creating a package is very simple. Just include the command package at the beginning of the program. The syntax for declaring the package is

package name_of_package

- This package statement defines the name space in which the classes are stored. If we omit the package then the default classes are put in the package that has no name.

- Basically Java creates a directory and the name of this directory becomes the package name.
For example - In your program, if you declare the package as -

package My_Package;

then we must create the directory name **My_Package** in the current working directory and the required classes must be stored in that directory. Note that the name of the package and the name of the directory must be the same. This name is case sensitive.

- We can create hierarchy of packages. For instance if you save the required class files in the subfolder **MyPkg3** and the path for this subfolder is C:\MyPkg1\MyPkg2\MyPkg3 then the declaration for the package in your java program will be -

package MyPkg1.MyPkg2.MyPkg3;

1.15.2 Creating and Accessing Package

In this section we discuss **how to develop a program** which makes use the classes from other package.

Step 1 : Create a folder named **My_Package**.

Step 2 : Create one class which contains two methods. We will store this class in a file named **A.java**. This file will be stored in a folder **My_Package**. The code for this class will be as follows-

Java Program[A.java]

```
package My_Package; //include this package at the beginning
```

```
public class A
```

```
{
```

```
int a;
```

```
public void set_val(int n)
```

```
{
```

```
    a=n;
```

```
}
```

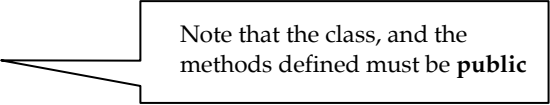
```
public void display()
```

```
{
```

```
    System.out.println("The value of a is: "+a);
```

```
}
```

```
}
```



Note that the class, and the methods defined must be **public**

Note that this class contains two methods namely- **set_val** and **display**. By the **set_val** method we can assign some value to a variable. The **display** function displays this stored value.

Step 3 : Now we will write another java program named **PackageDemo.java**. This program will use the methods defined in class **A**. This source file is also stored in the subdirectory

My_Package. The java code for this file is -

Java Program[PackageDemo.java]

```
import My_Package.A; //The java class A is referenced here by import statement
class PackageDemo
{
    public static void main(String args[]) throws NoClassDefFoundError
    {
        A obj=new A(); //creating an object of class A
        obj.set_val(10); //Using the object of class A, the methods present
        obj.display(); //in class A are accessed
    }
}
```

Step 4 : Now, open the command prompt and issue the following commands in order to run the package programs

D:\>set CLASSPATH=.;D:\;

Setting the class path

D:\>cd My_Package

D:\My_Package>javac A.java

Creating the classes A.class and PackageDemo.class files

D:\My_Package>javac PackageDemo.java

D:\My_Package>java PackageDemo

The value of a is: 10

Running the test program which uses the class A.class stored in another file

D:\My_Package>

1.15.3 CLASSPATH

The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. Thus if the required packages is in the current working directory then it will found. Otherwise you can specify the directory path setting the **CLASSPATH** statement. For instance- if the package name **My_Package** is present at prompt D:\> then we can specify

```
set CLASSPATH=.;D\;
```

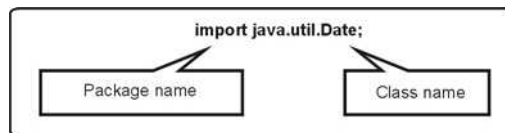
```
D:\>cd My_Package
```

```
D:\My_Package\> Now you can execute the required class files from this location
```

1.15.4 Importing Package

- All the standard classes in Java are stored in named packages.
- There is no standard class present in Java which is unnamed. But it is always complicated to write the class name using a long sequence of packages containing **dot** operator. Hence the **import** statement is needed.
- The import statement can be written at the beginning of the Java program, using the keyword **import**.
- There are two ways of accessing the classes stored in the core package.

1. Method 1 : We import the java package class using the keyword **import**. Suppose we want to use the Date class stored in the **java.util** package then we can write the import statement at the beginning of the program. It is as follows -



2. Method 2 : There are some situations in which we want to make use of several classes stored in a package. Then we can write it as

import java.util.*

Here * means any class in the corresponding package.

Ex. 1.15.1 : Write a java program to maintain the books details like BookId, accession number; book name, author, publication in books package and keep the journal details such as journal Id; journal name; in journal package in main class use these two packages details for staff and student classes and display the books and journals information as requested by the user.

AU : IT : May-13, Marks 16

Sol. :

Step 1 : Create a folder named MyLibrary and save following two Java programs namely **books.java** and **journals.java** within it.

books.java

```
package MyLibrary;  
import java.io.*;
```

```
public class books  
{  
    int BookId,AccessionNumber;  
    String BookName, Author, Publication;
```

```
DataInputStream input=new DataInputStream(System.in);
public void ReadData()
{
    try
    {
        System.out.println("Enter BookId: ");
        BookId=Integer.parseInt(input.readLine());

        System.out.println("Enter Accession Number: ");
        AccessionNumber=Integer.parseInt(input.readLine());

        System.out.println("Enter Book Name: ");
        BookName=input.readLine();

        System.out.println("Enter Author Name: ");
        Author=input.readLine();

        System.out.println("Enter Publication: ");
        Publication=input.readLine();
    }
    catch(Exception e)
    {
        System.out.println("You have entered wrong data!!!");
    }
}

public void Display()
{
    System.out.println("BookId: "+BookId);
    System.out.println("Accession Number: "+AccessionNumber);
    System.out.println("Book Name: "+BookName);
    System.out.println("Author Name: "+Author);
    System.out.println("Publication: "+Publication);
}
}
```

journals.java

```
package MyLibrary;
import java.io.*;
```

```
public class journals
{
    int JournalId;
    String JournalName;
    DataInputStream input=new DataInputStream(System.in);
    public void ReadData()
    {
```

```
try
{
    System.out.println("Enter Journal Id: ");
    JournalId=Integer.parseInt(input.readLine());

    System.out.println("Enter Journal Name: ");
    JournalName=input.readLine();
}
catch(Exception e)
{
    System.out.println("You have entered wrong data!!!");
}
}
public void Display()
{
    System.out.println("Journal Id: "+JournalId);
    System.out.println("Journal Name: "+JournalName);
}
}
```

Step 2 : Compile the above two programs using following javac command

D:\MyLibrary>javac books.java

D:\MyLibrary>javac journals.java

Due to above commands the **books.class** and **journals.java** get generated within the folder MyLibrary/

Step 3 : Come out of MyLibrary directory and create following program -

MainClass.java

```
import MyLibrary.*;
```

```
import java.io.*;
```

```
class Staff {
    int StaffId;
    String StaffName;
    String Department;
    DataInputStream input=new DataInputStream(System.in);
    void ReadData()
    {
        try
        {
            System.out.println("Enter Staff Id");
            StaffId= Integer.parseInt(input.readLine());

            System.out.println("Enter Staff Name");
            StaffName= input.readLine();
        }
    }
}
```

```
        System.out.println("Enter Department of Staff");
        Department= input.readLine();
    }
    catch(Exception e)
    {
        System.out.println("You have entered wrong data!!!");
    }
}
void Display()
{
    System.out.println("Staff Id: "+StaffId);
    System.out.println("Staff Name: "+StaffName);
    System.out.println("Department: "+Department);
}
}
class Student {
    int RollNumber;
    String StudentName;
    DataInputStream input=new DataInputStream(System.in);
    void ReadData()
    {
        try
        {
            System.out.println("Enter Student Roll Number");
            RollNumber= Integer.parseInt(input.readLine());

            System.out.println("Enter Student Name");
            StudentName= input.readLine();
        }
        catch(Exception e)
        {
            System.out.println("You have entered wrong data!!!");
        }
    }
    void Display()
    {
        System.out.println("Student Roll Number: "+RollNumber);
        System.out.println("Student Name: "+StudentName);
    }
}
public class MainClass
{
    public static void main(String[] args)throws IOException
    {
        Staff Staff_obj=new Staff();
        Student Student_obj=new Student();
    }
}
```

```
books Book_obj=new books();
journals journals_obj=new journals();
int choice;
char ans;
DataInputStream input=new DataInputStream(System.in);
do
{
    System.out.println("Enter your choice: ");
    System.out.print("\n 1.Staff \n 2.Student");
    choice= Integer.parseInt(input.readLine());
    switch (choice)
    {
        case 1:System.out.println("\n\t Enter The data for Staff...");
            Staff_obj.ReadData();
            Book_obj.ReadData();
            journals_obj.ReadData();
            System.out.println("\n\t Displaying Record...");
            Staff_obj.Display();
            Book_obj.Display();
            journals_obj.Display();

            break;
        case 2:System.out.println("\n\t Enter The data for Student...");
            Student_obj.ReadData();
            Book_obj.ReadData();
            journals_obj.ReadData();
            System.out.println("\n\t Displaying Record...");
            Student_obj.Display();
            Book_obj.Display();
            journals_obj.Display();
            break;
    }
    System.out.println("\n Do you want to continue?");
    String s1=input.readLine();
    ans=s1.charAt(0);
}while(ans=='y');
}
```

Step 3 : Execute above program using the following command

D:\>javac MainClass.java

D:\>java MainClass

Enter your choice:

1.Staff
2.Student
1

Enter The data for Staff...

Enter Staff Id
1
Enter Staff Name
Archana
Enter Department of Staff
Computer
Enter BookId:
101
Enter Accession Number:
1010
Enter Book Name:
Software Engineering
Enter Author Name:
Puntambekar
Enter Publication:
Technical
Enter Journal Id:
5001
Enter Journal Name:
IEEE

Displaying Record...

Staff Id: 1
Staff Name: Archana
Department: Computer
BookId: 101
Accession Number: 1010
Book Name: Software Engineering
Author Name: Puntambekar
Publication: Technical
Journal Id: 5001
Journal Name: IEEE

Do you want to continue?
n

Review Questions

1. Give a brief overview of java packages. Write necessary code snippets.

AU : IT : Dec.-11, 12, Marks 16

2. What are packages ? Explain how will you import a package in Java. Give example.

AU : IT : May-12, IT : Dec.-12, Marks 16

3. What is meant by package ? How it is created and implemented in JAVA.

AU : CSE : Dec.-13, Marks 8

4. With suitable examples explain how packages can be created, imported and used. Also elaborate on its scope.

AU : May-14, Marks 16

1.16 JavaDoc Comments

Javadoc is a convenient, standard way to document your Java code. Javadoc is actually a **special format of comments**. There are some utilities that read the comments, and then generate HTML document based on the comments. HTML files give us the convenience of hyperlinks from one document to another.

There are **two types** of Javadoc comments -

- Class level comments
- Member level comments

The class level comments describe the purpose of classes and member level comments describe the purpose of members.

The Javadoc comments start with `/**` and end with `*/` For example
`/** This is a Javadoc comment statement*/`

1.16.1 Class Level Comments

The class level comments provide the description and purpose of the classes. For example –

```
/**
 * @author XYZ
 * The Employee class contains salary of each employee the organisation
 */
public class Employee
{
    //Employee class code
}
```

1.16.2 Member Level Comments

The member level comments describe the data members, methods and constructors used in particular class. In this type of comments special tags are used to describe the things. The most commonly used tags are -

Tags	Description
@author	This can be used in the class level comment. It describes the name of the author who is writing the document/
@param	This tag describes the name of the method used in the class
@return	This tag describes the return tag of the method.
@throws	This tag describes the exertion that can be thrown by the method.
@exception	This tag describes the exception

The example of member level comments for the Employee class is as shown below -

```
/**
 * @author XYZ
 * The Employee class contains salary of each employee the organisation
 */
public class Employee
{
    /**
     * Employee information for knowing the salary
     */
    private int amtSalary;
    /**
     * The constructor defined to initialise the salary amount
     */
    public Employee()
    {
        this.salary=0;
    }
    /**
     * This method returns the salary amount of particular employee
     * @return int
     */
    public int getAmtSalary()
    {
        return salary;
    }
    /**
     * @param int No_of_Days_Worked is for total number of working days
     * @param int Payscale is for payment scale as per the designation of the employee
     */
    public void setAmtSalary(int No_of_Days_Worked, int Payscale)
    {
        this.salary=No_of_Days_Worked* Payscale;
    }
}
```

Along with the above mentioned tags some HTML tags can also be included in Javadoc comments. For example we can use <table> tags in Javadoc. Also we can include special tags like < in the Javadoc. Some words like true, false and null can also be included in Javadoc.

Two Marks Questions with Answers**Q.1 Mention some of the separators used in Java programming .****AU : IT : Dec.-12**

Ans. : Some of the separators that are used in Java are as given below –

Name	Description
.	The period or dot is used to select a field or method from an object. It is also used to separate out the package name from sub-packages or class names.
:	The colon is used in loops after the loop label.
,	The comma is used to separate out variables or to separate out the parameters.
;	The semicolon is used to terminate the statement in Java.
()	The round opening and closing parenthesis are used to enclose the parameters of the method definition or call. It adjusts the precedence in arithmetic expression. For type casting purpose also these parenthesis are used.
{}	For enclosing the function or the class block or for array initialization these parentheses are used.
[]	These parenthesis are used in array declaration.

Q.2 How dynamic initialization of variables is achieved in java ?**AU : IT : Dec.-12**

Ans. : The variables in Java are allowed to get initialized at run time. Such type of initialization is called dynamic initialization.

For example

```
double a=5.0;
```

```
double b=20.0
```

```
double c=Math.sqrt((a+b));
```

Here the expression is evaluated at run time and its square root value is calculated and then the variable c is initialized dynamically.

Q.3 What is the output of the main method in the given code ?**AU : CSE : Dec.-13**

```
public static void main(String[] args)
{
    screen.write(sum());
}
static int sum()
{
    int A=12;
    int B=13;
    return =A+B;
}
```

Ans. : It will generate error at the statement `return = A+B` as “Illegal start of expression”.

Q.4 What is Java Virtual Machine ?**AU : IT : May-13, CSE : Dec.-13**

Ans. : Java Virtual Machine is a set of software and program components which takes the **byte code** as an input and generates output corresponding to underlying operating system. Due to Virtual Machine Java can achieve platform independence feature.

Q.5 What are the features of Java ?**AU : Dec.-13**

Ans. :

- (1) Java is simple to implement.
- (2) Java is platform independent.
- (3) It is an object oriented programming language.
- (4) It is a robust programming language.
- (5) Java is designed for distributed system.

Q.6 Define objects and classes in Java.**AU : CSE : Dec.-10, 18**

Ans. : An object is an instance of a class. The objects represent the real world entity. The objects are used to provide a practical basis for the real world. Each class is a collection of data and the functions that manipulate the data. The data components of class are called data fields and the function components of the class are called member functions or methods.

Q.7 Why are classes important in OO technology ?

Ans. : Classes bind together the relative data and methods in a cohesive unit. Due to this arrangement, only certain methods are allowed to access the corresponding data. Secondly, if any modification is needed, then the class can be viewed as one module and the changes made in one class does not spoil rest of the code. Moreover, finding error from such a source code becomes simple.

Hence use of class is very important thing in OO technology.

Q.8 What is the difference between object and class ?

Ans. :

Following are some differences between the class and the object –

Sr. No.	Class	Object
1.	For a single class there can be any number of objects. For example - If we define the class as River then Ganga, Yamuna, Narmada can be the objects of the class River.	There are many objects that can be created from one class. These objects make use of the methods and attributes defined by the belonging class.

2.	The scope of the class is persistent throughout the program.	The objects can be created and destroyed as per the requirements.
3.	The class cannot be initialized with some property values.	We can assign some property values to the objects.
4.	A class has unique name.	Various objects having different names can be created for the same class.

Q.9 What is the difference between structure and class ?

Ans. :

Sr. No.	Structure	Class
1.	By default the members of structure are public.	By default the members of class are private.
2.	The structure can not be inherited.	The class can be inherited.
3.	The structures do not require constructors.	The classes require constructors for initializing the objects.
4.	A structure contains only data members.	A class contains the data as well as the function members.

Q.10 What is the difference between static and non static variables ?

AU : IT : Dec.-10

Ans. : A static variable is shared among all instances of class, whereas a non static variable (also called as instance variable) is specific to a single instance of that class.

Q.11 Define encapsulation.

AU : IT : Dec.-11

Ans. : Encapsulation means binding of data and method together in a single entity called class.

Q.12 What is an abstract class ?

AU : IT : Dec.-11,12, May-12

Ans. : When apply inheritance, the class becomes more and more specific as we move down. Sometimes a situation may occur that the superclass becomes more general and less specific. Such a class lists out only common features of other classes. Such a super class is called as **abstract class**.

Q.13 Define class. Give example.

AU : Comp : Dec.-11, IT : May-12

Ans. : Each class is a collection of data and the functions that manipulate the data. The data components of class are called data fields and the function components of the class are called member functions or methods.

For example

```
class Customer
{
    int ID;
    String Name; //Data Field
    Customer() //Constructor
    { }
    double withdraw_money() //method
    { ...}
}
```

Q.14 What is the default access to a member in a class ?

Ans. : If public or private is not used then by default the classes, methods and data fields are accessible by any class in the same package. This is called **package-private or package-access**.

Q.15 What are the benefits of encapsulation ? Should abstractions be user centric or developer-centric ?

Ans. : Due to encapsulation the corresponding data and the methods get bound together by means of class. The data inside the class is accessible by the function in the same class. It is normally not accessible from outside component. Thus the unwanted access to the data can be protected.

The abstraction should be user centric. While developing the system using the OO principles it is important to focus the user and not the developer.

Q.16 How would you declare an object of type animal named lion that takes a weight of 500 and length of 45 as parameters ?

Ans. :

```
public class Animal
{
    int weight,length;
    Animal(int wt,int len)
    {
        weight=wt;
        length=len;
    }
    void Show()
    {
        System.out.println("\n The weight of animal is: "+weight);
        System.out.println("\n The length of animal is: "+length);
    }
}
class AnimalMain
{
```

```
public static void main(String args[])
{
    Animal Lion=new Animal(500,45);
    Lion.Show();
}
```

Output

The weight of animal is: 500

The length of animal is: 45

Q.17 What is meant by private access specifier ?

Ans. : The private access specifier allows classes, methods and data fields accessible only from within the own class. The functions outside the class cannot access the data members or the member functions.

Q.18 What do you mean by instance variable ?

AU : CSE : May-12

Ans. : An Object is an instance of a class. The variables that the object contains are called instance variables. For example consider a class Student

```
class Student
{
    int RollNo;
    char name[10];
}
Student S; //Creation of object of type Student class.
S= new Student();
```

If we create an object of the class Student say S, then using this object the variables RollNo and name can be accessed. These variables that belong to object S are then called as instance variables. Thus the Instance variables are any variables, without "static" field modifier, that are defined within the class body.

Q.19 Define constructor.

AU : CSE : Dec.-12

Ans. : Constructor is a specialized method used for initializing the objects. The name of this method and the name of the class must be the same. The constructor is invoked whenever an object of its associated class is created.

Q.20 What is Static in Java ?

AU : IT : May-13

Ans. : In java, static is a member of a class that is not associated with an instance of a class. If there is a need for a variable to be common to all the objects of a single java class, then the static keyword should be used in the variable declaration.

Q.21 What is the difference between a constructor and a method ?**AU : IT : May-13****Ans. :**

Constructor	Method
Name of the constructor must be the same as the name of the class.	There is no such restriction on the name of the method.
Constructor does not have any return type.	Method has return type. If the method does not return anything then it must have void data type.
Constructors are chained, in the sense that they are called in some specific order.	Methods are not chained. That means - invoking of methods depend upon the logic of the program.

Q.22 What are key characteristics of objects ?**AU : CSE : May-13****Ans. :**

1. Object is an instance of a class
2. Objects are runtime entities.
3. Using object of a class the member variable and member functions of a class can be accesses.

Q.23 What is meant by parameter passing constructors? Give example.**AU : CSE : Dec.-13**

Ans. : The constructor methods to which the parameter are passed are called parameter passing constructors

Example -

```
Rectangle(int h,int w)
{
    height=h;
    weight=w;
}
```

Q.24 What is package ?

Ans. : Package represent a collection of classes,methods and interfaces. The name of the package must be written as the first statement in the Java source program. The syntax of specifying the package in the Java program is

`package name_of_package`

Due to package the systematic arrangement of classes is possible. All standard classes in Java are stored in named packages.

Q.25 Mention the necessity for import statement.**AU : IT : May-12**

Ans. : The import statement is used to refer the classes and methods that are present in particular package. This statement is written at the beginning of any Java program. For example -

```
import java.io.*
```

This statement allows to use the useful functionalities for performing input and output operations.

Q.26 Enumerate two situations in which static methods are used.**AU : May-14**

Ans. :

- (1) The situation in which object of belonging class is not created and we want to use the method of that class, then the method must be static.
- (2) For calling another static method, one such static method is used.
- (3) For accessing static data the static method must be used.

Q.27 How is constant defined in Java ?**AU : May-15**

Ans. : Refer section 1.7.2.

Q.28 List any four JavaDoc comments.**AU : CSE : Dec.-11**

Ans. :

Tags	Description
@author	This can be used in the class level comment. It describes the name of the author who is writing the document/
@param	This tag describes the name of the method used in the class
@return	This tag describes the return tag of the method.
@throws	This tag describes the exertion that can be thrown by the method.
@exception	This tag describes the exception

Q.29 What is the need for javadoc multiline comments ?

Ans. : Javadoc multiline comments can be used to document all java source code. Comments follow a standard format consisting of a description followed by block tags. The first sentence of the description should be clear and concise as it is used in the summary of the API item.

Q.30 List any four packages in java and highlight their features.**Ans. :**

Package	Feature
java.io	This package provide useful functionalities for performing input and output operations through data streams and through file system.
java.net	This package is for providing useful classes and interfaces for networking applications which are used in sockets and URL.
java.lang	This package provides core classes and interfaces used in design of Java language.
java.util	Some commonly used utilities such as random number generation, event model, date, time and some of the system properties are represented using java util package.

Q.31 What is API package ?**Ans. :** • The API packages are application programming interface packages used by java.

- These packages include important classes and interfaces which are used by the programmer while developing the java applications.
- For example java.io, java.util, java.net and so on are the commonly used API.

Q.32 Explain any three uses of package.**Ans. :**

1. The classes defined in the packages of other program can be easily reused.
2. Two classes from two different packages can have the same name. By using the package name the particular class can be referred.
3. Packages provide the complete separation between the two phases- design and coding.

Q.33 Explain the term CLASSPATH.**Ans. :** The packages are nothing but the directories. For locating the specified package the java run time system makes use of current working directory as its starting point. This directory path is called CLASSPATH.**Q.34 What are the ways of importing the java packages ?****OR Write the syntax for importing packages in a Java source file and give an example.****AU : May-19****Ans. :** The **import** statement can be written at the beginning of the Java program, using the keyword import. For example -

```
import java.io.File
or
import java.io.*
```

Either a class name can be specified explicitly or a * can be used which indicated that the Java compiler should import the entire package.

Q.35 Define access specifier.**AU : Dec.-18, 19****Ans. :** Access Specifier is used to set the accessibility of class members. There are three access specifiers in Java - (1) Public (2) Private (3) Protected.

Q.36 What is Javadoc ?**AU : Dec.-19**

Ans. : The Javadoc is a standard way to comment the java code. It has a special format of commenting the java code.

Q.37 Can Java source file be saved using a name other than the class name. Justify.**AU : May-19**

Ans. : Yes, we can save the java source file using a name other than the class name. But we should compile the program with file name and should run the program with the class name in which the main method exist. And there must not be any public class defined inside this java source file. For example, consider following Java code which is saved using the file name **test.java**

```
class A
{
    public static void main(String args[])
    {
        System.out.println("Class A");
    }
}
class B
{
    public static void main(String args[])
    {
        System.out.println("Class B");
    }
}
```

Output

Step 1 : Compile the above program using following command

javac test.java

Step 2 : Now execute the above program using

java A

The output will be

Class A

Step 3 : If you execute the above code as

java B

The output will be

ClassB

Q.38 What are inline function ? Give examples.**AU : May-19**

Ans. : Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call.

The inline function is one of the important feature of C++.

In java there is no inline function



Notes

UNIT-II

2

Inheritance

Syllabus

Inheritance - Super classes - sub classes - Protected members - constructors in sub classes - the Object class - abstract classes and methods - final methods and classes.

Contents

2.1	Inheritance	
2.2	Concept of Super and Sub Classes	
2.3	Types of Inheritance.....	May-11, 12, Dec.-19 Marks 16
2.4	Protected Members	
2.5	Implementation of Different Types of Inheritance	May-13, 15, Dec.-14, Marks 16
2.6	Constructors in Sub Classes.....	May-19
2.7	Method Overloading and Method Overriding	CSE : May-12, CSE, Dec.-13, Marks 16
2.8	Use of keyword Super.....	Dec.-14, Marks 8
2.9	Object Class.....	
2.10	Abstract Classes and Methods	CSE : Dec.-10, 12, 13,19, May-13, 19 IT : May-13, Marks 8
2.11	Final Methods and Classes.....	Dec.-13, Marks 8
2.12	Finalize() Method	Dec.-18, Marks 7

2.1 Inheritance

- **Definition :** Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.
- The inheritance can be achieved by incorporating the definition of one class into another using the keyword **extends**.

Advantages of Inheritance

One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses.

1. **Reusability :** The base class code can be used by derived class without any need to rewrite the code.
2. **Extensibility :** The base class logic can be extended in the derived classes.
3. **Data hiding :** Base class can decide to keep some data private so that it cannot be altered by the derived class.
4. **Overriding :** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class

2.2 Concept of Super and Sub Classes

- The inheritance is a mechanism in which the child class is derived from a parent class.
- This derivation is using the keyword **extends**.
- The parent class is called **base** class and child class is called **derived class**.
- For example

```
Class A                ← This is Base class
{
    ...
}
Class B extends A      ← This is Derived class
{
    ... // uses properties of A
}
```

- Inheritance is represented diagrammatically as follows

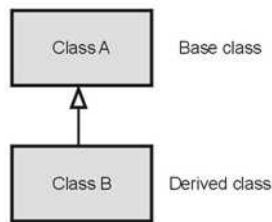


Fig. 2.2.1

2.3 Types of Inheritance

AU : May-11, 12, Dec.-19, Marks 16

1. Single inheritance :

- In single inheritance there is one parent per derived class. This is the most common form of inheritance.
- The simple program for such inheritance is -

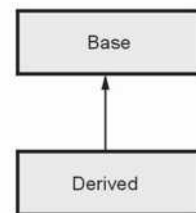


Fig. 2.3.1

2. Multiple inheritance :

In multiple inheritance the derived class is derived from more than one base class.

Java **does not implement multiple inheritance** directly but it makes use of the concept called interfaces to implement the multiple inheritance.

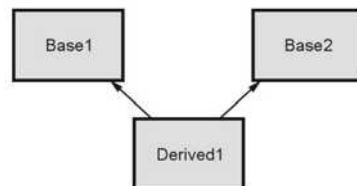


Fig. 2.3.2

3. Multilevel inheritance :

When a derived class is derived from a base class which itself is a derived class then that type of inheritance is called multilevel inheritance.

For example - If class A is a base class and class B is another class which is derived from A, similarly there is another class C being derived from class B then such a derivation leads to multilevel inheritance.

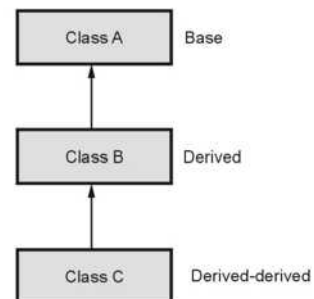


Fig. 2.3.3

4. Hybrid inheritance :

When two or more types of inheritances are combined together then it forms the hybrid inheritance. The following Fig. 2.3.4 represents the typical scenario of hybrid inheritance.

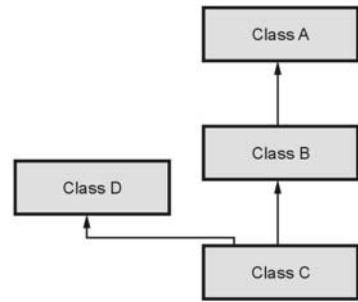


Fig. 2.3.4

Review Questions

1. Explain in detail as how inheritance is supported in Java with necessary examples
2. Give an elaborate discussion on inheritance.
3. Explain in detail about various types of inheritance in java with neat diagram.

AU : May-11, Marks 8

AU : May-12, Marks 16

AU : Dec.-19, Marks 13

2.4 Protected Members

- Protected mode is another access specifier which is used in inheritance.
- The protected mode allows accessing the members to all the classes and subclasses in the same package as well as to the subclasses in other package. But the non subclasses in other package can not access the protected members.
- For example, if some variable is declared as protected, then the class itself can access it, its subclass can access it, and any class in the same package can also access it.
- Similarly if the variable is declared as private then that variable is accessible by that class only and its subclass or package can not access it.

2.5 Implementation of Different Types of Inheritance

AU : May-13, 15, Dec.-14, 18, Marks 16

2.5.1 Single Inheritance

- The class which is inherited is called the **base class** or the **superclass** and the class that does the inheriting is called the **derived class** or the **subclass**.
- The method defined in base class can be used in derived class. There is no need to redefine the method in derived class. Thus inheritance promotes **software reuse**.
- The subclass can be defined as follows -
class nameofSubclass **extends** superclass

```

{
    variable declarations
    method declarations
}
  
```

```
}
```

Note that the keyword **extends** represents that the properties of superclass are extended to the subclass. Thus the subclass will now have both the properties of its own class and the properties that are inherited from superclass.

- Following is a simple Java program that illustrates the concept of single inheritance -

Java Program[InheritDemo1.java]

```
class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}

class B extends A //extending the base class A
{
    int b;
    void set_b(int i)
    {
        b=i;
    }
    void show_b()
    {
        System.out.println("The value of b= "+b);
    }
    void mul()
    {
        int c;
        c=a*b;
        System.out.println(" The value of c= "+c);
    }
}

class InheritDemo1
{
    public static void main(String args[])
    {
        A obj_A=new A();
        B obj_B=new B();
        obj_B.set_a(10);
        obj_B.set_b(20);
```

Note that object of class B is accessing method of class A

```

        obj_B.show_a();
        obj_B.show_b();
        obj_B.mul();
    }
}

```

Output

```
F:\test>javac InheritDemo1.java
```

```
F:\test>java InheritDemo1
```

```
The value of a= 10
```

```
The value of b= 20
```

```
The value of c= 200
```

Program Explanation

In above program, we have created two classes : class **A** and **B**. In **class A** we have declared one integer **a** and in **class B** we have declared an integer **b**. There are two methods defined in class **A** namely: **set_a** and **show_a**. Similarly, in class **B** there are two methods defined namely: **set_b** and **show_b**. As the name suggests these methods are for setting the values and for showing the contents.

In the class **InheriDemo1**, in the **main** function we have created two objects for class **A** and class **B**. The program allows us to access the variable **a** (belonging to class **A**) and the variable **b** (belonging to class **B**) using the object for class **B**. Thus it is said that class **B** has inherited value of variable **a**.

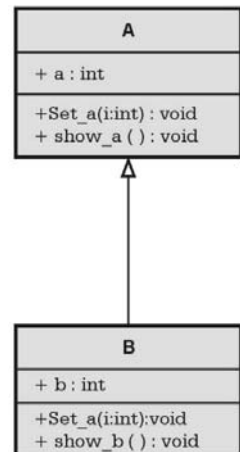


Fig. 2.5.1 Single inheritance

The class **A** is called **Superclass** and the class **B** is called **Subclass**. A **Superclass** is also called as **parent class** or **base class**. Similarly, the **Subclass** is also called as **child class** or **derived class**.

Ex. 2.5.1 : Write a java program to calculate area of rectangle using the single inheritance.

Sol. :

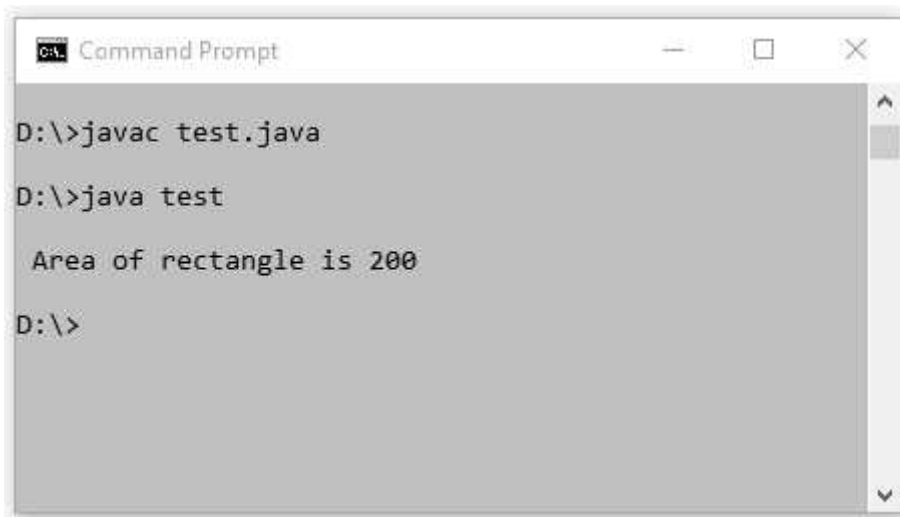
```

class Shape
{
    int len,br;
    void setValues(int a,int b)
    {

```

```
        len=a;
        br=b;
    }
}
class Rectangle extends Shape
{
    int area()
    {
        return len*br;
    }
}
class test
{
    public static void main(String args[])
    {
        Rectangle r=new Rectangle();
        r.setValues(10,20);
        System.out.println("\n Area of rectangle is "+r.area());
    }
}
```

Output



```
Command Prompt

D:\>javac test.java
D:\>java test

Area of rectangle is 200

D:\>
```

2.5.2 Multilevel Inheritance

The multilevel inheritance is a kind of inheritance in which the derived class itself derives the subclasses further.

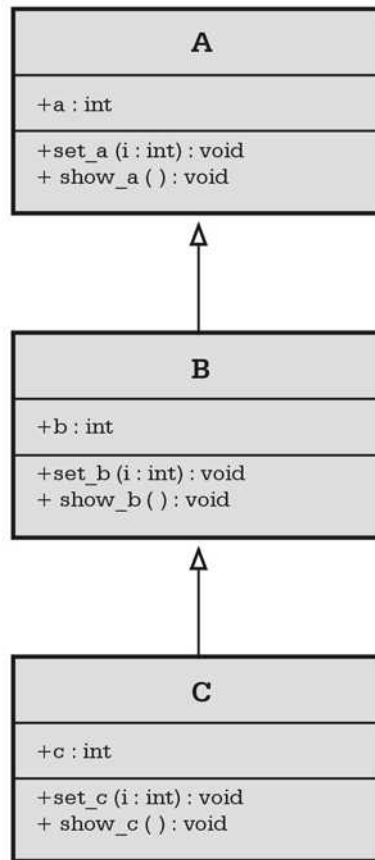


Fig. 2.5.2 Multilevel inheritance

In the following program, we have created a base class A from which the subclass B is derived. There is a class C which is derived from class B. In the function main we can access any of the field in the class hierarchy by creating the object of class C.

Java Program[MultiLvlInherit.java]

```
class A
{
    int a;
    void set_a(int i)
    {
        a=i;
    }
    void show_a()
    {
        System.out.println("The value of a= "+a);
    }
}
```

```
}  
class B extends A  
{  
    int b;  
    void set_b(int i)  
    {  
        b=i;  
    }  
    void show_b()  
    {  
        System.out.println("The value of b= "+b);  
    }  
}  
class C extends B  
{  
    int c;  
    void set_c(int i)  
    {  
        c=i;  
    }  
    void show_c()  
    {  
        System.out.println("The value of c= "+c);  
    }  
    void mul()  
    {  
        int ans;  
        ans=a*b*c;  
        System.out.println(" The value of ans= "+ans);  
    }  
}  
  
}
```

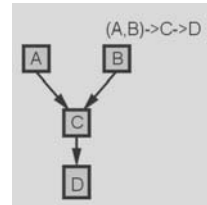
```
class MultiLvlInherit  
{  
    public static void main(String args[])  
    {  
        A obj_A=new A();  
        B obj_B=new B();  
        C obj_C=new C();  
        obj_C.set_a(10);  
        obj_C.set_b(20);  
        obj_C.set_c(30);  
        obj_C.show_a();  
        obj_C.show_b();  
        obj_C.show_c();  
    }  
}
```

```
    obj_C.mul();  
  }  
}
```

Output

```
The value of a= 10  
The value of b= 20  
The value of c= 30  
The value of ans= 6000
```

Ex. 2.5.2 : Explain single level and multiple inheritances in java. Write a program to demonstrate combination of both types of inheritance as shown in Fig. 2.5.3. i.e hybrid inheritance.

**Fig. 2.5.3**

Sol. :

```
class A  
{  
    int a=10;  
}  
interface B  
{  
    int b=20;  
}  
class C extends A implements B  
{  
    int c;  
    int mul()  
    {  
        c=a*b;  
        return c;  
    }  
}  
class D extends C  
{  
    void sum()  
    {  
        System.out.println("Adding all the three variables");  
        int d=a+b+mul();  
        System.out.println(d);  
    }  
}
```

```
}

}
class Demo1
{
    public static void main(String[] args)
    {
        C obj1=new C();
        D obj2=new D();
        System.out.println("Multiplying two variables");
        System.out.println(obj1.mul());
        obj2.sum();
    }
}
```

Output

```
Multiplying two variables
200
Adding all the three variables
230
```

Ex. 2.5.3 : Declare a class called employee having employee_id and employee_name as members. Extend class employee to have a subclass called salary having designation and monthly_salary as members. Define following :

- **Required constructors.**
- **A method to find and display all details of employees drawing salary more than 20000/-.**
- **Method main for creating an array for storing these details given as command line arguments and showing usage of above methods.**

Sol. :

```
class employee
{
    int employee_id;
    String employee_name;
}
class salary extends employee //derived class
{
    String designation;
    double monthly_salary;
    salary()    //default constructor
    {}
    salary(int employee_id,String employee_name,String designation,
           double monthly_salary) //parameterised construct.
```

```
{
    this.employee_id = employee_id;
    this.employee_name = employee_name;
    this.designation = designation;
    this.monthly_salary = monthly_salary;
}
void fun(String emp[][]) //Function displaying salary>20000
{
    for(int i=0;i<emp.length;i++)
    {
        if( (Double.parseDouble(emp[i][3]) ) > 20000 )
        {
            System.out.println("\nEmployee Drawing Salary More than ` 20000/- : ");
            System.out.println("Id = "+emp[i][0]);
            System.out.println("Name = "+emp[i][1]);
            System.out.println("Designation = "+emp[i][2]);
            System.out.println("Salary = "+emp[i][3]);
            System.out.println("-----");
        }
    }
}
}
public class employeetest
{
    public static void main(String args[])
    {
        salary obj[] = new salary[5];
        String Details[][];
        obj[0] = new salary(1,"AAA","Accountant",12000);
        obj[1] = new salary(2,"BBB","Manager",30000);
        obj[2] = new salary(3,"CCC","Executive",2000);
        obj[3] = new salary(4,"DDD","CEO",50000);
        obj[4] = new salary();
        try
        {
            obj[4].employee_id = Integer.parseInt(args[0]);
            obj[4].employee_name = args[1];
            obj[4].designation = args[2];
            obj[4].monthly_salary = Double.parseDouble(args[3]);
        }
        catch(NumberFormatException e)
        {
            System.out.println("Exception : "+ e);
        }
        Details = new String[obj.length][4];
        for(int i=0;i<obj.length;i++)
```

```
{
    Details[i][0] = String.valueOf(obj[i].employee_id);
    Details[i][1] = obj[i].employee_name;
    Details[i][2] = obj[i].designation;
    Details[i][3] = String.valueOf(obj[i].monthly_salary);
}
obj[4].fun(Details);
}
```

Output

```
E:\test>javac employeetest.java
```

```
E:\test>java employeetest 5 EEE Manager 20000
```

Employee Drawing Salary More than ` 20000/- :

Id = 2

Name = BBB

Designation = Manager

Salary = 30000.0

Employee Drawing Salary More than Rs. 20000/- :

Id = 4

Name = DDD

Designation = CEO

Salary = 50000.0

Ex.2.5.4 : Create a Java class Shape with constructor to initialize the one parameter “dimension”. Now create three subclasses of Shape with following methods :

(i) n”Circle” with methods to calculate the area and circumference of the circle with dimension as radius.

(ii) “Square” with methods to calculate the area and length of diagonal of square with dimension as length of one side.

(iii) “Sphere” with methods to calculate the volume and surface area of sphere with dimension as radius of the sphere . Write appropriate main method to create object of each class and test every method

AU : May-15, Marks 16

Sol. :

```
class Shape
{
    double dimension;
```

```
Shape()
{
    dimension=0;
}

}
class Circle extends Shape
{
    Circle(double r)
    {
        dimension = r;
    }
    void display()
    {
        System.out.println("-----");
        System.out.println("The radius of circle is: "+dimension);
    }
    double area()
    {
        System.out.print("Area Of Circle : ");
        return (3.14*dimension*dimension);
    }
    double circum()
    {
        System.out.print("Circumference Of Circle : ");
        return (2*3.14*dimension);
    }
}
class Square extends Shape
{
    Square(double d)
    {
        dimension = d;
    }
    void display()
    {
        System.out.println("-----");
        System.out.println("The side of square is: "+dimension);
    }
    double area()
    {
        System.out.print("Area Of Square : ");
        return (dimension*dimension);
    }
    double LenDiagonal()
    {
```

```
        System.out.print("Length of Diagonal of Square : ");
        return (dimension*Math.sqrt(2));
    }
}
class Sphere extends Shape
{
    Sphere(double r)
    {
        dimension = r;
    }
    void display()
    {
        System.out.println("-----");
        System.out.println("The radius of sphere is: "+dimension);
    }
    double area()
    {
        System.out.print("Surface Area Of Sphere : ");
        return (4*3.14*dimension*dimension);
    }
    double volume()
    {
        System.out.print("Volume of Sphere : ");
        return ((4/3)*3.14*dimension*dimension*dimension);
    }
}

class InheritanceDemo
{
    public static void main(String args[])
    {
        Circle cir = new Circle(10);
        cir.display();
        System.out.println(cir.area());
        System.out.println(cir.circum());
        Square sq = new Square(10);
        sq.display();
        System.out.println(sq.area());
        System.out.println(sq.LenDiagonal());
        Sphere sph = new Sphere(10);
        sph.display();
        System.out.println(sph.area());
        System.out.println(sph.volume());
    }
}
```

Output

```

-----
The radius of circle is: 10.0
Area Of Circle : 314.0
Circumference Of Circle : 62.800000000000004
-----
The side of square is: 10.0
Area Of Square : 100.0
Length of Diagonal of Square : 14.142135623730951
-----
The radius of sphere is: 10.0
Surface Area Of Sphere : 1256.0
Volume of Sphere : 3140.0

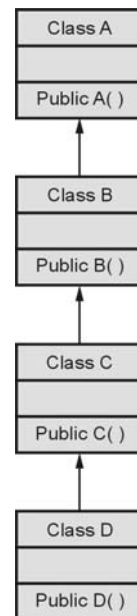
```

Review Questions

1. What is inheritance ? Write a program for inheriting a class.
2. What is inheritance ? With diagrammatic illustrations and Java programs illustrate different types of inheritance. Give self explanatory comments in your program.
3. Define Inheritance. With diagrammatic illustration and java programs illustrate the different types of inheritance with an example.

AU : May-13, Marks 8**AU : Dec.-14, Marks 16****AU : Dec.-18, Marks 7****2.6 Constructors in Sub Classes****AU : May-19**

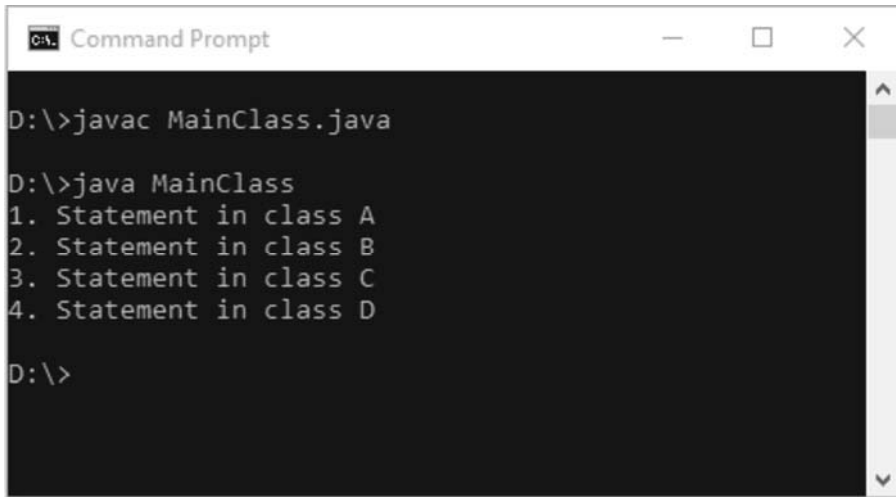
- A constructor invokes its superclass's constructor explicitly and if such explicit call to superclass's constructor is not given then compiler makes the call using **super()** as a first statement in constructor.
- Normally a superclass's constructor is called before the subclass's constructor. This is called **constructor chaining**.
- Thus the calling of constructor occurs from top to down.

**Fig. 2.6.1 Constructor call**

The constructor chaining can be illustrated by following Java program.

Java Program[MainClass.java]

```
class A
{
    public A()//constructor defined
    {
        System.out.println("1. Statement in class A");
    }
}
class B extends A //Derived child of A
{
    public B() //constructor defined
    {
        System.out.println("2. Statement in class B");
    }
}
class C extends B //derived class of class B
{
    public C()//constructor defined
    {
        System.out.println("3. Statement in class C");
    }
}
class D extends C //derived class of class C
{
    public D()//constructor defined
    {
        System.out.println("4. Statement in class D");
    }
}
class MainClass
{
    public static void main(String args[])
    {
        D obj=new D(); //calling constructor using derived class object
    }
}
```

Output

```
CA Command Prompt
D:\>javac MainClass.java

D:\>java MainClass
1. Statement in class A
2. Statement in class B
3. Statement in class C
4. Statement in class D

D:\>
```

Program Explanation : In above program, we have implemented **multilevel inheritance**. The base class is A from which class B is derived. Then from class B the class C is derived and then from class C the class D is derived. In each of these classes, the constructor is defined.

In the main method, the object for last derived class D is created. On creation of this object, the constructor of class A is invoked, then the constructor of class B is invoked, then constructor of class C and finally the constructor of class D is invoked.

Review Question

1. Explain hierarchical and multi-level inheritances supported by Java and demonstrate the execution order of constructors in these types.

AU : May-19, Marks 13**2.7 Method Overloading and Method Overriding****CSE : May-12, CSE, Dec.-13, Marks 16****2.7.1 Overriding**

Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

The method of superclass which gets modified in subclass has the same name and type signature. The overridden method must be called from the subclass. Consider following Java Program, in which the method(named as **fun**) in which **a** is assigned with some value is modified in the derived class. When an overridden method is called from within a subclass, it will always refer to the version of that method re-defined by the subclass. The version of the method defined by the superclass will be hidden.

Java Program[OverrideDemo.java]

```
class A
{
    int a=0;
    void fun(int i)
```

```
        {
            this.a=i;
        }
    }
    class B extends A
    {
        int b;
        void fun(int i)
        {

            int c;
            b=20;
            super.fun(i+5);
            System.out.println("value of a:"+a);
            System.out.println("value of b:"+b);
            c=a*b;
            System.out.println("The value of c= "+c);
        }
    }
}
class OverrideDemo
{
    public static void main(String args[])
    {
        B obj_B=new B();
        obj_B.fun(10);//function re-defined in derived class
    }
}
```

Output

```
F:\test>javac OverrideDemo.java
F:\test>java OverrideDemo
value of a:15
value of b:20
The value of c= 300
```

Program Explanation

In above program, there are two class - class A and class B. The class A acts as a superclass and the class B acts as a subclass. In class A, a method **fun** is defined in which the variable **a** is assigned with some value. In the derived class B, we use the same function name **fun** in which, we make use of **super** keyword to access the variable **a** and then it is multiplied by **b** and the result of multiplication will be printed.

Rules to be followed for method overriding

1. The private data fields in superclass are not accessible to the outside class. Hence the method of superclass using the private data field cannot be overridden by the subclass.

2. An instance method can be overridden only if it is accessible. Hence private method can not be overridden.
3. The static method can be inherited but can not be overridden.
4. Method overriding occurs only when the name of the two methods and their type signatures is same.

2.7.2 Methods Overloading

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

For example :

```
int sum(int a,int b);  
double sum(double a,double b);  
int sum(int a,int b,int c);
```

That means, by overloading mechanism, we can handle different number of parameters or different types of parameter by having the same method name.

Following Java program explains the concept overloading -

Java Program [OverloadingDemo.java]

```
public class OverloadingDemo {  
    public static void main(String args[]) {  
        System.out.println("Sum of two integers");  
        Sum(10,20); <----- line A  
        System.out.println("Sum of two double numbers");  
        Sum(10.5,20.4); <----- line B  
        System.out.println("Sum of three integers");  
        Sum(10,20,30); <----- line C  
    }  
    public static void Sum(int num1,int num2)  
    {  
        int ans;  
        ans=num1+num2;  
        System.out.println(ans);  
    }  
    public static void Sum(double num1,double num2)  
    {  
        double ans;  
        ans=num1+num2;  
        System.out.println(ans);  
    }  
    public static void Sum(int num1,int num2,int num3)
```

```
{  
    int ans;  
    ans=num1+num2+num3;  
    System.out.println(ans);  
}  
}
```

Output

```
F:\test>javac OverloadingDemo.java
```

```
F:\test>java OverloadingDemo
```

```
Sum of two integers
```

```
30
```

```
Sum of two double numbers
```

```
30.9
```

```
Sum of three integers
```

```
60
```

Program Explanation

In above program, we have used three different methods possessing the same name. Note that,

on line A

We have invoked a method to which two integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30 which is actually the addition of 10 and 20.

on line B

We have invoked a method to which two double parameters are passed. Then compiler automatically selects the definition **public static void Sum(double num1,double num2)** to fulfil the call. Hence we get the output as 30.9 which is actually the addition of 10.5 and 20.4.

on line C

We have invoked a method to which the three integer parameters are passed. Then compiler automatically selects the definition **public static void Sum(int num1,int num2,int num3)** to fulfil the call. Hence we get the output as 60 which is actually the addition of 10, 20 and 30.

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
The method overloading occurs at compile time .	The method overriding occurs at the run time or execution time .
In case of method overloading different number of parameters can be passed to the function.	In function overriding the number of parameters that are passed to the function are the same .

The overloaded functions may have different return types .	In method overriding all the methods will have the same return type .
Method overloading is performed within a class.	Method overriding is normally performed between two classes that have inheritance relationship.

Review Questions

1. Differentiate between method overloading and method overriding. Explain both with an example.

AU : CSE : May-12, Marks 16

2. What is meant by overriding method ? Give example

AU : CSE, Dec.-13, Marks 5

2.8 Use of keyword Super

AU : Dec.-14, Marks 8

Super is a keyword used to access the immediate parent class from subclass.

There are three ways by which the keyword **super** is used.

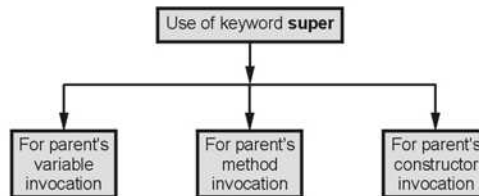


Fig. 2.8.1 Use of super

Let us understand these uses of keyword super with illustrative Java programs.

1. The **super()** is used to invoke the class method of immediate parent class.

Java Program[B.java]

```

class A
{
    int x=10;
}
class B extends A
{
    int x=20;
    void display()
    {
        System.out.println(super.x);
    }
    public static void main(String args[])
    {
        B obj=new B();
        obj.display();
    }
}
  
```

```
    }  
}
```

Output

10

Program Explanation : In above program class A is a immediate parent class of class B. Both the class A and Class B has variables **x**. In class A, the value of **x** variable is 10 and in class B the value of variable **x** is 20. In display function if we would write

```
System.out.println(x);
```

The output will be **20** but if we user **super.x** then the variable **x** of class **A** will be referred. Hence the output is **10**.

2. The super() is used to access the class variable of immediate parent class.

Java Program[B.java]

```
class A  
{  
    void fun()  
    {  
        System.out.println("Method: Class A");  
    }  
}  
class B extends A  
{  
    void fun()  
    {  
        System.out.println("Method: Class B");  
    }  
    void display()  
    {  
        super.fun();  
    }  
    public static void main(String args[])  
    {  
        B obj=new B();  
        obj.display();  
    }  
}
```

Output

Method: Class A

Program Explanation : In above program, the derived class can access the immediate parent's class method using **super.fun()**. Hence is the output. You can change **super.fun()** to **fun()**. Then note that in this case, the output will be invocation of subclass method **fun**.

3. The **super()** is used to invoke the immediate parent class constructor.

Java Program[B.java]

```
class A
{
    A()
    {
        System.out.println("Constructor of Class A");
    }
}
class B extends A
{
    B()
    {
        super();
        System.out.println("Constructor of Class B");
    }
    public static void main(String args[])
    {
        B obj=new B();
    }
}
```

Output

```
Constructor of Class A
Constructor of Class B
```

Program Explanation : In above program, the constructor in class B makes a call to the constructor of immediate parent class by using the keyword **super**, hence the print statement in parent class constructor is executed and then the print statement for class B constructor is executed.

Ex. 2.8.1 : What is inheritance ? How will you call parameterized constructor and overriden method from parent class in sub class ?

AU : Dec.-14, Marks 8

Sol. : Inheritance: - Inheritance is a mechanism in Java by which derived class can borrow the properties of base class and at the same time the derived class may have some additional properties.

Calling Parameterized constructor from parent class in subclass

```
class Parentclass
```

```
{
    //no-arg constructor
    Parentclass()
    {
        System.out.println("no-arg constructor of parent class");
    }
    //arg or parameterized constructor
    Parentclass(String str)
    {
        System.out.println(str+" This is parameterized constructor of parent class");
    }
    void display(String str)
    {
        System.out.println(str+" This is overridden method");
    }
}
```

```
class Subclass extends Parentclass
```

```
{
    Subclass()
    {
        super("Welcome");//calling super class's parameterized constructor
        System.out.println("Constructor of child class");
    }
    void display();//overridden method
    {
        super.display("Hi");//calling super class overridden method
    }
    public static void main(String args[])
    {
        Subclass obj= new Subclass();
        obj.display();
    }
}
```

Output

```
Welcome This is parameterized constructor of parent class
Constructor of child class
Hi This is overridden method
```

Review Question

1. With suitable program segments describe the usage of super keyword.

2.9 Object Class

In Java there is a special class named **Object**. If no inheritance is specified for the classes then all those classes are subclass of the **Object** class. In other words, **Object** is a superclass of all other classes by default. Hence

public class A { ... } is equal to **public class A extends Object { ... }**

There are two commonly used methods in **Object** class. Those are **toString()** and **equals()**.

Let us discuss these two methods one by one -

2.9.1 toString Method of Object Class

The **toString** method returns the String type value. Hence the signature (syntax) of this method is -

public String toString()

Illustration 1 :

```
class A extends Object
{
}
class B extends A
{
}
class ObjectClassDemo
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println("obj: "+obj);
        System.out.println("obj.toString(): "+obj.toString());
    }
}
```

Output

```
F:\test>javac ObjectClassDemo.java

F:\test>java ObjectClassDemo
obj: A@3e25a5
obj.toString(): A@3e25a5
```

Here A denotes
the class of the
object

This part denotes the
hexadecimal
address.

Illustration 2 :

```
class A extends Object
{
    public String toString() // Method is overridden
    {
        String str="Hello Friends!!!";
        return str;
    }
}
class B extends A
{
}
class ObjectClassDemo
{
    public static void main(String args[])
    {
        A obj=new A();
        System.out.println("obj: "+obj);
        System.out.println("obj.toString(): "+obj.toString());
    }
}
```

Output

```
F:\test>javac ObjectClassDemo.java

F:\test>java ObjectClassDemo
obj: Hello Friends!!!
obj.toString(): Hello Friends!!!
```

Fig. 2.9.1 Invoking toString() method

If we invoke the `toString` method, by default then it returns a string which describes the object. This returned string consists of the character "@" and object's memory address in hexadecimal form. But we can override the `toString` method.

Fig. 2.9.1 shows the two illustrations that support the above discussion -

The above method `toString` is overridden in second illustration, in order to print the desired string.

2.9.2 equals Method of Object Class

The method `equals` is useful for comparing values specified by two objects. Following is an example of Java program in which we have created two classes namely A and B. In order to compare the two values **a** and **b** of these two classes the method **`equals`** is overridden.

Java Program[ObjectClassDemo1.java]

```
class A extends Object
{
    int a=10;
    public boolean equals(Object obj)
    {
        if(obj instanceof B) { return a == ((B)obj).b; }
        else return false;
    }
}
class B extends A
{
    int b=10;
}
class ObjectClassDemo1
{
    public static void main(String args[])
    {
        A obj1=new A();
        B obj2=new B();
        System.out.println("The two values of a and b are equal:
                           "+obj1.equals(obj2));
    }
}
```

Output

```
F:\test>javac ObjectClassDemo1.java
```

```
F:\test>java ObjectClassDemo1
```

```
The two values of a and b are equal: true
```

Program Explanation

As the values of **a** and **b** in above program are same we get the Boolean result as **true** but just change the value of **a** in above program and notice the change in the above output.

2.10 Abstract Classes and Methods

AU : CSE : Dec.-10, 12, 13,19, May-13, IT : May-13, Marks 8

- In inheritance hierarchy, the superclass is very general and less specific. This class does nothing but only specifies the member functions that can be used in hierarchy. Such a class is called **abstract class**.

For example - In the following Java program we have created three classes - class A is a superclass containing two methods, the class B and class C are inherited from class A. The class A is an abstract class because it contains one abstract method **fun1()**. We have defined this method as abstract because, its definition is overridden in the subclasses **B** and **C**, another function of class A that is **fun2()** is a normal function. Refer Fig. 2.10.1.

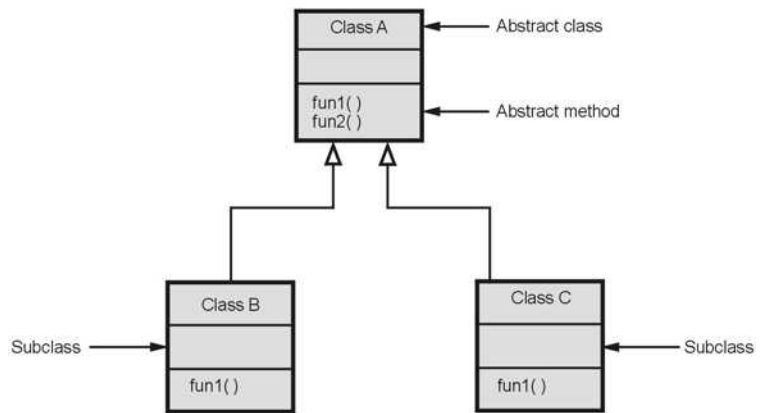


Fig. 2.10.1 Abstract class

Java Program [AbstractClsDemo.java]

abstract class A

```

{
  abstract void fun1();
  void fun2()
  {
    System.out.println("A:Infun2");
  }
}

```

This method is so abstract that it has not definition body.

This is an **abstract method**.

```

}
}
class B extends A
{
  void fun1()
  {
    System.out.println("B:In fun1");
  }
}
}

```

```
class C extends A
{
    void fun1()
    {
        System.out.println("C:In fun1");
    }
}

public class AbstractClsDemo
{
    public static void main(String[] args)
    {
        B b=new B();
        C c=new C();
        b.fun1(); //invoking the overridden method of class B
        b.fun2();
        c.fun1();//invoking the overridden method of class C
        c.fun2();
    }
}
```

Output

```
F:\test>javac AbstractClsDemo.java
```

```
F:\test>java AbstractClsDemo
```

```
B:In fun1
```

```
A:In fun2
```

```
C:In fun1
```

```
A:In fun2
```

Program Explanation

- In above program, the class **A** is a superclass. It is an abstract class as well. The name of this class is preceded by the keyword **abstract**. This class is abstract because it contains an abstract method **fun1**. The method is called abstract because it does not have any definition body. Note that the abstract method should be declared with the keyword **abstract**.
- There are two classes **B** and **C** which are subclasses of superclass **A**. The function definition **fun1** is overridden in these classes.
- In the **main function** we can access the methods of the subclasses by instantiating their objects. That is why we have created **b as an object of class B** and **c as an object of class C**. Using these objects appropriate **fun1** can be invoked. Note that the **fun2** will always be from class **A** even if we call it using the object of class B or C.

- If we write a statement **A a=new A()** i.e. if we instantiate the abstract class then it will generate compiler error. That means the *abstract classes can not be instantiated*.

Points to Remember about abstract classes and abstract methods

1. An **abstract** method must be present in an **abstract class** only. It should not be present in a non-abstract class.
2. In all the non-abstract subclasses extended from an abstract superclass all the abstract methods must be implemented. An un-implemented abstract method in the subclass is not allowed.
3. Abstract class cannot be instantiated using the new operator.
4. A constructor of an abstract class can be defined and can be invoked by the subclasses.
5. A class that contains abstract method must be abstract but the abstract class may not contain an abstract method. This class is simply used as a base class for defining new subclasses.
6. A subclass can be abstract but the superclass can be concrete. For example the superclass Object is concrete but the subclass class A of it can be abstract.
7. The abstract class cannot be instantiated using new operator but an abstract class can be used as a data type.

2.10.1 Difference between Abstract Class and Concrete Class

Abstract class	Concrete class
The abstract keyword is used to define the abstract class.	The keyword class is used to define the concrete class.
The abstract class have partial or no implementation at all.	The concrete class contains the data members and member functions defined within it.
Abstract class can not be instantiated.	Concrete class are usually instantiated in order to access the belonging data members and member functions.
Abstract classes may contain abstract methods.	Concrete classes contain concrete method i.e. with code and functionality. Concrete class can not contain abstract method.
Abstract classes need to be extended in order to make them useful.	Concrete classes may or may not be extended.
Abstract classes always act as a parent class.	Concrete class can be parent or child or neither of the two.

Ex. 2.10.1 : Create an abstract base class shape with two members base and height, a member function for initialization and a pure virtual function to compute area(). Derive two specific classes triangle and rectangle which override the function area(). Use these classes in a main function and display the area of a triangle and a rectangle.

Sol. :

Test.java

```
abstract class shape
{
    int base,height;
    double a;
    void initFun()
    {
        base=5;height=6;
    }
    abstract void compute_area(); //pure virtual function
}

class triangle extends shape
{
    public void compute_area()
    {

        a=(base*height)/2;
        System.out.println("\n Area of triangle is "+a);

    }
}

class Rectangle extends shape
{
    public void compute_area()
    {

        a=(base*height);
        System.out.println("\n Area of rectangle is "+a);

    }
}

public class Test
{
    public static void main(String[] args)
    {

        triangle obj1=new triangle();
```

```
obj1.initFun();
obj1.compute_area();
Rectangle obj2=new Rectangle();
obj2.initFun();
obj2.compute_area();

}
}
```

Output

```
D:\test>javac Test.java
D:\test>java Test
Area of triangle is 15.0
Area of rectangle is 30.0
```

Ex. 2.10.2 : What are the conditions to be satisfied while declaring abstract classes.

Sol. : Following are the conditions that must be satisfied while declaring the abstract classes -

1. The class name must be preceded by the keyword **abstract class**.
2. The abstract class must have one abstract method. This abstract method must also be preceded by the keyword **abstract** and it should not have definition body.

Ex. 2.10.3 : Can an abstract class in Java be instantiated ? Give the reason.

Sol. : The abstract class can not be instantiated (i.e. we can not create the object of this class using new operator) because the abstract class is very much general and less specific. It does nothing and simply lists out only common features of other classes.

Ex. 2.10.4 : Write a program to define abstract class, with two methods addition() and subtraction(), addition() is abstract method. Implement the abstract method and call that method using a program(s).

Sol. :

Java Program[AbstractDemo.java]

```
abstract class A
{
int a=10;
int b=20;
int c;
abstract void addition(int x,int y);
void subtraction()
{
c=b-a;
System.out.println("Subtraction of 20 and 10: "+c);
}
```

```
}  
class B extends A  
{  
void addition(int x,int y)  
{  
int z=x+y;  
System.out.println("addtion function in class B: "+z);  
}  
}  
class C extends A  
{  
void addition(int x,int y)  
{  
int z=x+y;  
System.out.println("addtion function in class C: "+z);  
}  
}  
public class AbstractDemo  
{  
public static void main(String[] args)  
{  
B b=new B();  
C c=new C();  
System.out.println("-----");  
b.addition(10,20);  
b.subtraction();  
System.out.println("-----");  
c.addition(100,200);  
c.subtraction();  
System.out.println("-----");  
}  
}
```

Output

```
-----  
addtion function in class B: 30  
Subtraction of 20 and 10: 10  
-----  
addtion function in class C: 300  
Subtraction of 20 and 10: 10
```

Review Questions

1. Write briefly on abstract classes with an example.

AU : CSE : Dec.-10, Marks 6

2. Write in detail about : Abstract classes.

AU : CSE : Dec.-12, Marks 8

3. What does it mean that a method or class is abstract ? Can we make an instance of an abstract class ? Explain it with example.

AU : IT : May-13, Marks 8

4. What is abstract class ? Write a program for abstract example.

AU : CSE: May-13, Marks 8

5. Explain the concept of abstract class with example.

AU : CSE : Dec.-13, Marks 8

6. What is an abstract class ? Illustrate with an example to demonstrate abstract class.

AU : Dec.-19, Marks 8

2.11 Final Methods and Classes

AU : Dec.-13, Marks 8

The final keyword can be applied at three places -

- For declaring variables
- For declaring the methods
- For declaring the class

2.11.1 Final Variables and Methods

A variable can be declared as final. If a particular variable is declared as final then it cannot be modified further. The final variable is always a constant. For example -

```
final int a=10;
```

The final keyword can also be applied to the method. When final keyword is applied to the method, the method overriding is avoided. That means the methods those are declared with the keyword **final** cannot be overridden.

Consider the following Java program which makes use of the keyword **final** for declaring the method -

```
class Test
{
    final void fun()
    {
        System.out.println("\n Hello, this function declared using final");
    }
}
class Test1 extends Test
{
    final void fun()
    {
        System.out.println("\n Hello, this another function");
    }
}
```

Output

```
D:\>javac Test.java
Test.java:10: fun() in Test1 cannot override fun() in Test; overridden method is final
    final void fun()
           ^
1 error
```

Program Explanation

The above program, on execution shows the error. Because the method **fun** is declared with the keyword **final** and it cannot be overridden in derived class.

2.11.2 Final Classes to Stop Inheritance

If we declare particular class as final, no class can be derived from it. Following Java program is an example of final classes.

```
final class Test
{
    void fun()
    {
        System.out.println("\n Hello, this function in base class");
    }
}
class Test1 extends Test
{
    final void fun()
    {
        System.out.println("\n Hello, this another function");
    }
}
```

Output

```
D:\>javac Test.java
Test.java:8: cannot inherit from final Test
class Test1 extends Test
      ^
1 error
```

Review Question

1. Write a note on final keyword.

AU : Dec.-13, Marks 8**2.12 Finalize() Method****AU : Dec.-18, Marks 7**

- Java has a facility of automatic garbage collection. Hence even though we allocate the memory and then forget to deallocate it then the objects that are no longer is used get freed.

- Inside the **finalize() method** you will specify those actions that must be performed before an object is destroyed.
- The garbage collector runs periodically checking for objects that are no longer referenced by any running state or indirectly through other referenced objects.
- To add finalizer to a class simply define the **finalize** method. The syntax to finalize() the code is -

void finalize()

```
{  
    finalization code  
}
```

Note that finalize() method is called just before the garbage collection. It is not called when an object goes out-of-scope.

- The finalize method of an object is called when garbage collector is about to clean up the object.
- The purpose of finalization is to perform some action before the objects get cleaned up.
- The clean up code can be kept in the finalize method block.

Review Question

1. State the purpose of finalize () method in java. With an example explain how finalize () method can be used in java program.

AU : Dec.-18, Marks 7

Two Marks Questions with Answers

Q.1 Define the term inheritance.

Ans. : Inheritance is a mechanism in which the derived class borrows the properties of the base class.

Q.2 Explain the use of extend keyword with suitable example.

Ans. :

- The extend keyword is used in inheritance.
- When the child class is derived from its parent class then the keyword extends is used.

Q.3 What is the difference between superclass and subclass ?

Ans. :

- The superclass is a parent class or base class from which another class can be derived.
- The subclass is always a derived class which inherits the properties of the base class or superclass.
- The superclass is normally a generalized class whereas the subclass is normally for specific purpose.

Q.4 Enlist various forms of inheritance.

Ans. : Various forms of inheritance are –

1. Single level inheritance 2. Multi-level inheritance
3. Multiple inheritance 4. Hierarchical inheritance 5. Hybrid inheritance

Q.5 What is the use of super keyword ?

Ans. : The super class is used to access immediate parent class from the subclass. It is used to

1. access parent's variable, 2. access parent's method
3. access parent's constructor invocation

Q.6 Differentiate between inheritance and polymorphism.

Ans. :

Sr. No.	Inheritance	Polymorphism
1.	Inheritance is a property in which some of the properties and methods of base class can be derived by the derived class.	Polymorphism is ability for an object to used different forms. The name of the function remains the same but it can perform different tasks.
2.	Various types of inheritance can be single inheritance, multiple inheritance, multilevel inheritance and hybrid inheritance.	Various types of polymorphism are compile time polymorphism and run time polymorphism. In compile time polymorphism there are two types of overloading possible. - Functional overloading and operator overloading. In run time polymorphism there is a use of virtual function.

Q.7 Distinguish between static and dynamic binding.

AU : IT : May-11

Ans. : Static binding is a type of binding in which the function call is resolved at compile time.

The dynamic binding is a type of binding in which the function call is resolved at run time.

The static binding is called the early binding and the dynamic binding is called late binding.

Q.8 What is the purpose of 'final' keyword

AU : IT : Dec.-11, CSE : Dec.-12

Ans. : The keyword 'final' is used to avoid further modification of a variable, method or a class. For instance if a variable is declared as final then it can not be modified further, similarly, if a method is declared as final then the method overriding is avoided.

Q.9 Define inheritance hierarchy.

AU : CSE : Dec.-11, 18

Ans. : The inheritance hierarchy represents the collection of classes the inherit from their base classes and thereby make use of the code present in the base class. The data reusability can be achieved by inheritance hierarchy. For example

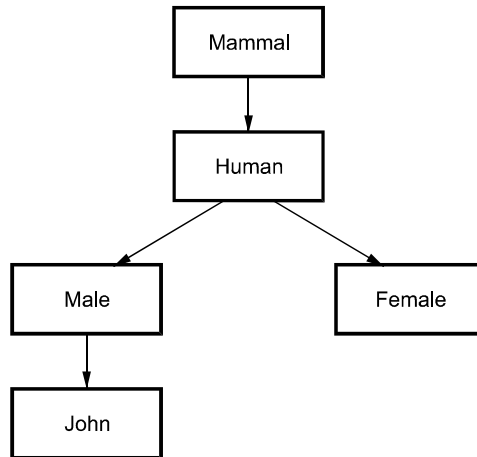


Fig. 2.1

Q.10 How to prevent inheritance ?

Ans. : If we declare particular class as **final** then no class can be derived from it. For example

```
final class Test
{
    ...
    ...
    ...
}
class Test1 extends Test
{
    ...
    ...
    ...
}
```

The above code will produce an error "cannot inherit from final Test".

Thus the use of keyword final for the class prevents the inheritance.

**Q.11 Write a class declarations for the following relationship, assuming that all classes are public:
a Bulldog is a kind of dog, and a Dog is a kind of Animal.**

Ans. :

```
class Animal
{
    ...
}
```

```

class dog extends Animal
{
    ...
}
class Bulldog extends dog
{
    ...
}

```

Q.12 What is meant by dynamic method dispatch ?

Ans. : The dynamic method dispatch is run time polymorphism in which a call to overridden function is resolved at runtime instead of at compile time. Hence is the name. For example

```

class Test1
{
    public void method1()
    { }
}
class Test2 extends Test1
{
    public void method1()
    { }
}
void main()
{
    Test1 obj=new Test2();
    obj.method1();
}

```

The **method1** of class **Test2** will be invoked here, thus the code supports the dynamic method dispatch.

Q.13 Can an abstract class be final ? Why ?

AU : IT : Dec.-10

Ans. : The abstract class cannot be final because once we declared the abstract base class with the keyword final then it can not be inherited.

Q.14 Can an abstract class in Java be instantiated ? Give the reason.

AU : IT : Dec.-13

Ans. : The abstract class can not be instantiated(i.e. we can not create the object of this class using new operator) because the abstract class is very much general and less specific. It does nothing and simply lists out only common features of other classes.

Q.15 Why is multiple inheritance using classes a disadvantage in Java ?

AU : IT : May-13

Ans. : In multiple inheritance, the child class is derived from two or more parent classes. It can lead to ambiguity when two base classes implement a method with the same name.

Q.16 State the condition for method overriding in Java

AU : May 19

Ans. : Method overriding occurs only when the name of the two methods(method in super class and method in subclass) are same and their type signature is same.



Notes

UNIT-II

3

Interfaces

Syllabus

Interfaces - defining an interface, implementing interface, differences between classes and interfaces and extending interfaces - Object cloning - inner classes, Array Lists, Strings.

Contents

3.1	Defining an Interface.....	May-19
3.2	Implementing Interface.....	Dec.-14,..... Marks 8
3.3	Applying Interfaces.....	May-13,19..... Marks 16
3.4	Variables in Interface	
3.5	Extending Interface	
3.6	Multiple Inheritance	
3.7	Object Cloning.....	IT : May-11, Dec.-10, 11, CSE : Dec.-10, 11,..... Marks 16
3.8	Inner Classes	CSE : Dec.-10, 11, 13, May-13,..... Marks 8
3.9	ArrayList	
3.10	Strings.....	IT : Dec.-13, Marks 8

3.1 Defining an Interface

AU : May-19, Marks 6

- The interface can be defined using following syntax

```
access_modifier interface name_of_interface
{
    return_type method_name1(parameter1,parameter2,...parametern);
    ...
    return_type method_name1(parameter1,parameter2,...parametern);
    type static final variable_name=value;
    ...
}
```

- The access_modifier specifies the whether the interface is public or not. If the access specifier is not specified for an interface then that interface will be accessible to all the classes present in that package only. But if the interface is declared as public then it will be accessible to any of the class.
- The methods declared within the interface have no body. It is expected that these methods must be defined within the class definition.

3.1.1 Difference between Class and Interface

An interface is similar to a class but there lies some difference between the two.

Class	Interface
The class is denoted by a keyword class	The interface is denoted by a keyword interface
The class contains data members and methods. But the methods are defined in class implementation. Thus class contains an executable code.	The interfaces may contain data members and methods but the methods are not defined. The interface serves as an outline for the class.
By creating an instance of a class the class members can be accessed.	You can not create an instance of an interface.
The class can use various access specifiers like public, private or protected.	The interface makes use of only public access specifier.
The members of a class can be constant or final.	The members of interfaces are always declared as final.

3.1.2 Difference between Abstract Class and Interface

Following table presents the difference between the abstract class and interface.

Sr. No.	Abstract class	Interface
1.	The class can inherit only one abstract class	The class can implement more than one interfaces.
2.	Members of abstract class can have any access modifier such as public , private and protected .	Members of interface are public by default.
3.	The methods in abstract class may or may not have implementation.	The methods in interface have no implementation at all. Only declaration of the methods is given.
4.	Java abstract classes are comparatively efficient.	The interfaces are comparatively slow and implies extra level of indirection.
5.	Java abstract class is extended using the keyword extends .	Java interface can be implemented by using the keyword implements
6.	The member variables of abstract class can be non final	The member variables of interface are by default final

Review Questions

1. Define an interface.
2. What is interface ? What are the possible contents of an interface ? Explain.
3. Write the difference between classes and interfaces ?
4. Define abstract class and Interface and what is the difference between them explain with suitable examples.
5. Present a detailed comparison between classes and interfaces.

AU : May-19 , Marks 6

3.2 Implementing Interface

AU : Dec.-14, Marks 8

- It is necessary to create a class for every interface.
- The class must be defined in the following form while using the interface

```

class Class_name extends superclass_name
implements interface_name1,interface_name2,...
{
    //body of class
}

```

- Let us learn how to use interface for a class

Step 1 : Write following code and save it as my_interface.java

```

public interface my_interface
{
    void my_method(int val);
}

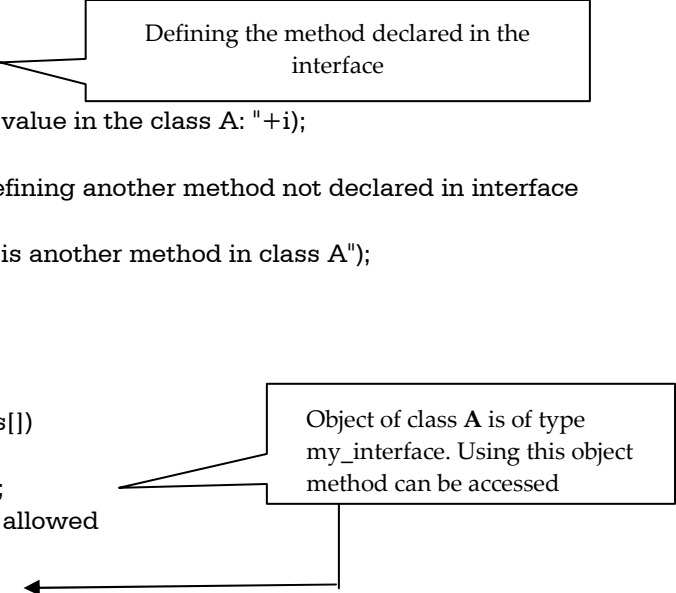
```

Do not compile this program. Simply save it.

Step 2 : Write following code in another file and save it using InterfaceDemo.java

Java Program[InterfaceDemo.java]

```
class A implements my_interface
{
    public void my_method(int i)
    {
        System.out.println("\n The value in the class A: "+i);
    }
    public void another_method() //Defining another method not declared in interface
    {
        System.out.println("\nThis is another method in class A");
    }
}
class InterfaceDemo
{
    public static void main(String args[])
    {
        my_interface obj=new A();
        //or A obj=new A() is also allowed
        A obj1=new A();
        obj.my_method(100);
        obj1.another_method();
    }
}
```



Step 3 : Compile the program created in step 2 and get the following output

```
F:\test>javac InterfaceDemo.java
```

```
F:\test>java InterfaceDemo
```

The value in the class A: 100

This is another method in class A

Program Explanation :

In above program, the interface **my_interface** declares only one method i.e. **my_method**. This method can be defined by class **A**. There is another method which is defined in class **A** and that is **another_method**. Note that this method is not declared in the interface **my_interface**. That means, a class can define any additional method which is not declared in the interface.

Various ways of interface implementation

- The design various ways by which the interface can be implemented by the class is represented by following Fig. 3.2.1.

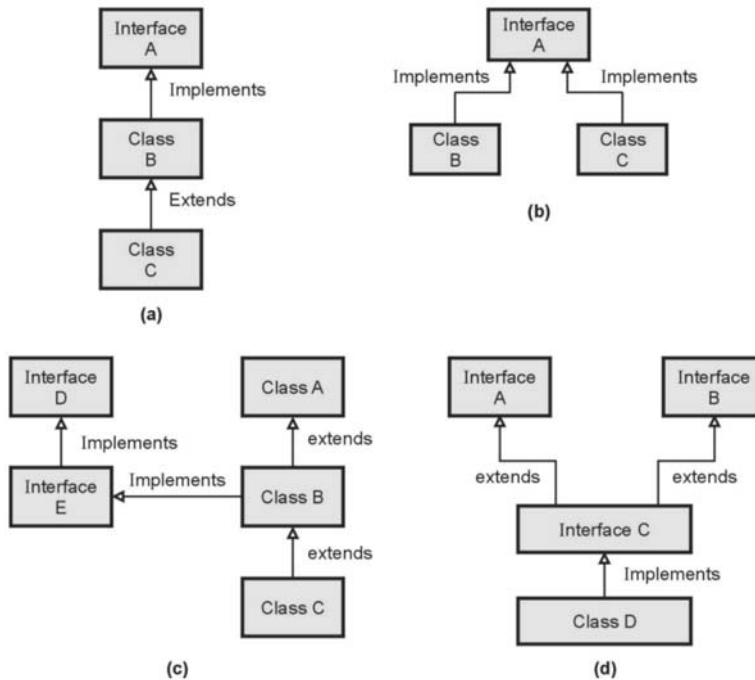


Fig. 3.2.1 Various ways of Interface implementation

- The variables can be assigned with some values within the interface. They are implicitly **final** and **static**. Even if you do not specify the variables as **final** and **static** they are assumed to be **final** and **static**.
- Interface variables are **static** because Java interfaces cannot be instantiated. Hence the value of the variable must be assigned in a static context in which no instance exists. The **final** modifier indicates that the value assigned to the interface variable is a true constant that cannot be re-assigned by program code.

Review Question

1. What is an interface ? Write a Java program to illustrate the use of interface. Give self explanatory comments in your program.

AU : Dec.-14, Marks 8

3.3 Applying Interfaces

AU : May-13, 19, Marks 16

The interface is a powerful tool. The same interface can be used by different classes for some method. Then this method can be implemented by each class in its own way. Thus same interface can provide variety of implementation. The selection of different implementations is done at the run time. Following is a simple Java program which illustrates this idea.

Step 1 : Write a simple interface as follows

Java Program [interface1.java]

```
interface interface1
{
    void MyMsg(String s);
}
```

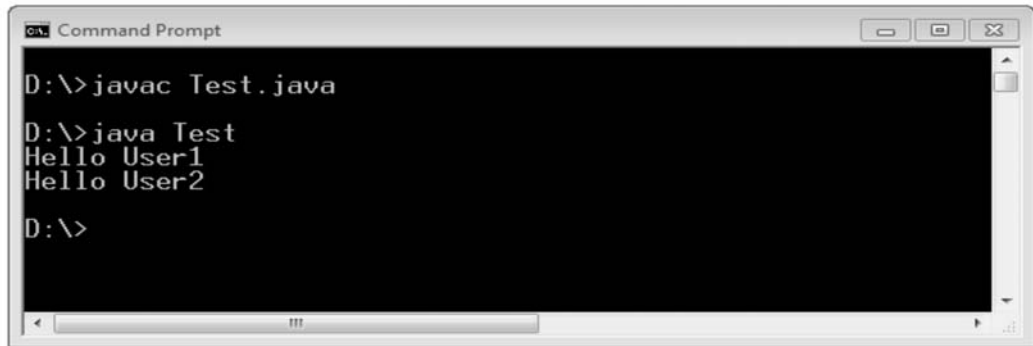
Step 2 : Write a simple Java Program having two classes having their own implementation of **MyMsg** method.

Java Program[Test.java]

```
import java.io.*;
import java.util.*;
class Class1 implements interface1
{
    private String s;
    public void MyMsg(String s)
    {
        System.out.println("Hello "+s);
    }
}
class Class2 implements interface1
{
    private String s;
    public void MyMsg(String s)
    {
        System.out.println("Hello "+s);
    }
}

class Test
{
    public static void main(String[] args)
    {
        interface1 inter;
        Class1 obj1=new Class1();
        Class2 obj2=new Class2();
        inter=obj1;
        inter.MyMsg("User1");
        inter=obj2;
        inter.MyMsg("User2");
    }
}
```

Step 3 : Execute the program.



```

D:\>javac Test.java

D:\>java Test
Hello User1
Hello User2

D:\>

```

Program Explanation :

- The selection of method **MyMsg** is done at the runtime and it depends upon the object which is assigned to the reference of the interface.
- We have created reference **inter** for the interface in which the method **MyMsg** is declared.
- Then the objects **obj1** and **obj2** are created for the classes **class1** and **class2** respectively. When the **obj1** is assigned to the reference **inter** then the message "Hello User1" will be displayed because the string passed to the method **MyMsg** is "User1". Similarly, when the **obj2** is assigned to the reference **inter** then the message "Hello User2" will be displayed because the string passed to the method **MyMsg** is "User2".
- Thus using the same interface different implementations can be selected.

Ex. 3.3.1 : Write a program to create interface method named **customer**. In this keep the methods called **information()**, **show()** and also maintain the tax rate. Implement this interface in **employee** class and calculate the tax of the employee based on their income.

Income	Tax percentage	
	Male	Female
>=1,90,000	Nil	Nil
>=2,00,000	10 %	Nil
>=5,00,000	20 %	10 %
<5,00,000	25 %	20 %

AU : May-13, Marks 16

Sol. :

Step 1 : Create an interface file named **Customer.java**. It is as follows -
interface Customer

```
{
    double rate1=0.10;
    double rate2=0.20;
    double rate3=0.25;
    double income=0.0;
    void information();
    void show();
}
```

Step 2 : Create a Java program named **Employee.java**. It is as follows -
Employee.java

```
import java.lang.System;
import java.util.Scanner;
class TestClass implements Customer
{
    private char sex;
    private double tax=0;
    private double netpay;
    private double income=0.0;
    public void information()
    {
        Scanner in=new Scanner(System.in);
        System.out.print("\n Enter Sex(M or F)");
        sex = in.next().charAt(0);
        System.out.println("\n Enter your income");
        income=in.nextDouble();
        if(sex=='M')
        {
            if((income >= 190000)&&(income < 200000))
            {
                netpay=income;
            }
            else if((income >= 200000)&&(income < 500000))
            {
                netpay=income-(income*rate1);
            }
            else if((income >= 500000)&&(income < 1000000))
            {
                netpay=income-(income*rate2);
            }
            else if(income > 1000000)
            {
                netpay=income-(income*rate3);
            }
        }
    }
}
```

```
        netpay=income-(income*rate3);
    }
    else
    {
        if((income >= 190000)&&(income < 200000))
        {
            netpay=income;
        }
        else if((income >= 200000)&&(income < 500000))
        {
            netpay=income;
        }
        else if((income >= 500000)&&(income < 1000000))
        {
            netpay=income-(income*rate1);
        }
        else if(income >= 1000000)
            netpay=income-(income*rate2);
    }
}

}
public void show()
{
    System.out.print("\n The net Income of the person is\n"+netpay);
}
}
class Employee
{
    public static void main(String[] args)
    {
        Customer cust;
        TestClass obj=new TestClass();
        cust=obj;
        cust.information();

        cust.show();
    }
}
```

Step 3 : Open the command prompt and issue the following commands to execute the above program

D:\>javac Employee.java

D:\>java Employee

Enter Sex(M or F)F

Enter your income
500000

The net Income of the person is
450000.0

Ex. 3.3.2 : Define an interface using JAVA that contains a method to calculate the perimeter of object. Define two classes - circle and rectangle with suitable fields and methods. Implement the interface “perimeter” in these classes. Write the appropriate main() method to create object of each class and test all methods.

Sol. :

Step 1 : We will write the simple interface. The name of this interface is **perimeter**. In this interface the method **calculate** is declared.

perimeter.java

```
interface perimeter
{
    void calculate();
}
```

Step 2 : Following is a java program in which two classes are defined- **circle** and **rectangle**.

The perimeter for each of this class is calculated using the function declared in the interface.

Main.java

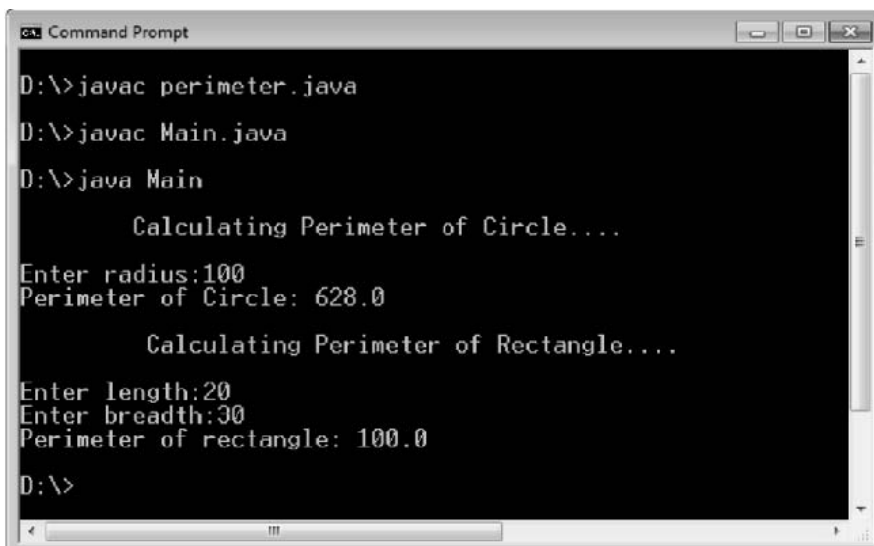
```
import java.io.*;
import java.util.*;
import java.util.Scanner;
class circle implements perimeter
{
    private double r;
    public void calculate()
    {
        System.out.print("Enter radius:");
        Scanner input=new Scanner(System.in);
        double r=input.nextDouble();
        double p=2*3.14*r;
        System.out.println("Perimeter of Circle: "+p);
    }
}
class rectangle implements perimeter
{
    private double length,breadth;
    public void calculate()
```

```
{
    System.out.print("Enter length:");
    Scanner input=new Scanner(System.in);
    double length=input.nextDouble();
    System.out.print("Enter breadth:");
    double breadth=input.nextDouble();
    double p=2*(length+breadth);
    System.out.println("Perimeter of rectangle: "+p);
}

}
class Main
{
    public static void main(String[] args)
    {
        perimeter obj;
        circle obj1=new circle();
        rectangle obj2=new rectangle();
        System.out.println("\n\tCalculating Perimeter of Circle....\n");
        obj=obj1;
        obj.calculate();

        System.out.println("\n\t Calculating Perimeter of Rectangle....\n");
        obj=obj2;
        obj.calculate();
    }
}
```

Step 3 : The output for this program will be as follows -



```
Command Prompt

D:\>javac perimeter.java
D:\>javac Main.java
D:\>java Main

        Calculating Perimeter of Circle....

Enter radius:100
Perimeter of Circle: 628.0

        Calculating Perimeter of Rectangle....

Enter length:20
Enter breadth:30
Perimeter of rectangle: 100.0

D:\>
```

Ex. 3.3.3 : Write an interface called shape with methods draw() and getArea(). Further design two classes called Circle and Rectangle that implements Shape to compute area of respective shapes. Use appropriate getter and setter methods. Write a Java program for the same.

Sol. :

```
interface Shape
{
    void draw();
    double getArea();
}
class Circle implements Shape
{
    private double radius;
    public void setRadius(double r)
    {
        radius=r;
    }
    public double getRadius()
    {
        return radius;
    }
    public double getArea()
    {
        return (3.14*radius*radius);
    }
    public void draw()
    {
        System.out.println("Area of Circle is :"+getArea());
    }
}
class Rectangle implements Shape
{
    private double length,breadth;
    public void setLength(double l)
    {
        length=l;
    }
    public double getLength()
    {
        return length;
    }
    public void setBreadth(double b)
    {
        breadth=b;
    }
}
```

```
        public double getBreadth()
        {
            return breadth;
        }
        public double getArea()
        {
            return (length*breadth);
        }
        public void draw()
        {
            System.out.println("Area of Rectangle is :"+getArea());
        }
    }

class Test
{
    public static void main(String[] args)
    {
        Circle c=new Circle();
        c.setRadius(10);
        System.out.println("The Radius of Circle is "+c.getRadius());
        c.draw();
        Rectangle r=new Rectangle();
        r.setLength(10);
        r.setBreadth(20);
        System.out.println("Length = "+r.getLength()+" , Breadth
                           = "+r.getBreadth());

        r.draw();
    }
}
```

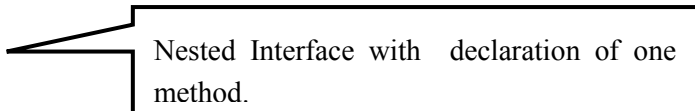
Output

```
The Radius of Circle is 10.0
Area of Circle is :314.0
Length = 10.0, Breadth = 20.0
Area of Rectangle is :200.0
```

3.3.1 Nested Interface

Nested interface is an interface which is declared within another interface. For example -

```
interface MyInterface1 {
    interface MyInterface2 {
        void show();
    }
}
```

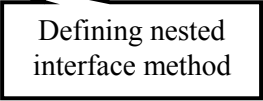


Nested Interface with declaration of one method.

```
class NestedInterfaceDemo implements MyInterface1.MyInterface2 {
    public void show() {

        System.out.println("Inside Nested interface method");
    }
    public static void main(String args[]) {

        MyInterface1.MyInterface2 obj= new NestedInterfaceDemo();
        obj.show();
    }
}
```



Defining nested interface method

Output

Inside Nested interface method

Ex. 3.3.4 : The transport interface declares a deliver() method. The abstract class Animal is the super class of Tiger, Camel, Deer and Donkey classes. The transport interface is implemented by the Camel and Donkey classes. Write a test program to initialize an array of four animal objects. If the object implements the transport interface, the deliver() method is invoked.

Sol. :

```
interface Transport
{
    public void deliver();
}
abstract class Animal
{
    String item;
    abstract void deliver();
}
class Tiger extends Animal
{
    public void deliver(){
    }
}
class Camel extends Animal implements Transport
{
    Camel(String s)
    {
        item=s;
    }
    public void deliver(){
        System.out.println("\tCamels are used in deserts to carry "+item+".");
    }
}
class Deer extends Animal
{
}
```

```
public void deliver(){
}
}
class Donkey extends Animal implements Transport
{
    Donkey(String s){
        item=s;
    }
    public void deliver(){
        System.out.println("\tDonkeys can carry "+item+".");
    }
}
class testclass
{
    public static void main(String args[]) throws NullPointerException
    {
        Transport[] t=new Transport[4];
        t[1]=new Camel("goods and people");
        t[2]=new Donkey("heavy load");
        System.out.println("Transport interface is implemented by following animals...");
        t[1].deliver();
        t[2].deliver();
    }
}
```

Output

Transport interface is implemented by following animal...
Camels are used in deserts to carry goods and people.
Donkeys can carry heavy load.

Review Question

1. Explain simple interfaces and nested interfaces with examples.

AU : May-19, Marks 7

3.4 Variables in Interface

- The variables can be assigned with some values within the interface. They are implicitly final and static. Even if you do not specify the variables as final and static they are assumed to be final and static.
- The members of interface are **static and final** because -
 - 1) The reason for being **static** - The members of interface belong to interface only and not object.
 - 2) The reason for being **final** - Any implementation can change value of fields if they are not defined as final. Then these members would become part of the implementation. An interface is pure specification without any implementation.

3.5 Extending Interface

- Interfaces can be extended similar to the classes. That means we can derive subclasses from the main class using the keyword **extend** , similarly we can derive the subinterfaces from main interfaces by using the keyword **extends**.
- The syntax is

```
interface Interface_name2 extends interface_name1
{
    ...
    ...
    ...
}
```

} Body of interface

- For example

```
interface A
{
    int val=10;
}
interface B extends A
{
    void print_val();
}
```

That means in **interface B** the **display** method can access the value of variable **val**.

Similarly more than one interfaces can be extended.

```
interface A
{
    int val=10;
}
interface B
{
    void print_val();
}
interface C extends A,B
{
    ...

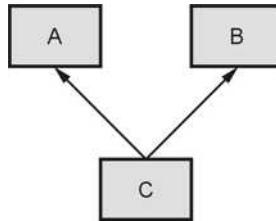
    ...
}
```

Even-though methods are declared inside the interfaces and sub-interfaces, these methods are not allowed to be defined in them. Note that methods are defined only in the classes and not in the interfaces.

3.6 Multiple Inheritance

Definition: Multiple inheritance is a mechanism in which the child class inherits the properties from more than one parent classes.

For example



- **Java does not support multiple inheritance** because it creates ambiguity when the properties from both the parent classes are inherited in child class or derived class. But it is supported in case of **interface** because there is no ambiguity as implementation is provided by the implementation class.

- **Implementation**

```
interface A
{
    void display1();
}
interface B
{
    void display2();
}
class C implements A,B
{
    public void display1()
    {
        System.out.println("From Interface A");
    }
    public void display2()
    {
        System.out.println("From Interface B");
    }
    public static void main(String args[])
    {
        C obj = new C();
        obj.display1();
    }
}
```

```
        obj.display2();
    }
}
```

Output

```
From Interface A
From Interface B
```

Program Explanation :

In above program, we have created both the parent interfaces and the derived class is using method declared on the parent interface. Note that the definition of these methods is present in the child class. Thus multiple inheritance is achieved with the help of interface.

3.7 Object Cloning

AU : IT : May-11, Dec.-10, 11, CSE : Dec.-10, 11, Marks 16

Object cloning is technique of **duplicating the object** in the Java program. If object copying is planned then the copy must be done by copying the each fields of the object. This is poor method of copying the contents from one class to another. But object cloning technique simplifies it. The object cloning is implemented using the **interface** called **cloneable**. This interface requires the method **clone**. The **clone()** method returns the object. Following Java code shows how objects are cloned. In the following example we have created a simple class named **student**. The object of this class has two fields Name and Course which define the name of the student and the course chosen by him. For getting and setting these field values the `getName`, `getCourse`, `setName` and `setCourse` methods are used.

Java Program

```
public class CloneDemo
{
    public static void main(String[] args)
    {
        Student s1=new Student();
        s1.setName("Chitra");
        s1.setCourse("Computer");
        Student s2=(Student)s1.clone();
        System.out.println("\t\tStudent 1");
        System.out.println("Name: "+s1.getName());
        System.out.println("Course: "+s1.getCourse());

        System.out.println("\t\tStudent 2");
        System.out.println("Name: "+s2.getName());
        System.out.println("Course: "+s2.getCourse());
    }
}
```

```
}  
class Student implements Cloneable  
{  
    private String Name;  
    private String Course;  
    public Object clone()  
    {  
        Student obj=new Student();  
        obj.setName(this.Name); //current name  
        obj.setCourse(this.Course); //current course  
        return obj; //is assigned to the obj  
    }  
    public String getName()  
    {  
        return Name;  
    }  
    public void setName(String Name)  
    {  
        this.Name=Name;  
    }  
  
    public String getCourse()  
    {  
        return Course;  
    }  
    public void setCourse(String Course)  
    {  
        this.Course=Course;  
    }  
}
```

Output

```
        Student 1  
Name: Chitra  
Course: Computer  
        Student 2  
Name: Chitra  
Course: Computer
```

Note that the contents of the first object gets copied to second object.

Ex. 3.7.1 : Consider a class person with attributes firstname and lastname. Write a java program to create and clone the instances of the Person class

AU : IT : May-11, Marks 16

Sol. :

```
public class CloneDemo  
{
```

```
public static void main(String[] args)
{
    Person p1=new Person();
    p1.setfirstname("Chitra");
    p1.setlastname("Roy");
    Person p2=(Person)p1.clone();
    System.out.println("\t\tPerson 1");
    System.out.println("First Name: "+p1.getfirstname());
    System.out.println("Last Name: "+p1.getlastname());

    System.out.println("\t\tPerson 2");
    System.out.println("First Name: "+p2.getfirstname());
    System.out.println("Last Name: "+p2.getlastname());
}
}
class Person implements Cloneable
{
    private String firstname;
    private String lastname;
    public Object clone()
    {
        Person obj=new Person();
        obj.setfirstname(this.firstname); //current firstname
        obj.setlastname(this.lastname); //current lastname
        return obj; //is assigned to the obj
    }
    public String getfirstname()
    {
        return firstname;
    }
    public void setfirstname(String firstname)
    {
        this.firstname=firstname;
    }
    public String getlastname()
    {
        return lastname;
    }
    public void setlastname(String lastname)
    {
        this.lastname=lastname;
    }
}
```

Output

Person 1
First Name: Chitra

Last Name: Roy
Person 2
First Name: Chitra
Last Name: Roy

Review Questions

1. What is object cloning ? Why it is needed ? Explain how objects are cloned ?

AU : MIT : Dec.-10, Marks 16

1. Explain the following with examples : 1)The clonable interface 2)The property interface.

AU : CSE : Dec.-10, Marks 8

3. What is object cloning ? Discuss with examples.

AU : IT : Dec.-11, CSE : Dec.-11, Marks 8

3.8 Inner Classes

AU : CSE : Dec.-10, 11, 13, May-13, Marks 8

Inner classes are the nested classes. That means these are the classes that are defined inside the other classes. The syntax of defining the inner class is -

```
Access_modifier class OuterClass
{
    //code
    Access_modifier class InnerClass
    {
        //code
    }
}
```

Following are some **properties** of inner **class** -

- The outer class can inherit as many number of inner class objects as it wants.
- If the outer class and the corresponding inner class both are **public** then any other class can create an instance of this inner class.
- The inner class objects do not get instantiated with an outer class object.
- The outer class can call the private methods of inner class.
- Inner class code has free access to all elements of the outer class object that contains it.
- If the inner class has a variable with same name then the outer class's variable can be accessed like this -

outerclassname.this.variable_name

There are **four types of inner classes** -

1. Static member classes

- This inner class is defined as the static member variable of another class.
- Static members of the outer class are visible to the **static inner** class.

- The non-static members of the outer class are not available to inner class.

Syntax

```
Access_modifier class OuterClass
{
    //code
    public static class InnerClass
    {
        //code
    }
}
```

2. Member classes

- This type of inner class is non-static member of outer class.

3. Local classes

- This class is defined within a Java code just like a local variable.
- Local classes are never declared with an access specifier.
- The scope inner classes is always restricted to the block in which they are declared.
- The local classes are completely hidden from the outside world.

Syntax

```
Access_modifier class OuterClass
{
    //code
    Access_modifier return_type methodname(arguments)
    {
        class InnerClass
        {
            //code
        }
    }
    //code
}
```

4. Anonymous classes

- Anonymous class is a local class without any name.
- Anonymous class is a one-shot class- created exactly where needed
- The anonymous class is **created in** following **situations** -
 - When the class has very short body.
 - Only one instance of the class is needed.

- Class is used immediately after defining it.
- The anonymous inner class can extend the class, it can implement the interface or it can be declared in method argument.

Example

```
class MyInnerClass implements Runnable
{
    public void run()
    {
        System.out.println("Hello");
    }
}
class DemoClass
{
    public static void main(String[] arg)
    {
        MyInnerClass my=new MyInnerClass();
        Thread th=new Thread(my);
        my.start();
    }
}
```

Review Questions

1. What is static inner class ? Explain with example.
2. Discuss in detail about inner class with its usefulness.
3. Define Inner classes. How to access object state using inner classes ? Give an example.
4. Discuss the object and inner classes with examples.

AU : CSE : Dec.-10, Marks 8**AU : CSE : Dec.-11, Marks 8****AU : CSE : May-13, Marks 8****AU : CSE : Dec.-13, Marks 8****3.9 ArrayList**

The **ArrayList** class implements the List interface. It is used to implement the dynamic array. The dynamic array means the size of array can be created as per requirement. Hence **ArrayList** is a variable length array of object references. Initially ArrayList is created with some initial size and then as per requirement we can extend the size of array. When the objects are removed then the size of array can be reduced. The syntax of using ArrayList is as given below

-

ArrayList() ← Creates an empty list

ArrayList(Collection collection) ← Creates a list in which the collection elements are added

ArrayList(int c) ← Creates a list with specified capacity *c*, the capacity represents the size of the underlying array

Let us see simple Java program which demonstrates the ArrayList

Java Program [ArrayListProg.Java]

```

/*****
Program uses the ArrayList collection class to
implement ArrayList data structure
*****/
import java.util.*;
class ArrayListProg
{
    public static void main(String[] args)
    {
        System.out.println("\n\t\t Program for Implementing Array List");
        ArrayList obj=new ArrayList();           //creation of ArrayList
        System.out.println("\n Inserting some elements in the array");
        obj.add(10);
        obj.add(20);
        obj.add(30);
        obj.add(40);
        obj.add(50);
        System.out.println("The array elements are... "+obj);
        System.out.println("\n Inserting some elements in the array in between");
        obj.add(4,45);                           //inserting after element 40
        System.out.println("The array elements are... "+obj);
        System.out.println("\n Removing some elements from the array");
        obj.remove(1);                            //array index is passed to remove method
        System.out.println("The array elements are... "+obj);
        System.out.println("The array size is... "+obj.size());
    }
}

```

Output

Program for Implementing Array List

Inserting some elements in the array
The array elements are... [10, 20, 30, 40, 50]

Inserting some elements in the array in between
The array elements are... [10, 20, 30, 40, 45, 50]

Removing some elements from the array
The array elements are... [10, 30, 40, 45, 50]
The array size is... 5

Program Explanation

In above program we have created an array list of integer numbers we can also pass some character elements to the array list. While removing the element from the list we should pass the index position of the element. Note that the elements are placed from 0th index in the ArrayList. The above array is a dynamic array. As we insert the elements it grows and as we remove the elements it gets shrunk.

Ex. 3.9.1 : Write a program which stores the list of strings in an ArrayList and then displays the contents of the list.

Sol. :

```
import java.util.*;
class ArrayListProg
{
    public static void main(String[] args)
    {
        System.out.println("\n\t\t Program for Implementing Array List for List of Strings");
        ArrayList obj=new ArrayList();           //creation of ArrayList
        System.out.println("\n Inserting some elements in the array");
        obj.add("AAA");
        obj.add("BBB");
        obj.add("CCC");
        obj.add("DDD");
        obj.add("EEE");
        System.out.println("The array elements are... "+obj);
        System.out.println("The array size is... "+obj.size());
    }
}
```

Output

```
Program for Implementing Array List for List of Strings
Inserting some elements in the array
The array elements are... [AAA, BBB, CCC, DDD, EEE]
The array size is... 5
```

3.10 Strings

AU : IT : Dec.-13, Marks 8

Definition : String is a collection of characters. In Java String defines the object.

The string is normally used to represent the collection of characters

Syntax of String Constructor

String()	It is used for initialization of string object. It represents empty character sequence.
String(byte[] byte)	Constructs a new String by decoding the specified array of bytes using the platform's default charset.
String(byte[] bytes, int offset, int length, String charsetName)	Constructs a new String by decoding the specified subarray of bytes using the specified charset.
String (String strObj)	It constructs a String object which is initialized to same character sequence as that of the string referenced by strObj.

Example Program

Let us look at a sample program in which the string object is used in order to handle some text.

Java Program [StringDemo.java]

```
/*This is a Java program which makes use of strings
*/
```

```
class StringDemo
{
    public static void main(String args[])
    {
        String s="Hello, How are You?";
        System.out.println(s);
    }
}
```

Output

Hello, How are You?

String Literal

In Java String can be represented as a sequence of characters enclosed within the double quotes.

For example

```
String s="Hello,How are you?"
```

We can initialize the string using the empty string literal. For example

```
String mystr=""
```

Similarly the escape sequence can be used to denote a particular character in the string literal. For example we can use back slash followed by double quote to print the double quote character.

"\" "

Alternatively, String can be denoted as the array of characters. For example :

```
char str[] = {'P','R','O','G','R','A','M'};
```

Now we can have a variable `s` which can then be initialized with the string "PROGRAM" as follows -

```
String s = new String(str);
```

Operations on String

Following are some commonly defined methods by a string class -

Method	Description
<code>s1.CharAt(position)</code>	Returns the character present at the index position.
<code>s1.compareTo(s2)</code>	If <code>s1<s2</code> then it returns positive, if <code>s1>s2</code> then it returns negative and if <code>s1=s2</code> then it returns zero.
<code>s1.concat(s2)</code>	It returns the concatenated string of <code>s1</code> and <code>s2</code> .
<code>s1.equals(s2)</code>	If <code>s1</code> and <code>s2</code> are both equal then it returns true.
<code>s1.equalsIgnoreCase(s2)</code>	By ignoring case, if <code>s1</code> and <code>s2</code> are equal then it returns true.
<code>s1.indexOf('c')</code>	It returns the first occurrence of character 'c' in the string <code>s1</code> .
<code>s1.indexOf('c',n)</code>	It returns the position of 'c' that occur at after <code>n</code> th position in string <code>s1</code> .
<code>s1.length()</code>	It gives the length of string <code>s1</code> .
<code>String.Valueof(var)</code>	Converts the value of the variable passed to it into String type.

3.10.1 Finding Length of String

We can find the length of a given string using the method `length()`. Following program shows the use of `length()` method for strings

Java Program [Str_lengthDemo.java]

```
class Str_lengthDemo
{
    public static void main(String[] args)
    {
        char str[]={ 'P','R','O','G','R','A','M' };
        String S=new String(str);
```

```
System.out.println("The string S is " +S);
System.out.println("The length of string is " +S.length());
System.out.print("The string in character array is ");
for(int i=0;i<S.length();i++)
    System.out.print(str[i]);
}
}
```

Output

```
D:\>javac Str_lengthDemo.java
D:\>java Str_lengthDemo
The string S is PROGRAM
The length of string is 7
The string in character array is PROGRAM
```

String in reverse direction

There is no direct method for reversing the string but we can display it in reverse direction. Following is the Java program for the same

Java Program[Str_reverse.java]

```
/******
Printing the String in reverse order
******/
class Str_reverse
{
    public static void main(String[] args)
    {
        char str[]={'S','T','R','I','N','G'};
        String S=new String(str);
        System.out.println("The string S is " +S);
        System.out.print("The string written in Reverse order ");
        for(int i=S.length()-1;i>=0;i--)
            System.out.print(str[i]);
    }
}
```

Output

```
The string S is STRING
The string written in Reverse order GNIRTS
```

3.10.2 Character Extraction

- As we know that the string is a collection of characters. **String** class provides the facility to extract the character from the String object. There is a method called **charAt(index)** with the help of which we can extract the character denoted by some index in the array.

For example :

```
String fruit=new String("mango");
```

```
char ch;  
ch=fruit.charAt(2);  
System.out.println(ch);
```

- The output of above code fragment will be n because at the index position 2 of string object fruit the character n is present.

3.10.3 String Comparison

Sometimes we need to know whether two strings are equal or not. We use method **equals()** for that purpose. This method is of Boolean type. That is, if two strings are equal then it returns true otherwise it returns false.

The syntax is -

```
Boolean equals(String Str);
```

Let us see one illustrative program -

Java Program [StringCompareDemo.java]

```
class StringCompareDemo  
{  
    public static void main(String[] args)  
    {  
        String str1=new String("INDIA");  
        String str2=new String("india");  
        if(str1.equals(str2)==true)  
            System.out.println("\n The two strings are equal");  
        else  
            System.out.println("\n The two strings are not equal");  
    }  
}
```

Output

```
D:\>javac StringCompareDemo.java
```

```
D:\>java StringCompareDemo  
The two strings are not equal
```

The above program will generate the message "The two strings are not equal" if str1 and str2 are not equal but if we write same str1 and str2 then naturally the output will be "The two strings are equal". This string comparison is case sensitive. But if we want to compare the two strings without caring for their case differences then we can use the method **equalsIgnoreCase()**.

3.10.4 Searching for the Substring

We can look for the desired substring from the given string using a method **substring()**. The syntax of method **substring()** is as follows -

String **substring**(int start_index,int end_index)

Java Program [SubstringDemo.java]

```
class SubstringDemo
{
    public static void main(String[] args)
    {
        String str1=new String("I love my country very much");
        System.out.println("\n One substring from the given sentence
                           is:"+str1.substring(2,6));
        System.out.println("\n And another substring
                           is:"+str1.substring(18));
    }
}
```

Output

D:\>javac SubstringDemo.java

D:\>java SubstringDemo

One substring from the given sentence is:love
And another substring is:very much

3.10.5 Replacing the Character from String

We can replace the character by some desired character. For example

Java Program [replaceDemo.java]

```
class replaceDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        String s=str.replace('i','a');
        System.out.println(s);
    }
}
```

Output

D:\>javac replaceDemo.java

D:\>java replaceDemo

Nasha as Indaan

3.10.6 Upper and Lower Case

We can convert the given string to either upper case or lower case using the methods **toUpperCase()** and **toLowerCase()**. Following program demonstrates the handling of the case of the string -

Java Program [CaseDemo.java]

```
class caseDemo
{
    public static void main(String[] args)
    {
        String str=new String("Nisha is Indian ");
        System.out.println("\n The original string is: " +str);
        String str_upper= str.toUpperCase();
        System.out.println("\n The Upper case string is: " +str_upper);
        String str_lower= str.toLowerCase();
        System.out.println("\n The Lower case string is: " +str_lower);
    }
}
```

Output

```
D:\>javac caseDemo.java
```

```
D:\>java caseDemo
```

```
The original string is: Nisha is Indian
```

```
The Upper case string is: NISHA IS INDIAN
```

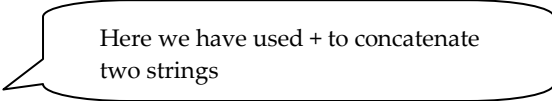
```
The Lower case string is: nisha is Indian
```

3.10.7 Concatenating Strings

We can concatenate two strings using + operator which is illustrated by following code -

Java Program [Test.java]

```
class Test
{
    public static void main(String[] args)
    {
        String fruit=new String("mango");
        System.out.println("I like " +fruit + " very much");
    }
}
```



Here we have used + to concatenate two strings

Output

```
I like mango very much
```

We can make use of a method **concat()** for concatenating two strings. The above program is slightly modified for the use of **concat()** -

Java Program

```
class Test
{
public static void main(String[] args)
{
String str=new String("I like ");
String fruit=new String("mango very much");
System.out.println(str.concat(fruit));
}
}
```

Output

I like mango very much

Ex. 3.10.1 : Write a Java Program that collects the input as a decimal number of integer type and converts it into a String of equivalent hexadecimal number.

IT: Dec.-13, Marks 8

Sol. :

```
import java.io.*;
class DecToHex
{
public static void main(String[] args) throws IOException
{
//Following 3 statements is required to read the input through keyboard
BufferedReader br=new BufferedReader (new InputStreamReader(System.in));

System.out.print("Enter a decimal number : ");
int num=Integer.parseInt(br.readLine());

int rem;
String str="";
//array storing the digits (as characters) in a hexadecimal number system
char digits[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
while(num>0)//divide until num is 0
{
rem=num%16; //finding remainder by dividing the number by 16
str=digits[rem]+str; //adding the remainder to the result
num=num/16;
}
System.out.println("Output = "+str);
}
}
```

Output

Enter a decimal number : 10
Output = A

Ex. 3.10.2 : Write a Java Program that arranges the given set of strings in alphabetical order. Supply the strings through the command line

AU : IT: Dec.-13, Marks 8

Sol. :

```
class StringDemo
{
    public static void main(String[] args)
    {
        String temp;
        int n = args.length;
        int i,j,c;
        for (i=0; i<n-1; i++)
        {
            for (j=i+1; j<n; j++)
            {
                c = args[i].compareTo(args[j]);
                if (c >0)
                {
                    temp = args[i];
                    args[i] = args[j];
                    args[j] = temp;
                }
            }
        }
        for (i=0; i<n ;i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

Output

```
D:\>javac StringDemo.java
D:\>java StringDemo Swapna Radha Archana Shilpa
Archana
Radha
Shilpa
Swapna
```

Two Marks Questions with Answers**Q.1 In Java what is the use of interface ?****AU : CSE : Dec.-10****OR What is interface mention its use.****AU : IT : Dec.-11,19**

Ans. : In java, the interface is used to specify the behaviour of a group of classes. Using interfaces the concept of multiple inheritance can be achieved.

Q.2 Define Interface and write the syntax of the Interface.**AU : CSE : Dec.-12**

Ans. : The interface can be defined using following syntax

```
access_modifier interface name_of_interface
{
    return_type method_name1(parameter1,parameter2,...parametern);
    ...
    return_type method_name1(parameter1,parameter2,...parametern);
}
```

```
type static final variable_name=value;
...
}
```

The interface is a collection of various methods that can be used by the class that is attached to the interface. The interface name must be preceded by the keyword interface.

Q.3 What modifiers may be used with an interface declaration ?

Ans. : An interface may be declared as public or abstract.

Q.4 Differentiate copying and cloning.

Ans. : In object copying the copy must be done by copying each field of object to another. This is poor method of duplicating the object. But the object cloning is a method in which the object is duplicated without explicitly copying each field. It makes use of the cloneable interface to duplicate the object.

Q.5 What is object cloning ?

AU : CSE : Dec.-13,19

Ans. : The object cloning is a method of duplicating the object. It is the simplest technique in which the cloneable interface is used. This interface requires a method clone(). The clone method returns duplicated object.

Q.6 What is an anonymous inner class ?

AU : IT : Dec.- 10

Ans. : Anonymous inner class is a local class without any name. This class is created in following situations -

1. When the class has very short body.
2. Only one instance of the class is needed.
3. Class is used immediately after defining it.

Q.7 How does the declaration of a local inner class differ from the declaration of an anonymous inner class ?

AU : IT : May-11

Ans. : The local inner class is an inner class within the body of a method just like the local variable. For example -

```
Access_modifier class OuterClass
{
    //code
    Access_modifier return_type methodname(arguments)
    {
        class InnerClass
        {
            //code
        }
    }
}
```

```
    }  
    //code  
}
```

Anonymous inner class is a local class without any name. Because it has no name it can only be used once at place of declaration. For example

```
class MyInnerClass implements Runnable  
{  
    public void run()  
    {  
        System.out.println("Hello");  
    }  
}  
class DemoClass  
{  
    public static void main(String[] arg)  
    {  
        MyInnerClass my=new MyInnerClass();  
        Thread th=new Thread(my);  
        my.start();  
    }  
}
```

Q.8 State the difference between inner class and anonymous class.

AU : IT : Dec.-11

Ans. : The inner class is a kind of class within the body of a method. An inner class can have any accessibility including private. Anonymous inner class is a local class without any name. Because it has no name it can only be used once at place of declaration.

Q.9 Why the variables in interface static and final ?

Ans. : The members of interface are static and final because -

- 1) The reason for being static - The members of interface belong to interface only and not object.
- 2) The reason for being final - Any implementation can change value of fields if they are not defined as final. Then these members would become part of the implementation. An interface is pure specification without any implementation.

Q.10 What is the purpose of nested interface ?

Ans. : The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

Q.11 What are the properties of nested interface ?

Ans. :

- 1) Nested interfaces are static by default. You don't have to mark them static explicitly.
- 2) Nested interfaces declared inside class can take any access modifier.
- 3) Nested interface declared inside interface is public implicitly.

Q.12 If a class B is derived from class A and extends the interface myinterface, then how will you write this statement for class B definition ?

Ans. :

class B extends class A implements myinterface

```
{  
    ....//body of class B  
}
```



UNIT-III

4

Exception Handling

Syllabus

Exceptions - exception hierarchy - throwing and catching exceptions - built-in exceptions, creating own exceptions, Stack Trace Elements.

Contents

4.1	Exceptions.....	IT : May-12,.....	Marks 16
4.2	Exception Hierarchy.....	CSE : Dec.-10,18,19	
		IT : May-11,.....	Marks 16
4.3	Types of Exception		
4.4	Throwing and Catching Exceptions.....	CSE : Dec.-10, 12,	
		May-12, 13, 14, 15, 19	
		IT : Dec.-11, 12, 14,	Marks 16
4.5	Built in Exceptions.....	May-19	
4.6	Creating Own Exceptions.....	IT : Dec.-13,	Marks 8
4.7	Stack Trace Elements.....	CSE : Dec.-13,.....	Marks 16

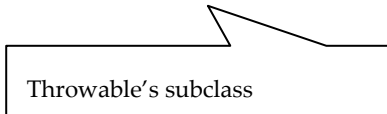
4.1 Exceptions

AU : IT : May-12, Marks 16

- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing our own exception is -
throw new Throwable's subclass
- Here the Throwable's subclass is actually a subclass derived from the **Exception** class.

For example -

```
throw new ArithmeticException();
```



- Let us see a simple Java program which illustrates this concept.

Java Program[MyExceptDemo.java]

```
import java.lang.Exception;
class MyOwnException extends Exception
{

    MyOwnException(String msg)
    {
        super(msg);
    }
}
class MyExceptDemo
{
    public static void main (String args [])
    {
        int age;
        age=15;
        try
        {
            if(age<21)
                throw new MyOwnException("Your age is very less than the condition");

        }
        catch (MyOwnException e)
        {
            System.out.println ("This is My Exception block");
            System.out.println (e.getMessage());
        }
        finally
        {

```

```
        System.out.println ("Finally block:End of the program");
    }
}
}
```

Output

This is My Exception block
Your age is very less than the condition
Finally block:End of the program

Program Explanation

- In above code, the age value is 15 and in the **try** block - the **if** condition **throws** the exception if the value is less than 21. As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block". Then the control is transferred to the **class MyOwnException**(defined at the top of the program). The message is set and it is "Your age is very less than the condition". This message can then be printed by the catch block using the `System.out.println` statement by means of `e.message`.
- At the end the finally block gets executed.

Review Question

1. Explain exception handling feature of Java in detail.

IT : May-12, Marks 16

4.2 Exception Hierarchy

AU : CSE : Dec.-10, 18, 19, IT : May-11, Marks 16

The exception hierarchy is derived from the base class **Throwable**. The **Throwable** class is further divided into two classes - **Exceptions** and **Errors**.

Exceptions : Exceptions are thrown if any kind of unusual condition occurs that can be caught. Sometimes it also happens that the exception could not be caught and the program may get terminated.

Errors : When any kind of serious problem occurs which could not be handled easily like **OutOfMemoryError** then an error is thrown.

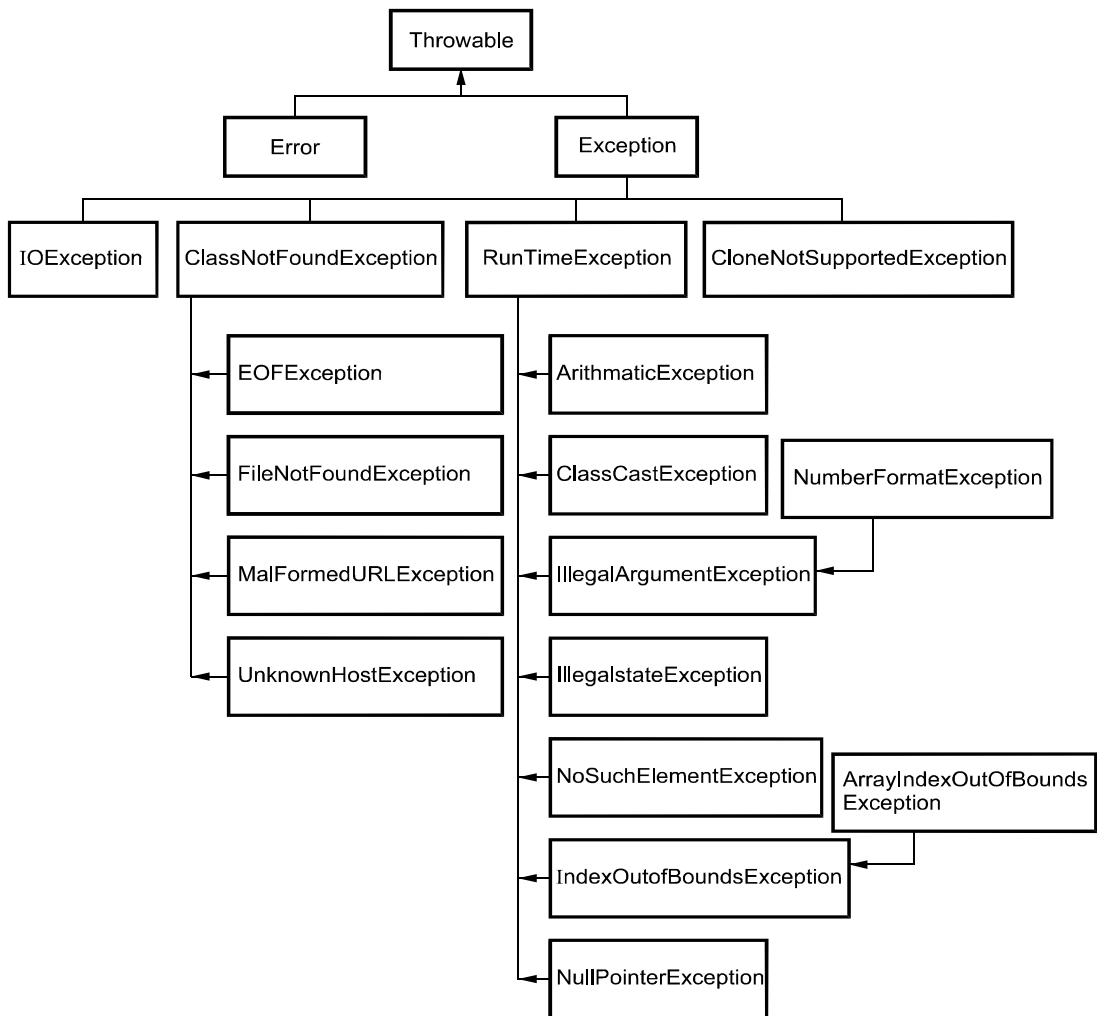


Fig. 4.2.1 Exception hierarchy

Review Questions

1. Explain the exception hierarchy. **AU : CSE : Dec.-10, Marks 8**
2. Draw the exception hierarchy in java and explain with examples throwing and catching exceptions and the common exception. **AU : IT : May-11, Marks 16**
3. Explain the different types of exceptions and the exception hierarchy in java with appropriate examples. **AU : CSE : Dec.-18, Marks 13**
4. Explain different types of exceptions in java. **AU : Dec.-19, Marks 13**

4.3 Types of Exception

- There are two type of exceptions in Java

- **Checked Exception** : These types of exceptions need to be handled explicitly by the code itself either by using the **try-catch** block or by using **throws**. These exceptions are extended from the **java.lang.Exception** class.

For example : IOException which should be handled using the try-catch block.

- **Unchecked Exception** : These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from **java.lang.RuntimeException** class.

For example : ArrayIndexOutOfBoundsException, NullPointerException, RuntimeException.

4.4 Throwing and Catching Exceptions

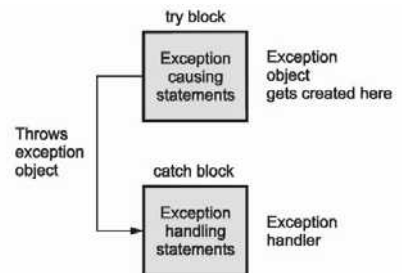
AU : CSE : Dec.-10, 12, May-12, 13, 14, 15, 19, IT : Dec.-11, 12, 14, Marks 16

- Various keywords used in handling the exception are -
 - try** - A block of source code that is to be monitored for the exception.
 - catch** - The catch block handles the specific type of exception along with the try block. Note that for each corresponding try block there exists the catch block.
 - finally** - It specifies the code that must be executed even though exception may or may not occur.
 - throw** - This keyword is used to throw specific exception from the program code.
 - throws** - It specifies the exceptions that can be thrown by a particular method.

4.4.1 try-catch Block

- The statements that are likely to cause an exception are enclosed within a **try** block. For these statements the exception is thrown.
- There is another block defined by the keyword **catch** which is responsible for handling the exception thrown by the try block.
- As soon as exception occurs it is handled by the **catch** block.
- The catch block is added immediately after the try block.
- Following is an example of **try-catch** block

```
try
{
    //exception gets generated here
}
catch(Type_of_Exception e)
{
}
```



```
//exception is handled here  
}
```

- If any one statement in the try block generates exception then remaining statements are skipped and the control is then transferred to the catch statement.

Java Program[RunErrDemo.java]

```
class RunErrDemo  
{  
    public static void main(String[] args)  
    {  
        int a,b,c;  
        a=10;  
        b=0;  
        try  
        {  
            c=a/b;  
        }  
        catch(ArithmeticException e)  
        {  
            System.out.println("\n Divide by zero");  
        }  
        System.out.println("\n The value of a: "+a);  
        System.out.println("\n The value of b: "+b);  
    }  
}
```

Exception occurs because the element is divided by 0.

Exception is handled using **catch** block

Output

Divide by zero

The value of a: 10

The value of b: 0

Note that even if the exception occurs at some point, the program does not stop at that point.

4.4.2 Uncaught Exception

- If there exists some code in the source program which may cause an exception and if the programmer does not handle this exception then java runtime system raises the exception.
- Following example illustrates how Java runtime system deals with an **uncaught exception**.
- When we use an index which is beyond the range of index then **ArrayIndexOutOfBoundsException** occurs. Following Java program illustrates it

Java Program [ExceptionProg.java]

```
class ExceptionProg
```

```
{
    static void fun(int a[])
    {
        int c;
        c=a[0]/a[2];
    }
    public static void main(String args[])
    {
        int a[]={10,5};
        fun(a);
    }
}
```

Output



- When Java runtime system detects an attempt to use an array index which is out of bound then it constructs the new exception object and throws the corresponding exception. The default handler displays the description of exception, prints the stack trace from the point at which the exception occurred and then terminates the program.
- Note that in the above output, the name of the method as **ExceptionProg.fun** and the line number 6 is displayed to represent the position problematic statements. It also displays the name of the exception as **ArrayIndexOutOfBoundsException** to represent the type of exception that is caused by the above code.

i) Demonstratic **ArrayIndexOutOfBoundsException**

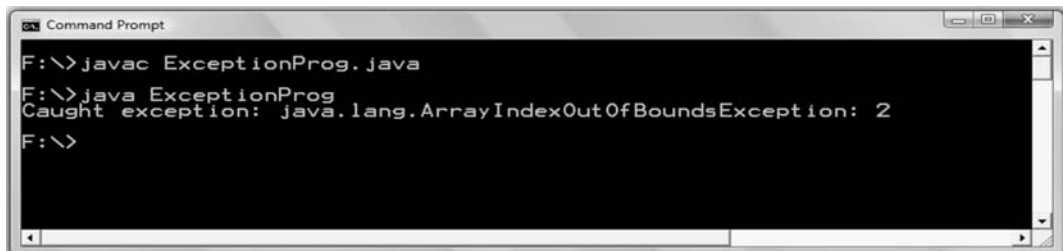
Now to handle this exception we can write the following modified code -

```
class ExceptionProg
{
    static void fun(int a[]) throws ArrayIndexOutOfBoundsException
    {
        int c;
        try
        {
            c=a[0]/a[2];//Exception occurs due to this line
        }
        catch(ArrayIndexOutOfBoundsException e)
```

```
{
System.out.println("Caught exception: "+e);
}
}
public static void main(String args[])
{
int a[]={10,5};
fun(a);
}
}
```

Then we will get following output

Output



ii) Demonstrating ArithmeticException

This type of exception occurs due to error in Math operation. Following is a simple Java program that shows illustration of this type of exception.

Ex. 4.4.1 : Write a program that includes a try block and catch clause which poses the arithmetic exception generated by divide by zero .

Sol. :

```
public class ArithmeticExceptionDemo
{
    public static void main(String args[])
    {
        try
        {
            int a=10;
            int b=0;
            int c;
            c=a/b;//Exception gets raised due to this line
        }
        catch(ArithmeticException e)
        {
            System.out.println("Caught Exception "+e);
        }
    }
}
```

Output

```
D:\test>javac ArithmeticExceptionDemo.java
```

```
D:\test>java ArithmeticExceptionDemo
Caught Exception java.lang.ArithmeticException: / by zero
```

iii) Demonstrating NumberFormatException

When we try to convert invalid string to number then **NumberFormatException** occurs. Following is a simple Java program that shows illustration of this type of exception.

```
public class NumberFormatDemo
{
    public static void main(String args[])
    {
        try
        {
            String str="Hello";
            int num=Integer.parseInt(str);//Exception gets raised due to this line
            System.out.println(num);
        }
        catch(NumberFormatException e)
        {
            System.out.println("Caught Exception "+e);
        }
    }
}
```

Output

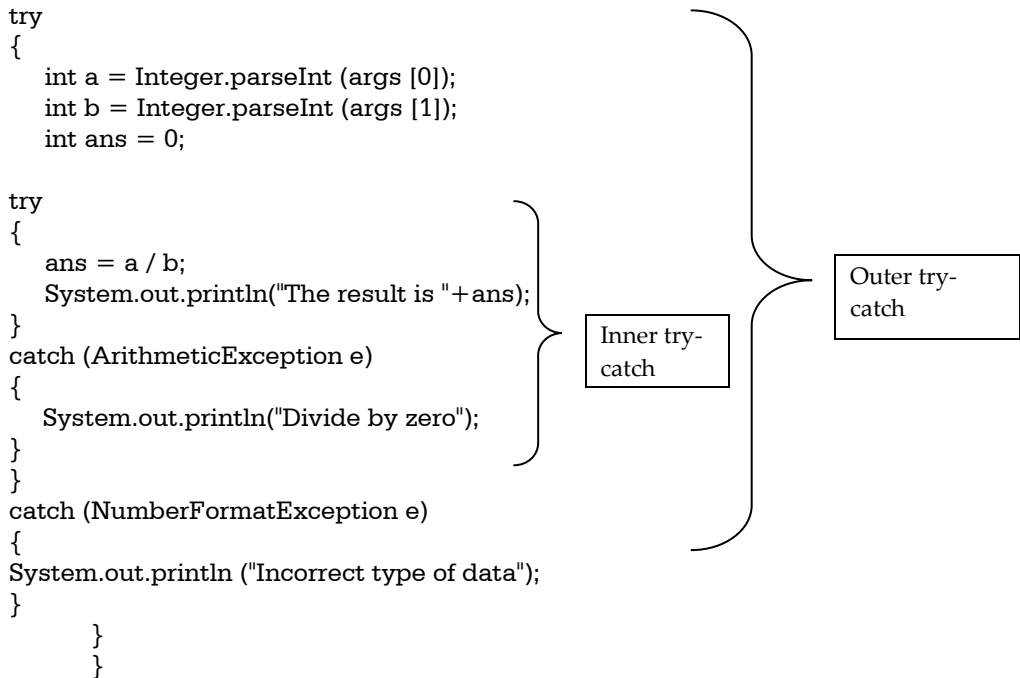
```
D:\test>javac NumberFormatDemo.java
D:\test>java NumberFormatDemo
Caught Exception java.lang.NumberFormatException: For input string: "Hello"
```

4.4.3 Nested try Statements

- When there are chances of occurring multiple exceptions of different types by the same set of statements then such situation can be handled using the nested try statements.
- Following is an example of nested try-catch statements -

Java Program[NestedtryDemo.java]

```
class NestedtryDemo
{
    public static void main (String[] args)
    {
```



Output

D:\>javac NestedtryDemo.java

D:\>java NestedtryDemo 20 10

The result is 2

D:\>java NestedtryDemo 20 a

Incorrect type of data

Outer catch handles the error

D:\>java NestedtryDemo 20 0

Divide by zero

Inner catch handles the error

4.4.4 Multiple Catch

- It is not possible for the try block to throw a single exception always.
- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.
- The syntax for single try and multiple catch is -

```

try
{
    ...
    ...//exception occurs
}

```

```
catch(Exception_type e)
{
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...//exception is handled here
}
catch(Exception_type e)
{
    ...//exception is handled here
}
```

Example

Java Program[MultipleCatchDemo.java]

```
class MultipleCatchDemo
{
    public static void main (String args [ ])
    {
        int a[] = new int [3];

        try
        {
            for (int i = 1; i <=3; i++)
            {
                a[i] = i *i;
            }

            for (int i = 0; i <3; i++)
            {
                a[i] = i/i;
            }
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println ("Array index is out of bounds");
        }
        catch (ArithmeticException e)
        {
            System.out.println ("Divide by zero error");
        }
    }
}
```

Output

Array index is out of bounds

Note : If we comment the first for loop in the try block and then execute the above code we will get following output

Divide by zero error

4.4.5 Using finally

- Sometimes because of execution of **try** block the execution gets break off. And due to this some important code (which comes after throwing off an exception) may not get executed. That means, sometimes try block may bring some unwanted things to happen.
- The **finally** block provides the assurance of execution of some important code that must be executed after the try block.
- Even though there is any exception in the try block the statements assured by **finally** block are sure to execute. These statements are sometimes called as **clean up code**. The syntax of **finally** block is

```
finally
{
    //clean up code that has to be executed finally
}
```

- The finally block always executes. The finally block is to free the resources.

Java Program [finallyDemo.java]

```
/*
This is a java program which shows the use of finally block for handling exception
*/
class finallyDemo
{
    public static void main(String args[])
    {
        int a=10,b=-1;
        try
        {
            b=a/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("In catch block: "+e);
        }
        finally
        {
```

```
    if(b!=-1)
        System.out.println("Finally block executes without occurrence of exception");
    else
        System.out.println("Finally block executes on occurrence of exception");
    }
}
```

Output

In catch block: java.lang.ArithmeticException: / by zero
Finally block executes on occurrence of exception

Program Explanation

- In above program, on occurrence of exception in try block the control goes to catch block, the exception of instance *ArithmeticException* gets caught. This is **divide by zero** exception and therefore */ by zero* will be printed as output. Following are the rules for using try, catch and finally block
 1. There should be some preceding try block for catch or finally block. Only catch block or only finally block without preceding try block is not at all possible.
 2. There can be zero or more catch blocks for each try block but there must be single finally block present at the end.

4.4.6 Using throws

- When a method wants to throw an exception then keyword **throws** is used. The syntax is -

```
method_name(parameter_list) throws exception_list
{
}
```
- Let us understand this exception handling mechanism with the help of simple Java program.

Java Program

```
/* This programs shows the exception handling mechanism using throws
*/
class ExceptionThrows
{
    static void fun(int a,int b) throws ArithmeticException
    {
        int c;
        try
        {
            c=a/b;
```

```
}
catch(ArithmeticException e)
{
    System.out.println("Caught exception: "+e);
}

}

public static void main(String args[])
{
    int a=5;
    fun(a,0);
}
}
```

Output

Caught exception: java.lang.ArithmeticException: / by zero

- In above program the method *fun* is for handling the exception *divide by zero*. This is an arithmetic exception hence we write

static void fun(int a,int b) throws ArithmeticException

- This method should be of *static* type. Also note as this method is responsible for handling the exception the **try-catch** block should be within *fun*.

4.4.7 Using throw

- For explicitly throwing the exception, the keyword **throw** is used. The keyword **throw** is normally used within a method. We can not throw multiple exceptions using **throw**.

Java Programming Example

```
class ExceptionThrow
{
    static void fun(int a,int b)
    {
        int c;
        if(b==0)
            throw new ArithmeticException("Divide By Zero!!!");
        else
            c=a/b;
    }
    public static void main(String args[])
    {
        int a=5;
        fun(a,0);
    }
}
```

```
}
}
```

Output

Exception in thread "main" java.lang.ArithmeticException: Divide By Zero!!!
 at ExceptionThrow.fun(ExceptionThrow.java:7)
 at ExceptionThrow.main(ExceptionThrow.java:14)

Difference between throw and throws

Throw	Throws
For explicitly throwing the exception, the keyword throw is used.	For declaring the exception the keyword throws is used.
Throw is followed by instance.	Throws is followed by exception class.
Throw is used within the method.	Throw is used with method signature.
We cannot throw multiple exceptions.	It is possible to declare multiple exceptions using throws.
Example - Refer section 4.4.7.	Example - Refer section 4.4.6.

Review Questions

1. Explain the throwing and catching the exception in Java

AU : CSE : Dec.-10, Marks 8, Dec.-12, Marks 16, IT : Dec.-12, Marks 16

2. Discuss the concept of exception handling with an application of your choice. Write necessary code snippets.

AU : IT : Dec.-11, Marks 16

3. Discuss on exception handling in detail.

AU : CSE : May-12, Marks 16

4. What is exception ? How to throw an exception ? Given an example.

AU : CSE : May-13, Marks 8

5. What is finally class ? How to catching exceptions ? Write an example.

AU : CSE : May-13, Marks 8

6. How are Exceptions handled in Java? Elaborate with suitable examples

AU : May-14, Marks 16

7. With relevant examples discuss exception handling in Java

AU : Dec.-14, Marks 16

8. Discuss about the Java error handling mechanism? What is the difference between 'unchecked exceptions' and 'checked exceptions'? What is the implication of catching all the exceptions with the type 'Exception'?

AU : May-15, Marks 8

9. Give an example for nested try statements in Java source file and explain.

AU : May-19, Marks 7

(Hint : Refer section 4.3 for checked and unchecked exception.)

4.5 Built in Exceptions**AU : May-19**

- Various common exception types and causes are enlisted in the following table -

Exception	Description
ArithmeticException	This is caused by error in Math operations. For e.g. Divide by zero.
NullPointerException	Caused when an attempt to access an object with a Null reference is made.
IOException	When an illegal input/output operation is performed then this exception is raised.
IndexOutOfBoundsException	An index when gets out of bound ,this exception will be caused.
ArrayIndexOutOfBoundsException	Array index when gets out of bound, this exception will be caused.
ArrayStoreException	When a wrong object is stored in an array this exception must occur.
EmptyStackException	An attempt to pop the element from empty stack is made then this exception occurs
NumberFormatException	When we try to convert an invalid string to number this exception gets caused.
RuntimeException	To show general run time error this exception must be raised.
Exception	This is the most general type of exception
ClassCastException	This type of exception indicates that one tries to cast an object to a type to which the object can not be casted.
IllegalStateException	This exception shows that a method has been invoked at an illegal or inappropriate time. That means when Java environment or Java application is not in an appropriate state then the method is invoked.

Review Question

1. Write a note on built in exceptions.

AU : May-19, Marks 6**4.6 Creating Own Exceptions****AU : IT : Dec.-13, Marks 8**

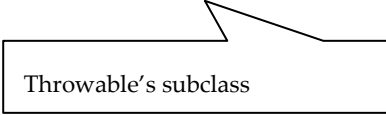
- We can throw our own exceptions using the keyword **throw**.
- The syntax for throwing out own exception is -

throw new Throwable's subclass

- Here the Throwable's subclass is actually a subclass derived from the **Exception** class.

For example -

```
throw new ArithmeticException();
```



Throwable's subclass

- Let us see a simple Java program which illustrates this concept.

Java Program[MyExceptDemo.java]

```
import java.lang.Exception;
class MyOwnException extends Exception
{

    MyOwnException(String msg)
    {
        super(msg);
    }
}
class MyExceptDemo
{
    public static void main (String args [])
    {
        int age;
        age=15;
        try
        {
            if(age<21)
                throw new MyOwnException("Your age is very less than the condition");

        }
        catch (MyOwnException e)
        {
            System.out.println ("This is My Exception block");
            System.out.println (e.getMessage());
        }
        finally
        {
            System.out.println ("Finally block:End of the program");
        }
    }
}
```

Output

This is My Exception block
Your age is very less than the condition
Finally block:End of the program

Program Explanation

- In above code, the age value is 15 and in the **try** block - the **if** condition **throws** the exception if the value is less than 21. As soon as the exception is thrown the catch block gets executed. Hence as an output we get the first message "This is My Exception block". Then the control is transferred to the **class MyOwnException**(defined at the top of the program). The message is set and it is "Your age is very less than the condition". This message can then be printed by the catch block using the `System.out.println` statement by means of **e.message**.
- At the end the finally block gets executed.

Ex. 4.6.1 : Write an exception class for a time of day that can accept only 24 hour representation of clock hours. Write a java program to input various formats of timings and throw suitable error messages.

AU : IT : Dec.-13, Marks 8

Sol. :

```
import java.lang.Exception;
import java.io.*;
import java.util.*;

class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}

class Clock
{
    private int hour;
    private int minute;
    public void input()throws IOException
    {
        BufferedReader br=new BufferedReader (new InputStreamReader(System.in));
        System.out.println("\n Enter the time in hh:mm format");
        String str=br.readLine();
        StringTokenizer tokn=new StringTokenizer(str,":");
```

```
        String h=token.nextToken();
        String m=token.nextToken();
        hour=Integer.parseInt(h);
        minute=Integer.parseInt(m);
        try
        {
            System.out.println("Hour: "+hour);
            if((hour < 0) || (hour >24))
                throw new MyException("Fatal error: invalid hour");
        }
        catch(MyException e)
        {
            System.out.println(e.getMessage());
        }
        try
        {
            System.out.println("Minute: "+minute);
            if((minute <0) || (minute > 59))
                throw new MyException("Fatal error: invalid minute");
        }
        catch(MyException e)
        {
            System.out.println(e.getMessage());
        }
    }
}

class ClockDemo
{
    public static void main(String[] args)throws IOException
    {
        Clock c=new Clock();
        c.input();
    }
}
```

Output(Run1)

```
Enter the time in hh:mm format
25:80
Hour: 25
Fatal error: invalid hour
Minute: 80
Fatal error: invalid minute
```

Output(Run2)

```
Enter the time in hh:mm format
10:70
Hour: 10
```

Minute: 70

Fatal error: invalid minute

Output(Run3)

Enter the time in hh:mm format

30:40

Hour: 30

Fatal error: invalid hour

Minute: 40

Review Question

1. With suitable Java program, explain user defined exception handling.

4.7 Stack Trace Elements

AU : CSE : Dec.-13, Marks 16

The stack trace element is created to represent the specific execution point. The syntax of **StackTraceElement** is.

StackTraceElement (String ClassName, String methodName, String filename, int lineNumber);

Where

className represents the name of the class for which the execution point is specified.

methodName represents the name of the method containing execution point which is represented by the **StackTraceElement**

filename represents the name of the file containing the execution point which is to be represented by **StackTraceElement**

lineNumber represents the source code line which represents the execution point. If this is a negative value then it indicates that the desired information is unavailable. If the value is -2 then it indicates that method containing execution point is a native method.

Various methods of **StackTraceElement** are -

Method	Description
public boolean equals(Object obj)	Returns true if the specified object is another StackTraceElement instance representing the same execution point as this instance.
String getClassName()	Returns the full class name containing execution point represented by the StackTraceElement .
String getFileName()	Represents the name of the file containing execution point represented by StackTraceElement .

int getLineNumber()	Represents the name of the line containing execution point represented by StackTraceElement .
int hashCode()	It returns the hash code element for the stack trace element.
boolean isNativeMethod()	It represents whether the method containing is native or not, using StackTraceElement .
String toString()	It returns a string represented by the StackTraceElement .

Review Question

1. What is meant by exceptions ? Why it is needed ? Describe the exception hierarchy. Write note on Stack Trace Elements. Give example.

AU : CSE : Dec.-13, Marks 16**Two Marks Questions with Answers****Q.1 What is an exception ? Give example.****AU : IT, CSE : May-12**

Ans : Exception is a mechanism which is used for handling unusual situation that may occur in the program. For example -

ArithmeticException : This exception is used to handle arithmetic exceptions such as divide by zero.

IOException : This exception occurs when an illegal input/output operation is performed.

Q.2 What will happen if an exception is not caught ?**AU : CSE : Nov.-10**

Ans. : An uncaught exception results in invoking of the `uncaughtException()` method. As a result eventually the program will terminate in which it is thrown.

Q.3 Give any methods available in Stack Trace Element.**AU : CSE : Dec.-10**

Ans. : `int getLineNumber()` - This method represents the name of the line containing execution point represented by `StackTraceElement`.

Q.4 What are stack trace elements.**AU : CSE : Dec.-12**

Ans. : Stack trace element is created to represent the specific execution point. The syntax of stack trace element is `StackTraceElement(String ClassName, String MethodName, String filename,int lineNumber);`

The class name represents the name of the class for which the execution point is specified.

Q.5 What is the benefit of exception handling ?

Ans. : When calling method encounters some error then the exception can be thrown. This avoids crashing of the entire application abruptly.

Q.6 What is the difference between error and exception in java?

Ans. : Errors are mainly caused by the environment in which an application is running. For example, `OutOfMemoryError` happens there is shortage of memory. Where as exceptions are mainly caused by the application itself. For example, `NullPointerException` occurs when an application tries to access null object.

Q.7 What is compile time and run time error ?

Ans. :

1. The errors that are detected by the Java compiler during the compile time are called compiler time errors.
2. The runtime errors are basically the logically errors that get caused due to wrong logic.

Q.8 What is the use of try, catch keywords ?

Ans. : try - A block of source code that is to be monitored for the exception.

catch - The catch block handles the specific type of exception along with the try block.

Q.9 What is the difference between throw and throws ?

Ans. :

1. The throw keyword is used to explicitly throw an exception.
2. The throws keyword is used to declare an exception.

Q.10 What is `ArrayIndexOutOfBoundsException` ?

Ans. : When we use an array index which is beyond the range of index then `ArrayIndexOutOfBoundsException` occurs.

Q.11 What is the need of multiple catch ?

Ans. :

- There may be the situations in which different exceptions may get raised by a single try block statements and depending upon the type of exception thrown it must be caught.
- To handle such situation multiple catch blocks may exist for the single try block statements.

Q.12 There are three statements in a try block – statement1, statement2 and statement3. After that there is a catch block to catch the exceptions occurred in the try block. Assume that exception has occurred in statement2. Does statement3 get executed or not?

Ans. : No. Once a try block throws an exception, remaining statements will not be executed. Control comes directly to catch block.

Q.13 Explain the types of exceptions.

Ans. : There are two types of exceptions :

1. **Checked Exception :** These types of exceptions need to be handled explicitly by the code itself either by using the try-catch block or by using throws. These exceptions are extended from the `java.lang.Exception` class.

For example : `IOException` which should be handled using the try-catch block.

2. **Unchecked Exception :** These type of exceptions need not be handled explicitly. The Java Virtual Machine handles these type of exceptions. These exceptions are extended from `java.lang.RuntimeException` class.

For example : `ArrayIndexOutOfBoundsException`, `NullPointerException`, `RunTimeException`.

Q.14 Does finally block get executed If either try or catch blocks are returning the control ?

Ans. : Yes, finally block will be always executed no matter whether try or catch blocks are returning the control or not.

Q.15 Can we have empty catch block ?

Ans. : Yes we can have empty catch block, but it is an indication of poor programming practice. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it will create problem while debugging the code.

Q.16 Which class is the super class for all types of errors and exceptions in java ?

Ans. : The `java.lang.Throwable` is the super class for all types of errors and exceptions in java.

Q.17 What is runtime exceptions ?

AU : Dec.-18

Ans. : `RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine. `Runtime Exception` is the parent class in all exceptions of the Java.

Q.18 What is assert keyword ?

AU : Dec.-18, 19

Ans. : The `assert` keyword is used for testing. It can be used for assumption about the program. While executing the assertion it is assumed to be true.

Q.19 What is exception handling ?**AU : Dec.-19**

Ans. : Exception handling is a mechanism in which the statements that are likely to cause an exception are enclosed within try block. As soon as exception occurs it is handled using catch block.

Thus exception handling is a mechanism that prevents the program from crashing when some unusual situation occurs.

Q.20 What happens when the statement `int value = 25/0` is executed ?**AU : May-19**

Ans. : The exception Arithmetic Exception : Divide by zero will occur.



UNIT-III

5

Input and Output

Syllabus

Input / Output Basics - Streams - Byte streams and Character streams - Reading and Writing Console - Reading and Writing Files.

Contents

- 5.1 Streams
- 5.2 Byte Streams and Character Streams **IT : May-13, 15,**..... Marks 8
- 5.3 Reading and Writing Console
- 5.4 Reading and Writing Files **Dec.-19, May-19** Marks 13

5.1 Streams

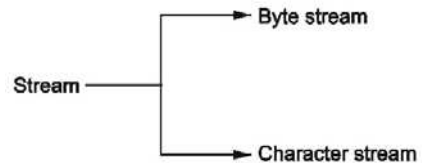
Definition :

- Stream is basically a channel on which the data flow from sender to receiver.
- An input object that reads the stream of data from a file is called **input stream**.
- The output object that writes the stream of data to a file is called **output stream**.

5.2 Byte Streams and Character Streams

AU : IT : May-13, 15, Marks 8

- In Java input and output operations are performed using streams. And Java implements streams within the classes that are defined in **java.io**.
- There are basically two types of streams used in Java.



5.2.1 Byte Stream

- The byte stream is used for inputting or outputting the bytes.
- There are two super classes in byte stream and those are **InputStream** and **OutputStream** from which most of the other classes are derived.
- These classes define several important methods such as **read()** and **write()**.
- The input and output stream classes are as shown in Fig. 5.2.1 and Fig. 5.2.2.

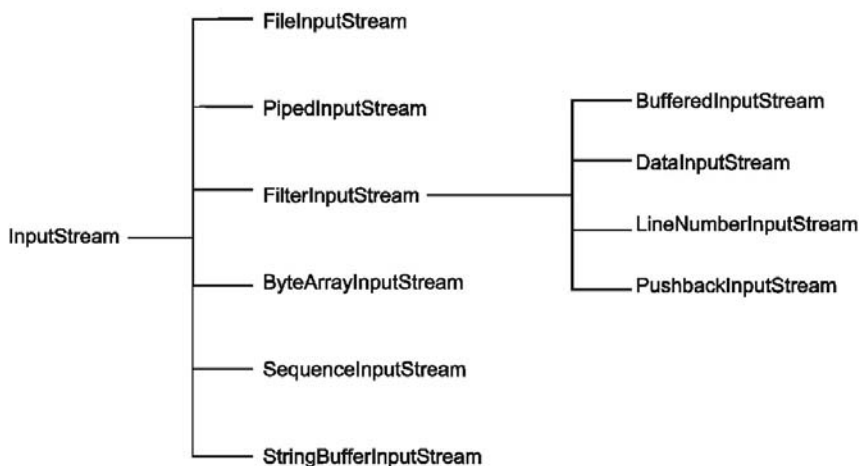


Fig. 5.2.1 InputStream classes

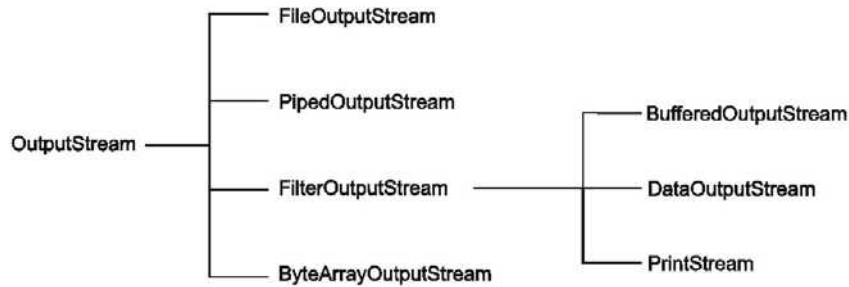
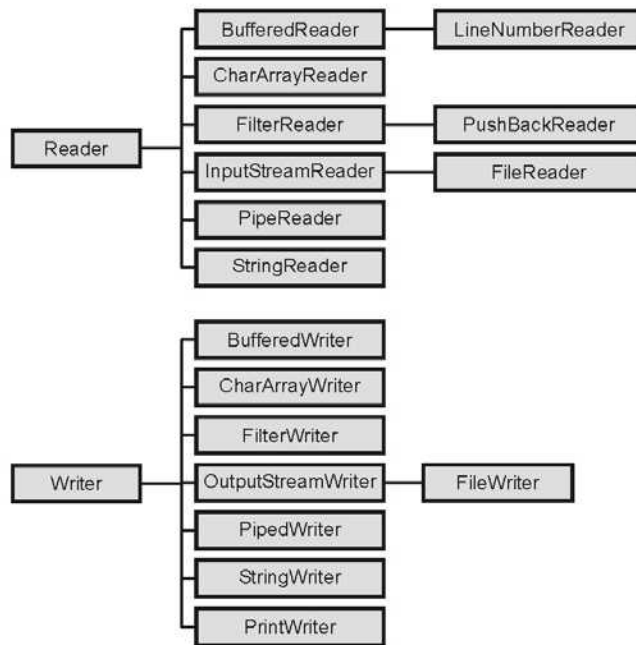


Fig. 5.2.2 OutputStream classes

5.2.2 Character Stream

- The character stream is used for inputting or outputting the characters.
- There are two super classes in character stream and those are **Reader** and **Writer**.
- These classes handle the Unicode character streams.
- The **two important methods** defined by the character stream classes are **read()** and **Write()** for reading and writing the characters of data.
- Various Reader and Writer classes are –

FileReader	FileWriter
PipeReader	PipeWriter
FilterReader	FilterWriter
BufferedReader	BufferedWriter
DataReader	DataWriter
LineNumberReader	LineNumberWriter
PushbackReader	PushbackWriter
ByteArrayReader	ByteArrayWriter
SequenceReader	SequenceWriter
StringBufferReader	StringBufferWriter

**Fig. 5.2.3**

Reader and **InputStream** define similar APIs but for different data types. For example, Reader contains these methods for reading characters and arrays of characters :

```
int read()
int read(char buf[])
int read(char buf[], int offset, int length)
```

InputStream defines the same methods but for reading bytes and arrays of bytes :

```
int read()
int read(byte buf[])
int read(byte buf[], int offset, int length)
```

Writer and **OutputStream** are similarly parallel. **Writer** defines these methods for writing characters and arrays of characters :

```
int write(int c)
int write(char buf[])
int write(char buf[], int offset, int length)
```

And **OutputStream** defines the same methods but for bytes :

```
int write(int c)
int write(byte buf[])
int write(byte buf[], int offset, int length)
```

5.2.3 Comparison between Byte Stream and Character Stream

Sr.No.	Byte Stream	Character Stream
1.	The byte stream is used for inputting and outputting the bytes.	The character stream is used for inputting and outputting the characters.
2.	There are two super classes used in byte stream and those are – InputStream and OutputStream	There are two super classes used in byte stream and those are – Reader and Writer
3.	A byte is a 8-bit number type that can represent values from – 127 to 127. Ascii values can be represented with exactly 7 bits.	A character is a 16 bit number type that represents Unicode.
4.	It never supports Unicode characters	It supports Unicode characters.

Review Questions

1. What is meant by stream ? What are the types of streams and classes ? Explain in detail.

AU : IT : May-13, Marks 8

2. Explain I/O streams with suitable example

AU : May-15, Marks 8**5.3 Reading and Writing Console****5.3.1 Reading Console Input**

- In this section, we will understand how to *read the input from console*?
- In Java, **System** is a class defined in **java.lang** and **in**, **out** and **err** are the variables of the class **System**.
- Hence **System.out** is an object used for standard output stream and **System.in** and **System.err** are the objects for standard input stream and error.

Method 1 : Reading input using Buffered Reader, Inputstream Reader, system. in

- When we want to read some character from the console we should make use of **system.in**.
- The character stream class **BufferedReader** is used for this purpose. In fact we should pass the variable **System.in** to **BufferedReader** object. Along with it we should also mention the abstract class of **BufferedReader** class which is **InputStreamReader**.

- Hence the syntax is,

```
BufferedReader object_name =new  
BufferedReader(new InputStreamReader(System.in));
```

Enables us to read
console input

For example the object named *obj* can be created as

```
BufferedReader obj =new BufferedReader(new  
InputStreamReader(System.in));
```

- Then, using this object we invoke **read()** method. For e.g. **obj.read()**.
- But this is not the only thing needed for reading the input from console; we have to mention the exception **IOException** for this purpose. Hence if we are going to use **read()** method in our program then function **main** can be written like this –

```
public static void main(String args[]) throws IOException
```
- Let us understand all these complicated issues with the help of some simple Java program

Java program [ReadChar.java]

```
/*  
This is a java program which is for reading the input from console  
*/  
import java.io.*;  
class ReadChar  
{  
    public static void main(String args[])  
        throws IOException  
    {  
        //declaring obj for read() method  
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));  
        int count=0;  
        char ch;  
        System.out.println("\n Enter five characters");  
        while(count<5)  
        {  
            ch=(char)obj.read();//reading single character  
            System.out.println(ch);//outputting it  
            count++; //count to keep track of 5 characters  
        }  
    }  
}
```

Step 1 : Creating instance
for reading console input

Step 2 : Actually reading
character

Output

```
Enter five characters
hello
h
e
l
l
o
```

Program Explanation

- This program allows you take the input from console.
- As given in above program, **read()** method is used for reading the input from the console.
- The method **read()** returns the integer value, hence it is typecast to **char** because we are reading characters from the console.

- And last but not least we should write,

`import java.io.*;`

in our java program, because these I/O operations are supported by the java package **java.io**.

Let us discuss one more example of reading input from console.

Ex. 5.3.1 : Write a Java program to read entire line through keyboard.

Sol. : Java Program [ReadString.java]

```
/*
This is a java program which is for reading the input from console
*/
import java.io.*;
class ReadString
{
    public static void main(String args[])
    throws IOException
    {
        BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
        String s;
        s=obj.readLine();//for reading the string
        while(!s.equals("end"))
        {
            System.out.println(s);
            s=obj.readLine();
        }
    }
}
```

Reading input using BufferedReader
InputStreamReader and System.in

Output

```
hello how are you  
hello how are you  
end
```

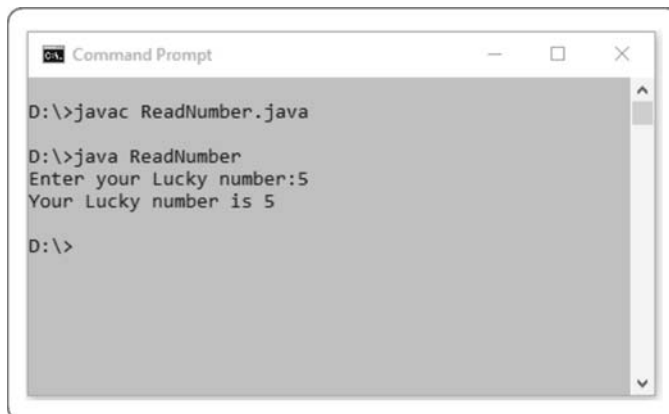
Note that to read the string input we have used **readLine()** function. Otherwise this program is almost same as previous one.

Ex. 5.3.2 : Write a Java Program by which user can enter his/her lucky number using keyboard. This number should also be displayed on the console.

Sol. :

```
import java.io.*;  
public class ReadNumber  
{  
    public static void main(String[] args)throws IOException  
    {  
        String s;  
        int num;  
        BufferedReader reader;//create BufferedReader object  
        reader=new BufferedReader(new InputStreamReader(System.in));  
        System.out.print("Enter your Lucky number:");  
        s=reader.readLine();  
        num=Integer.parseInt(s); //string value is converted to integer value  
        System.out.println("Your Lucky number is "+num);  
    }  
}
```

Output



Method 2 : Reading Console input using Scanner and System.in

- The Scanner is a class defined in java.util package. With the help of **System.in** we can read the console input by creating the instance for the class Scanner.
- The steps to read the console input are as follows
 - **Step 1** : Import java.util package
 - **Step 2** : Create instance for Scanner class as follows

```
Scanner scanner = new Scanner(System.in);
```

- **Step 3** : Invoke the nextXX() method to read the input of specific data type

- **Example 1 : Reading String**

```
import java.util.*;           //Step 1
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in); //Step 2
        System.out.print("What is your Name? ");
        String name = scanner.next(); //Step 3
        System.out.println("My Name is: " + name);
    }
}
```

Output

```
What is your Name? Ankita
My Name is: Ankita
```

- **Example 2 : Reading integer**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your Lucky number ");
        int num = scanner.nextInt();
        System.out.println("My Lucky number is: " + num);
    }
}
```

Output

```
Enter your Lucky number 5
My Lucky number is: 5
```

- **Example 3 : Reading float**

```
import java.util.*;
public class Example
```

```
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your marks ");
        float num = scanner.nextFloat();
        System.out.println("My marks are: " + num);
    }
}
```

- **Example 4 : Reading double**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter double number ");
        double num = scanner.nextDouble();
        System.out.println("You have entered: " + num);
    }
}
```

Output

```
Enter double number 212.444
You have entered : 212.444
```

- **Example 5 : Reading boolean**

```
import java.util.*;
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter boolean value as true or false ");
        boolean b = scanner.nextBoolean();
        System.out.println("You have entered: " + b);
    }
}
```

Output

```
Enter boolean value as true or false false
You have entered: false
```

Ex. 5.3.3 : Write a Java program for calculating the bill amount. User enters the price of the item and its quantity.

Sol. :

```
import java.util.*;
```

```
public class Example
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the price: ");
        float price = scanner.nextFloat();
        System.out.print("Enter the quantity: ");
        int quantity= scanner.nextInt();
        float amount;
        amount=price*quantity;
        System.out.println("\nPrice: "+price);
        System.out.println("Quantity: "+quantity);
        System.out.println("-----");
        System.out.println("Total Bill in Rs: "+amount);
        System.out.println("-----");
    }
}
```

Output

```
Enter the price: 20.50
Enter the quantity: 10
```

```
Price: 20.5
Quantity: 10
-----
Total Bill in Rs: 205.0
-----
```

5.3.2 Writing Console Output

The simple method used for writing the output on the console is **write()**. Here is the syntax of using **write()** method –

```
System.out.write(int b)
```

The bytes specified by **b** has to be written on the console. But typically **print()** or **println()** is used to write the output on the console. And these methods belong to **PrintWriter** class.

5.4 Reading and Writing Files

AU : Dec.-19, May-19, Marks 13

InputStream is an abstract class for streaming the byte input. Various methods defined by this class are as follows.

Methods	Purpose
int available()	It returns total number of byte input that are available currently for reading.
void close()	The input source gets closed by this method. If we try to read further after closing the stream then IOException gets generated.
void mark(int n).	This method places a mark in the input stream at the current point. This mark remains valid until the n bytes are read.
boolean markSupported()	It returns true if mark() or reset() are supported by the invoking stream.
void reset()	This method resets the input pointer to the previously set mark.
long skip(long n)	This method ignores the n bytes of input. It then returns the actually ignored number of bytes.
int read()	It returns the next available number of bytes to read from the input stream. If this method returns the value -1 then that means the end of file is encountered.
int read(byte buffer[])	This method reads the number of bytes equal to length of <i>buffer</i> . Thus it returns the actual number of bytes that were successfully read.
int read(byte <i>buffer</i> [], int offset,int n)	This method returns the <i>n</i> bytes into the <i>buffer</i> which is starting at offset. It returns the number of bytes that are read successfully.

OutputStream is an abstract class for streaming the byte output. All these methods under this class are of void type. Various methods defined by this class are as follows

Methods	Purpose
void close()	This method closes the output stream and if we try to write further after using this method then it will generate IOException

<code>void flush()</code>	This method clears output buffers.
<code>void write(int val)</code>	This method allows to write a single byte to an output stream.
<code>void write(byte <i>buffer</i>[])</code>	This method allows to write complete array of bytes to an output stream.
<code>void write(byte <i>buffer</i>[],int offset, int n)</code>	This method writes <i>n</i> bytes to the output stream from an array <i>buffer</i> which is beginning at <i>buffer[offset]</i>

5.4.1 **FileInputStream / FileOutputStream**

The **FileInputStream** class creates an **InputStream** using which we can read bytes from a file.

The two common constructors of **FileInputStream** are -

```
FileInputStream(String filename);
FileInputStream(File fileobject);
```

In the following Java program, various methods of **FileInputStream** are illustrated -

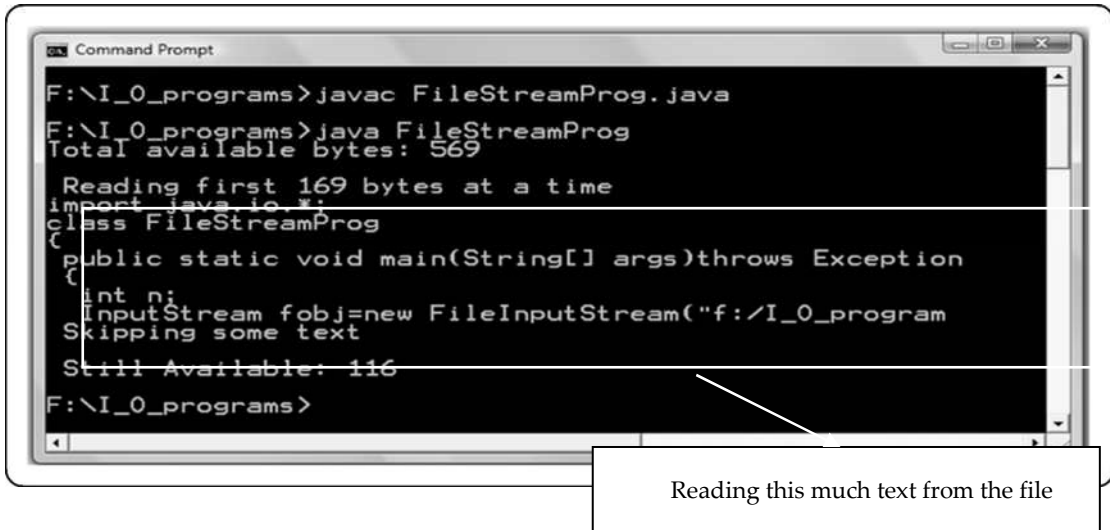
Java Program[FileStreamProg.java]

```
import java.io.*;

class FileStreamProg
{
    public static void main(String[] args)throws Exception
    {
        int n;

        InputStream fobj=new FileInputStream("f:/I_O_programs/FileStreamProg.java");
        System.out.println("Total available bytes: "+(n=fobj.available()));
        int m=n-400;
        System.out.println("\n Reading first "+m+" bytes at a time");
        for(int i=0;i<m;i++)
        System.out.print((char)fobj.read());
        System.out.println("\n Skipping some text");
        fobj.skip(n/2);
        System.out.println("\n Still Available: "+fobj.available());
        fobj.close();
    }
}
```

Output



The **FileOutputStream** can be used to write the data to the file using the **OutputStream**. The constructors are-

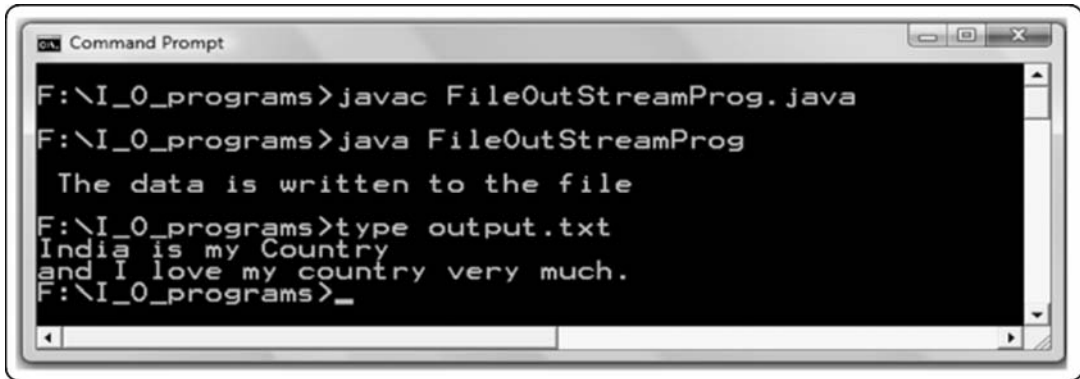
```
FileOutputStream(String filename)
FileOutputStream(Object fileobject)
FileInputStream(String filename,boolean app);
FileInputStream(String fileobject,boolean app);
```

The *app* denotes that the append mode can be true or false. If the append mode is true then you can append the data in the file otherwise the data can not be appended in the file.

In the following Java program, we are writing some data to the file.

Java Program[FileOutStreamProg.java]

```
import java.io.*;
class FileOutStreamProg
{
    public static void main(String[] args)throws Exception
    {
        String my_text="India is my Country\n"+"and I love my country very much.";
        byte b[]=my_text.getBytes();
        OutputStream fobj=new FileOutputStream("f:/I_O_programs/output.txt");
        for(int i=0;i< b.length;i++)
            fobj.write(b[i]);
        System.out.println("\n The data is written to the file");
        fobj.close();
    }
}
```

Output

```
Command Prompt
F:\I_0_programs>javac FileOutputStreamProg.java
F:\I_0_programs>java FileOutputStreamProg
The data is written to the file
F:\I_0_programs>type output.txt
India is my Country
and I love my country very much.
F:\I_0_programs>_
```

Ex. 5.4.1 : Write a program that copies the content of one file to another by removing unnecessary spaces between words

Sol. :

```
import java.io.*;
public class CopyFile
{
    private static void CopyDemo(String src, String dst)
    {
        try
        {
            File f1 = new File(src);
            File f2 = new File(dst);
            InputStream in = new FileInputStream(f1);
            OutputStream out = new FileOutputStream(f2);
            byte[] buff = new byte[1024];
            int len;
            len=in.read(buff);
            while (len > 0)
            {
                String text=new String(buff);//converting bytes to text
                text = text.replaceAll("\\s+", "");//removing unnecessary spaces
                buff=text.getBytes();//converting that text back to bytes
                out.write(buff,0,len);//writing bytes to destination file
                len=in.read(buff);//reading the remaining content of the file
            }
            in.close();
            out.close();
            System.out.println("File copied.");
        }
        catch(FileNotFoundException ex)
        {
        }
```

```
        System.out.println(ex.getMessage() + " in the specified directory.");
        System.exit(0);
    }
    catch(IOException e)
    {
        System.out.println(e.getMessage());
    }
}
public static void main(String[] args)
{
    CopyDemo(args[0],args[1]);
}
}
```

Output

```
D:\test>javac CopyFile.java
```

```
D:\test>javac CopyFile.java
```

```
D:\test>java CopyFile in.txt out.txt
```

```
File copied.
```

```
D:\test>type in.txt
```

```
This is my India.
```

```
I love my country.
```

```
D:\test>type out.txt
```

```
This is my India. I love my country.
```

5.4.2 FilterInputStream / FilterOutputStream

FilterStream class are those classes which wrap the input stream with the byte. That means using input stream you can read the bytes but if you want to read integers, doubles or strings you need to a filter class to wrap the byte input stream.

The syntax for FilterInputStream and FilterOutputStream are as follows –

```
FilterOutputStream(OutputStream o)
```

```
FilterInputStream(InputStream i)
```

The methods in the Filter stream are similar to the methods in **InputStream** and **OutputStream**

5.4.2.1 DataInputStream / DataOutputStream

DataInputStream reads bytes from the stream and converts them into appropriate primitive data types whereas the **DataOutputStream** converts the primitive data types into the bytes and then writes these bytes to the stream. The superclass of **DataInputStream** class is **FilterInputStream** and superclass of **DataOutputStream** class is **FilterOutputStream** class.

Various methods under these classes are -

Methods	Purpose
boolean readBoolean()	Reads Boolean value from the inputstream.
byte readByte()	Reads byte value from the inputstream.
char readChar()	Reads char value from the inputstream.
float readFloat()	Reads float value from the inputstream.
double readDouble()	Reads double value from the inputstream.
int readInt()	Reads integer value from the inputstream.
long readLong()	Reads long value from the inputstream.
String readLine()	Reads string value from the inputstream.
String readUTF()	Reads string in UTF form.
void writeBoolean(boolean val)	Writes the Boolean value to output stream.
void writeBytes(String str)	Writes the bytes of characters in a string to output stream.
void writeFloat(float val)	Writes the float value to output stream.
void writeDouble(float val)	Writes the double value to output stream.
void writeInt(int val)	writes the integer value to output stream.
void writeLong(long val)	writes the long value to output stream.
void writeUTF(String str)	writes the string value in UTF form to output stream.

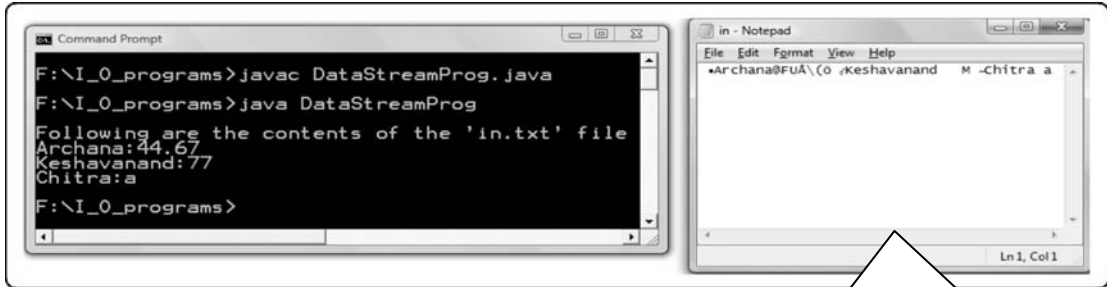
Let us see how these methods can be used in a Java Program-

Java Program[DataStreamProg.java]

```
import java.io.*;
class DataStreamProg
{
    public static void main(String[] args)throws Exception
    {
        DataOutputStream obj=new DataOutputStream
            (new FileOutputStream("in.txt"));
        obj.writeUTF("Archana");
        obj.writeDouble(44.67);
        obj.writeUTF("Keshavanand");
        obj.writeInt(77);
        obj.writeUTF("Chitra");
        obj.writeChar('a');
        obj.close();
        DataInputStream in=new DataInputStream(new FileInputStream("in.txt"));
        System.out.println("\nFollowing are the contents of the 'in.txt' file");
```

```
        System.out.println(in.readUTF()+":"+in.readDouble());
        System.out.println(in.readUTF()+":"+in.readInt());
        System.out.println(in.readUTF()+":"+in.readChar());
    }
}
```

Output



Open the **in.txt** file in the Notepad and you can see the contents that gets saved through your Java program

Program explanation

Note that in above program, along with other simple read and write methods we have used **readUTF** and **writeUTF** methods. Basically UTF is an encoding scheme that allows systems to operate with both ASCII and Unicode. Normally all the operating systems use ASCII and Java uses Unicode. The **writeUTF** method converts the string into a series of bytes in the UTF-8 format and writes them into a binary stream. The **readUTF** method reads a **string** which is written using the **writeUTF** method.

5.4.2.2 BufferedInputStream / BufferedOutputStream

The **BufferedInputStream** and **BufferedOutputStream** is an efficient class used for speedy read and write operations. All the methods of **BufferedInputStream** and **BufferedOutputStream** class are inherited from **InputStream** and **OutputStream** classes. But these methods add the buffers in the stream for efficient read and write operations. We can specify the buffer size otherwise the default buffer size is 512 bytes.

In the following Java program the **BufferedInputStream** and **BufferedOutputStream** classes are used.

Java Program[BufferedStreamProg.java]

```
import java.io.*;
class BufferedStreamProg
{
```

```

public static void main(String[] args) throws Exception
{
    DataOutputStream obj=new DataOutputStream(new BufferedOutputStream(new
FileOutputStream("in.txt")));
    obj.writeUTF(" Archana");
    obj.writeDouble(44.67);
    obj.writeUTF("Keshavanand");
    obj.writeInt(77);
    obj.writeUTF("Chitra");
    obj.writeChar('a');
    obj.close();
    DataInputStream in=new DataInputStream(new BufferedInputStream(new
FileInputStream("in.txt")));
    System.out.println("\nFollowing are the contents of the 'in.txt' file");
    System.out.println(in.readUTF()+":"+in.readDouble());
    System.out.println(in.readUTF()+":"+in.readInt());
    System.out.println(in.readUTF()+":"+in.readChar());
}
}

```

Note that the **Buffered stream class** is added here.

Program Explanation

This program is similar to the Java program we have discussed in section 5.4.2.1. The only change is we have added buffers. The creation of buffers is shown by the bold faced statements in above program.

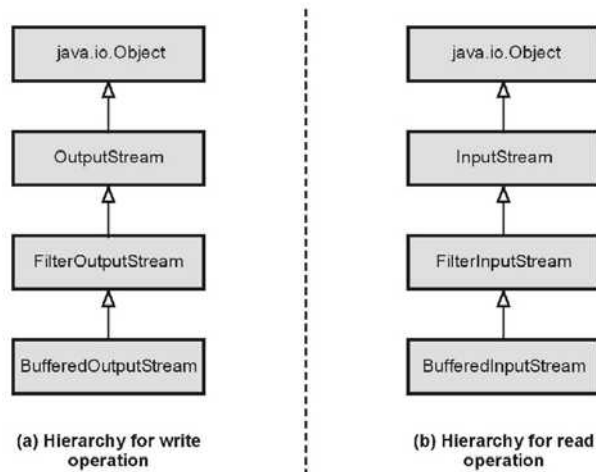


Fig. 5.4.1 Inheritance hierarchy

5.4.3 Programming Example on Reading and Writing Files

Ex. 5.4.2 : Write a program to replace all "word1" by "word2" from a file 1, and output is written to file2 file and display the number of replacement.

Sol. :

```
import java.io.*;
import java.util.*;
class FileDemo
{
    public static void main(String[] args)throws Exception
    {
        int count=0;
        File fin=new File("C:/lab/word.txt");
        BufferedReader br=new BufferedReader(new FileReader(fin));

        String FullStr="";
        String TempStr;
        while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
        {
            //then it is collected in one string separated by spaces
            FullStr+=TempStr+" ";
        }
        Scanner input=new Scanner(System.in);
        System.out.print("Enter the word to be searched from the file: ");
        String word1=input.next();

        String Word_List[]=FullStr.split(" ");//Extract words from string for counting
                                                //no.of occurrences

        //Searching for the word1
        for(int i=0;i<Word_List.length;i++)
        {
            if(Word_List[i].equals(word1))
            {
                count++; //counting no. of occurrences
            }
        }

        System.out.print("\nEnter new Word for replacement: ");
        String word2=input.next();

        String New_Str=FullStr.replace(word1,word2);

        //File is opened for writing purpose
        File fout=new File("C:/lab/Newword.txt");
        BufferedWriter bw=new BufferedWriter(new FileWriter(fout));
        //Modified contents are written to the file
```

```

        bw.write(New_Str);
        System.out.println("\n The replacement is done and file is rewritten");
        System.out.println("\nThe number of replacements are "+count);
        bw.close();

        br=new BufferedReader(new FileReader(fout));
        System.out.println("\n Now,the Contents of the file are...\n ");
        while((TempStr=br.readLine())!=null)
            System.out.println(TempStr);
        br.close();

    }
}
/*****
<word.txt>
*****/
PHP is fun.
Programming in PHP is interesting.
People love PHP

```

Output

Enter the word to be searched from the file: PHP

Enter new Word for replacement: Java

The replacement is done and file is rewritten

The number of replacements are 3

Now,the Contents of the file are...

Java is fun. Programming in Java is interesting. People love Java

Ex. 5.4.3 : Write a program to replace all "word1" by "word2" to a file without using temporary file and display the number of replacement.

Sol. :

```

import java.io.*;
import java.util.*;
class FileDemo
{
    public static void main(String[] args)throws Exception
    {
        int count=0;
        File fname=new File("C:/lab/word.txt");
        BufferedReader br=new BufferedReader(new FileReader(fname));

```

```
String FullStr="";
    String TempStr;
while((TempStr=br.readLine())!=null)//if multi-line input is present in the file
{
    //then it is collected in one string separated by spaces
    FullStr+=TempStr+" ";
}
Scanner input=new Scanner(System.in);
System.out.print("Enter the word to be searched from the file: ");
String word1=input.next();

String Word_List[]=FullStr.split(" ");//Extract words from string for counting
                                   //no.of occurrences

//Searching for the word1
for(int i=0;i<Word_List.length;i++)
{
    if(Word_List[i].equals(word1))
    {
        count++; //counting no. of occurrences
    }
}

    System.out.print("\nEnter new Word for replacement: ");
String word2=input.next();

String New_Str=FullStr.replace(word1,word2);
    //File is opened for writing purpose
    BufferedWriter bw=new BufferedWriter(new FileWriter(fname));
//Modified contents are written to the file
bw.write(New_Str);
    System.out.println("\n The replacement is done and file is rewritten");
System.out.println("\nThe number of replacements are "+count);
bw.close();

    br=new BufferedReader(new FileReader(fname));
System.out.println("\n Now,the Contents of the file are...\n ");
while((TempStr=br.readLine())!=null)
    System.out.println(TempStr);
br.close();

}
}
```

Ex. 5.4.4 : Write a program that takes input for filename and search word from command-line arguments and checks whether that file exists or not. If exists, the program will display those lines from a file that contains given search word.

Sol. :

```
import java.io.*;
class WordSearchDemo
{
    public static void main(String args[]) throws IOException
    {
        boolean status;
        boolean WordOccur=false;
        File fobj = new File(args[0]);
        if(fobj.exists())
            status = true;
        else

        {
            status=false;
            System.out.println("\n This file does not exist!!!");
            return;
        }
        if(status)
        {
            BufferedReader in = new BufferedReader(new FileReader(fobj));
            String Line;
            String key = args[1];
            while ((Line = in.readLine()) != null)
            {
                String[] Data = Line.split(" ");
                for(String Element:Data)
                {
                    if(Element.equalsIgnoreCase(key))
                    {
                        System.out.println("""+key+"" + " is present in the line " + Line+""");
                        WordOccur = true;
                    }
                }
            }
        }
        if(!WordOccur)
            System.out.println("This Word is not present in the Input File");
    }
}
```

Output

```
E:\test\src>javac WordSearchDemo.java
E:\test\src>java WordSearchDemo input.dat java
'java' is present in the line 'learn Java Programming.'
```

Ex. 5.4.5 : Write a program that counts the no. of words in a text file. The file name is passed as a command line argument. The program should check whether the file exists or not. The words in the file are separated by white space characters.

Sol. :

```
import java.io.*;
import java.util.*;

public class WordCountDemo
{
    public static void main(String[] args) throws NullPointerException,IOException
    {

        try
        {
            BufferedReader in = new BufferedReader(new FileReader(args[0]));
            String New_Line = "", Word = "";
            int Word_Count = 0;
            while ((New_Line = in.readLine()) != null)

                Word += New_Line + " ";
            StringTokenizer Token = new StringTokenizer(Word);
            while (Token.hasMoreTokens())
            {
                String s = Token.nextToken();
                Word_Count++;
            }
            System.out.println("Text file contains " + Word_Count + " words.");
        }
        catch (NullPointerException e)
        {System.out.println("NULL PointerError: " + e.getMessage());}
        catch (IOException e)
        {System.out.println("IO Error: " + e.getMessage());}
    }
}

<input.txt>
Hello Friends
It is very interesting to
learn Java Programming.
```

Output

E:\test\src>javac WordCountDemo.java

E:\test\src>java WordCountDemo input.txt

Text file contains 10 words.

Ex. 5.4.6 : Write a program to count the total no. of chars, words, lines, alphabets, digits, white spaces in a given file.

Sol. :

```
import java.lang.*;
import java.io.*;
import java.util.*;
class WordCount
{
    public static void main(String arg[]) throws Exception
    {
        int char_count=0;
        int word_count=0;
        int line_count=0;
        int wspace_count=0;
        int digit_count=0;
        int alphabet_count=0;

        String fname;
        String temp_str;
        StringTokenizer token;

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter filename : ");
        fname=br.readLine();

        br=new BufferedReader(new FileReader(fname));
        while((temp_str=br.readLine())!=null)
        {
            line_count++;
            for(int i = 0; i < temp_str.length(); i++)
            {
                if(Character.isWhitespace(temp_str.charAt(i)))
                    wspace_count++;
                if(Character.isDigit(temp_str.charAt(i)))
                    digit_count++;
                if(Character.isLetter(temp_str.charAt(i)))
                    alphabet_count++;
            }
        }
    }
}
```

```
token=new StringTokenizer(temp_str," ");
while(token.hasMoreTokens())
{
    word_count++;
    String s=token.nextToken();
    char_count+=s.length();
}

}

System.out.println("Character Count : "+char_count);
System.out.println("Word Count : "+word_count);
System.out.println("Line Count : "+line_count);
System.out.println("Aplphabet Count : "+alphabet_count);
System.out.println("Digit Count : "+digit_count);
System.out.println("White Space Count : "+wspace_count);

br.close();
}
}
```

Sample input file: inputfile.txt

Hello123

How are u?

I am fine

Enjoying my job

Output

Enter filename : inputfile.txt

Character Count : 36

Word Count : 10

Line Count: 4

Alphabet Count : 32

Digit Count : 3

White Space Count : 6

Ex.5.4.7 Create an IN file in Java to store the details of 100 students using Student class. Read the details from IN file, Convert all the letters of IN file to lowercase letters and write it to OUT file.

AU : May-19, Marks 13

Sol. :

Step 1 : Create a Student class in a separate Java file

```
import java.io.Serializable;
public class Student implements Serializable {
    //default serialVersionUID
    private static final long serialVersionUID = 1L;

    private String first_name;
    private String last_name;
    private int age;

    public Student(String fname, String lname, int age){
        this.first_name = fname;
```

```
        this.last_name = lname;
        this.age = age;
    }

    public void setFirstName(String fname) {
        this.first_name = fname;
    }

    public String getFirstName() {
        return this.first_name;
    }

    public void setLastName(String lname) {
        this.first_name = lname;
    }

    public String getLastName() {
        return this.last_name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public int getAge() {
        return this.age;
    }
    @Override
    public String toString() {
        return new StringBuffer( "\t").append(this.first_name)
            .append("\t").append(this.last_name).append("\t").append(this.age).toString();
    }
}
```

Step 2 : Create another java file for performing file handling operations.

StudentDemo.java

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.Scanner;
public class StudentDemo
{
    private static final String filepath="D:\\IN.txt";
    public static void main(String args[])
    {
```

```
StudentDemo objectIO = new StudentDemo();
try
{
    FileOutputStream fileOut = new FileOutputStream("D:\\IN.txt");
    ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
    for(int i=0;i<100;i++)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter first name of the student: ");
        String fname= sc.next();
        System.out.println("Enter last name of the student: ");
        String lname= sc.next();
        System.out.println("Enter Age of the student: ");
        int age= sc.nextInt();
        Student student = new Student(fname,lname,age);
        objectOut.writeObject((Object)student);
    }
    System.out.println(" The File is created Successfully ");
    objectOut.close();
}
catch (Exception ex)
{
    ex.printStackTrace();
}

//Read object from IN.txt and write to OUT.txt
Student st;
try
{
    FileInputStream fileIn = new FileInputStream("D:\\IN.txt");
    ObjectInputStream objectIn = new ObjectInputStream(fileIn);
    FileOutputStream fileOut = new FileOutputStream("D:\\OUT.txt");
    ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);

    Object obj;
    for(int i=0;i<100;i++)
    {
        obj = objectIn.readObject();//reading object
        st=(Student)obj; //converting it to Student type
        String fn= st.getFirstName().toLowerCase();
        String ln= st.getLastName().toLowerCase();
        Student student = new Student(fn,ln,st.getAge());
        objectOut.writeObject((Object)student); //writing the modified object
    }
    System.out.println("The File is copied Successfully");
    objectIn.close();
    objectOut.close();
}
```

```
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
//Read object from OUT.txt
try
{
    FileInputStream fileIn = new FileInputStream("D:\\OUT.txt");
    ObjectInputStream objectIn = new ObjectInputStream(fileIn);
    Object obj;
    System.out.println("  FirstName LastName  Age");
    for(int i=0;i<100;i++)
    {
        obj = objectIn.readObject();
        System.out.println((Student)obj);
    }
    objectIn.close();
}
catch (Exception ex)
{
    ex.printStackTrace();
}
}
```

Review Question

1. Explain in detail about the following with sample program :
i) Reading from a file ii) Writing in a file.

AU : Dec.-19, Marks 13**Two Marks Questions with Answers****Q.1 What is stream ?**

Ans. : Stream is basically a channel on which the data flow from sender to receiver.

Q.2 What is input stream and output stream ?

Ans. : An input object that reads the stream of data from a file is called input stream and the output object that writes the stream of data to a file is called output stream.

Q.3 What is byte stream ? Enlist its super classes.

Ans. : The byte stream is used for inputting or outputting the bytes. The InputStream and OutputStream are the superclass of byte stream.

Q.4 What is character stream ? Enlist its super classes.

Ans. : The character stream is used for inputting or outputting the characters. The Reader and Writer are the superclass of character stream.

Q.5 List the byte stream classes.

Ans. : The Byte Stream classes are –

- | | |
|-----------------------|--------------------------|
| 1. FileInputStream | 2. PipedInputStream |
| 3. FilterInputStream | 4. ByteArrayInputStream |
| 5. FileOutputStream | 6. PipedOutputStream |
| 7. FilterOutputStream | 8. ByteArrayOutputStream |

Q.6 What is the purpose of BufferedInputStream and BufferedOutputStream classes ?

Ans. :

- These are efficient classes used for speedy read and write operations.
- We can specify Buffer size while using these classes.

Q.7 Give an example on stream.

Ans. :

- There are two types of streams - Byte stream and Character stream.
- Stream classes for byte stream : FileInputStream, PipedInputStream, BufferedInputStream, FileOutputStream, PipedOutputStream and so on.
- Stream classes for character stream : FileReader, PipeReader, BufferedReader, FileWriter, PipeWriter, BufferedWriter and so on.

Q.8 What is absolute file name ?

Ans. : The filename that can be specified with the complete path name and drive letter is called absolute file name.

Q.9 What is relative file name ?

Ans. : The file name which is specified as a path relative to the current directory is called relative file name. For example `new File("myprogram.html");`

Q.10 What is the use of seek method ?

Ans. : The `seek()` allows you to specify the index from where the read and write operations will start. The syntax is

```
void seek(long position);
```

Q.11 Write a Java code to check if the command line argument is file or not.

Ans. :

```
class Test
{
    public static void main(String [] args)
    {
        File obj=new File(args[0]);
```

```
        If(obj.isFile())
        {
            System.out.println("This is a file name"+obj.getPath());
        }
    }
}
```

Q.12 Enlist the two common constructors of FileInputStream.

Ans. : The common constructors of FileInputStream are

```
FileInputStream(String filename);
FileInputStream(File fileobject);
```

Q.13 What is the use of Input Stream Reader and Output Stream Writer ?

Ans. : The InputStreamReader converts byte into character and OutputStreamWriter converts the characters written into byte.

Q.14 Give an example for reading data from files using FileInputStream**AU : May 19**

Ans. :

```
import java.io.FileInputStream;
public class test
{
    public static void main(String args[])
    {
        try
        {
            FileInputStream fin=new FileInputStream("D:\\input.txt");
            int i=0;
            while((i=fin.read())!=-1)
            {
                System.out.print((char)i);
            }
            fin.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```



Notes

UNIT-IV

6

Multithreading

Syllabus

Differences between multi-threading and multitasking, thread life cycle, creating threads, synchronizing threads, inter-thread communication, daemon threads, thread groups.

Contents

6.1	What is Thread ?	
6.2	Difference between Multithreading and Multitasking	CSE : Dec.-19 Marks 13
6.3	Thread Life Cycle	CSE : Dec.-10, 11, May-12, 13, Marks 10
6.4	Creating Threads	IT : Dec.-10, CSE : Dec.-11, 13,18, May-13, Marks 16
6.5	Creating Multiple Threads	IT : Dec.-11,18, May-12, CSE : May-12, 19 Dec.-11,12,18, 19 Marks 16
6.6	Synchronizing Threads	Dec.-14, 19 Marks 13
6.7	Inter-thread Communication.....	IT, May-13, 19, Dec.-18 Marks 16
6.8	Daemon Threads	
6.9	Thread Groups	

6.1 What is Thread ?

- One of the exciting features of Windows operating system is that - It allows the user to handle multiple tasks together. This facility in Windows operating system is called multitasking.
- In Java we can write the programs that perform multitasking using the multithreading concept. Thus Java allows to have multiple control flows in a single program by means of multithreading.
- **Definition of thread :** Thread is a tiny program running continuously. It is sometimes called as light-weight process. But there lies differences between thread and process.

Sr. No.	Thread	Process
1.	Thread is a light-weight process.	Process is a heavy weight process.
2.	Threads do not require separate address space for its execution. It runs in the address space of the process to which it belongs to.	Each process requires separate address space to execute.

6.2 Difference between Multithreading and Multitasking

AU : Dec.-19, Marks 13

Sr. No.	Multithreading	Multiprocessing
1.	Thread is a fundamental unit of multithreading.	Program or process is a fundamental unit of multiprocessing environment.
2.	Multiple parts of a single program gets executed in multithreading environment.	Multiple programs get executed in multiprocessing environment.
3.	During multithreading the processor switches between multiple threads of the program.	During multiprocessing the processor switches between multiple programs or processes.
4.	It is cost effective because CPU can be shared among multiple threads at a time.	It is expensive because when a particular process uses CPU other processes has to wait.
5.	It is highly efficient.	It is less efficient in comparison with multithreading.
6.	It helps in developing efficient application programs.	It helps in developing efficient operating system programs.

Review Question

1. What is thread ? Explain multithreading and multitasking in detail.

AU : CSE : Dec.-19, Marks 13

6.3 Thread Life Cycle

AU : CSE : Dec.-10, 11,18, May-12, 13, Marks 10

Thread life cycle specifies how a thread gets processed in the Java program. By executing various methods. Following figure represents how a particular thread can be in one of the state at any time.

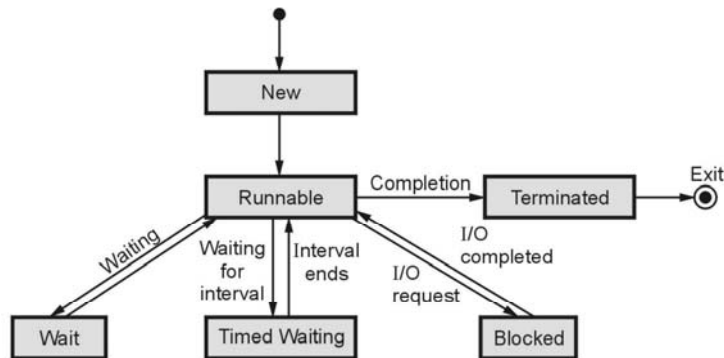


Fig. 6.3.1 Life cycle of thread

New state

- When a thread starts its life cycle it enters in the new state or a create state.

Runnable state

- This is a state in which a thread starts executing.
- Waiting state
- Sometimes one thread has to undergo in waiting state because another thread starts executing.

Timed waiting state

- There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.

Blocked state

- When a particular thread issues an Input/Output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state.

Terminated state

- After successful completion of the execution the thread in runnable state enters the terminated state.

Review Questions

1. What is thread ? Explain its state and methods.

AU : CSE : Dec.-10, Marks 8

2. Write about various threads states in Java.

AU : CSE : Dec.-11, Marks 8

3. Explain : States of a thread with a neat diagram.

AU : CSE : May-12, Marks 10

4. Define threads. Describe in detail about thread life cycle.

AU : CSE : May-13, Marks 8

5. Explain in detail the different states of a thread.

AU : Dec.-18, Marks 13

6.4 Creating Threads

AU : IT : Dec.-10, CSE : Dec.-11, 13, May-13, Marks 16

- In Java we can implement the thread programs using two approaches -
 - Using **Thread** class
 - sing **Runnable** interface.

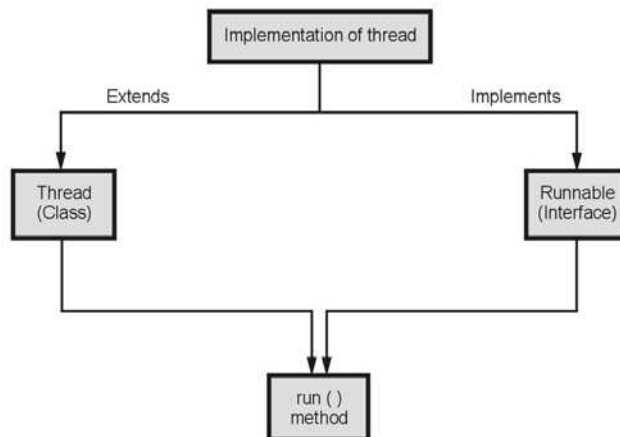


Fig. 6.4.1 : Creation of Java thread

As given in Fig. 6.4.1, there are two methods by which we can write the Java thread programs one is by extending **thread class** and the other is by implementing the **Runnable interface**.

- The **run()** method is the most important method in any threading program. By using this method the thread's behaviour can be implemented. The run method can be written as follows -

```
public void run()
{
    //statements for implementing thread
}
```

For invoking the thread's run method the object of a thread is required. This object can be obtained by creating and initiating a thread using the **start()** method.

6.4.1 Extending Thread Class

- The **Thread** class can be used to create a thread.
- Using the **extend** keyword your class extends the **Thread** class for creation of thread. For example if I have a class named **A** then it can be written as
class A extends Thread
- **Constructor of Thread Class** : Following are various syntaxes used for writing the constructor of Thread Class.
Thread()
Thread(String s)
Thread(Runnable obj)
Thread(Runnable obj, String s);
- Various commonly used methods during thread programming are as given below -

Method	Purpose
start()	The thread can be started and invokes the run method.
run()	Once thread is started it executes in run method.
setName()	We can give the name to a thread using this method.
getName()	The name of the thread can be obtained using this name.
join()	This method waits for thread to end

Following program shows how to create a single thread by extending Thread Class.

Java Program

```
class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread is created!!!");
    }
}
class ThreadProg
{
    public static void main(String args[])
    {
        MyThread t=new MyThread();
        t.start();
    }
}
```

Output

```
Command Prompt

D:\>javac ThreadProg.java

D:\>java ThreadProg
Thread is created!!!

D:\>
```

Program Explanation :

In above program, we have created two classes. One class named **MyThread** extends the Thread class. In this class the run method is defined. This run method is called by **t.start()** in main() method of class **ThreadProg**

The thread gets created and executes by displaying the message **Thread** is created.

Ex. 6.4.1 : Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol. :

```
class MyThread extends Thread
{
    String str=""; //data member of class MyThread
    MyThread(String s)//constructor
    {
        this.str=s;
    }
    public void run()
    {
        System.out.println(str);
    }
}
class ThreadProg
{
    public static void main(String args[])
    {
        MyThread t=new MyThread("You are Welcome to Thread Programming");
        t.start();
    }
}
```

```
}  
}
```

Output

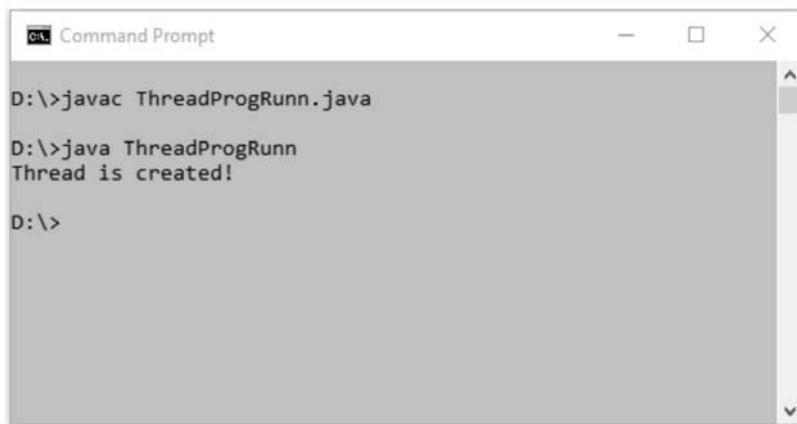
You are Welcome to Thread Programming

6.4.2 Implementing Runnable Interface

- The thread can also be created using **Runnable** interface.
- Implementing thread program using **Runnable interface** is **preferable** than implementing it by extending the **thread** class because of the following two reasons -
 1. If a class extends a **thread class** then it can not extends any other class which may be required to extend.
 2. If a class **thread** is extended then all its functionalities get inherited. This is an expensive operation.
- Following Java program shows how to implement **Runnable interface** for creating a single thread.

Java Program

```
class MyThread implements Runnable  
{  
    public void run()  
    {  
        System.out.println("Thread is created!");  
    }  
}  
class ThreadProgRunn  
{  
    public static void main(String args[])  
    {  
        MyThread obj=new MyThread();  
        Thread t=new Thread(obj);  
        t.start();  
    }  
}
```

Output

```
Command Prompt

D:\>javac ThreadProgRunn.java

D:\>java ThreadProgRunn
Thread is created!

D:\>
```

Program Explanation :

- In above program, we have used interface Runnable.
- While using the interface, it is necessary to use **implements** keyword.
- Inside the main method
 1. Create the instance of class **MyClass**.
 2. This instance is passed as a parameter to **Thread** class.
 3. Using the instance of class **Thread** invoke the **start** method.
 4. The start method in-turn calls the **run** method written in **MyClass**.
- The **run** method executes and display the message for thread creation.

Ex. 6.4.2 : Create a thread by extending the Thread Class. Also make use of constructor and display message "You are Welcome to Thread Programming".

Sol. :

```
class MyThread implements Runnable
{
    String str;
    MyThread(String s)
    {
        this.str=s;
    }
    public void run()
    {
        System.out.println(str);
    }
}
```

```
class ThreadProgRunn
{
    public static void main(String args[])
    {
        MyThread obj=new MyThread("You are Welcome to Thread Programming");
        Thread t=new Thread(obj);
        t.start();
    }
}
```

Output

You are Welcome to Thread Programming

Ex. 6.4.3 : Write a Java program that prints the numbers from 1 to 10 line by line after every 10 seconds.

AU : IT : Dec.-10

Sol. :

Java Program[MultiThNum.java]

```
class NumPrint extends Thread
{
    int num;
    NumPrint()
    {
        start();//directs to the run method
    }
    public void run();//thread execution starts
    {
        try
        {

            for(int i=1;i<=10;i++)
            {
                System.out.println(i);
                Thread.sleep(10000);//10 sec,b'coz 1000millisec=1 sec.
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Exception in Thread handling");
        }
    }
}

public class MultiThNum
{
    public static void main(String args[])
```

```
{
NumPrint t1;
t1=new NumPrint();

}
}
```

Output

C:>javac MultiThNum.java

C:>java MultiThNum

1
2
3
4
5
6
7
8
9
10

Review Questions

1. Explain how threads are created in Java.

AU : CSE : Dec.-11, Marks 8

2. How to extends the thread class ? Give an example.

AU : CSE : May-13, Marks 8

3. How to implement runnable interface for creating and starting threads ?

AU : CSE : May-13, Marks 8

4. What is meant by concurrent programming ? Define thread. Discuss the two ways of implementing thread using example.

AU : CSE : Dec.-13, Marks 16

6.5 Creating Multiple Threads

AU : IT : Dec.-11, May-12, CSE : May-12, 19, Dec.-11,12,18, 19, Marks 16

The multiple threads can be created both by extending Thread class and by implementing the Runnable interface.

1. Java Program for creating multiple threads by extending Thread Class

class A extends Thread

```
{
    public void run()
    {
        for(int i=0;i<=5;i++)//printing 0 to 5
        {
            System.out.println(i);
        }
    }
}
```

```
    }  
}  
class B extends Thread  
{  
    public void run()  
    {  
        for(int i=10;i>=5;i--)//printing 10 to 5  
        {  
            System.out.println(i);  
        }  
    }  
}  
class ThreadProg  
{  
    public static void main(String args[])  
    {  
        A t1=new A();  
        B t2=new B();  
        t1.start();  
        t2.start();  
    }  
}
```

Output

```
0  
1  
2  
3  
4  
5  
10  
9  
8  
7  
6  
5
```

2. Java Program for creating multiple threads by implementing the Runnable interface

class A implements Runnable

```
{  
    public void run()  
    {  
        for(int i=0;i<=5;i++)//printing 0 to 5  
        {  
            System.out.println(i);  
        }  
    }  
}
```

```
    }  
  }  
}  
class B implements Runnable  
{  
    public void run()  
    {  
        for(int i=10;i>=5;i--)//printing 10 to 5  
        {  
            System.out.println(i);  
        }  
    }  
}  
class ThreadProgRunn  
{  
    public static void main(String args[])  
    {  
        A obj1=new A();  
        B obj2=new B();  
        Thread t1=new Thread(obj1);  
        Thread t2=new Thread(obj2);  
        t1.start();  
        t2.start();  
    }  
}
```

Output

```
0  
1  
2  
3  
4  
5  
10  
9  
8  
7  
6  
5
```

Ex. 6.5.1 : Write a Java application program for generating four threads to perform the following operations - i) Getting N numbers as input ii) Printing the numbers divisible by five iii) Printing prime numbers iv) Computing the average.

Sol. :

```
import java.io.*;
import java.util.*;
class FirstThread extends Thread
{
    public void run() //generating N numbers
    {

        int i;
        System.out.println("\nGenerating Numbers: ");
        for (i=1;i<=10;i++)
        {
            System.out.println(i);
        }
    }
}

class SecondThread extends Thread
{
    public void run() //Displaying the numbers divisible by five
    {
        int i;
        System.out.println("\nDivisible by Five: ");
        for (i=1;i<=10;i++) //10 can be replaced by any desired value
        {
            if (i%5==0)
                System.out.println(i);

        }

    }
}

class ThirdThread extends Thread
{
    public void run() //generating the prime numbers
    {
        int i;
        System.out.println("\nPrime Numbers: ");
        for (i=1;i<=10;i++) //10 can be replaced by any desired value
        {
            int j;
            for (j=2; j<i; j++)
            {
                int n = i%j;
```

```
        if (n==0)
            break;
    }
    if(i == j)
        System.out.println(i);

}

}
}

class FourthThread extends Thread
{

    public void run() //generating the prime numbers
    {
        int i,sum;
        double avg;
        sum=0;
        System.out.println("\nComputing Average: ");
        for (i=1;i<=10;i++) //10 can be replaced by any desired value
            sum=sum+i;
        avg=sum/(i-1);
        System.out.println(avg);
    }
}

class MainThread
{
    public static void main(String[] args) throws IOException
    {
        FirstThread T1 = new FirstThread(); //creating first thread
        SecondThread T2 = new SecondThread(); //creating second thread
        ThirdThread T3 = new ThirdThread(); //creating Third thread
        FourthThread T4 = new FourthThread(); //creating Fourth thread
        T1.start(); //First Thread starts executing
        T2.start(); //Second Thread starts executing
        T3.start(); //Third Thread starts executing
        T4.start(); //Fourth Thread starts executing
    }
}
```

Output

Generating Numbers:

1
2
3

4

5

6

7

8

9

10

Divisible by Five:

5

10

Computing Average:

5.0

Prime Numbers:

2

3

5

7

Ex. 6.5.2 : Create a Bank Database application program to illustrate the use of multithreads

AU : Dec.-18, Marks 15**Sol. :**

```
public class BankAppl implements Runnable {
    private Account acc = new Account();
    public static void main(String[] args) {
        BankAppl obj = new BankAppl();
        Thread t1 = new Thread(obj);
        Thread t2 = new Thread(obj);
        Thread t3 = new Thread(obj);
        t1.setName("Mr.XYZ");
        t2.setName("Mr.ABC");
        t3.setName("Mr.PQR");
        t1.start();
        t2.start();
        t3.start();
    }
    public void run() {
        for (int x = 0; x < 10; x++) {
            makeWithdrawal(10);
            if (acc.getBalance() < 0) {
                System.out.println("Account Overdrawn!!!");
            }
        }
    }
    void makeWithdrawal(int amt) {
```

```

        int bal;
        bal=acc.getBalance();
        if (bal >= amt) {
            System.out.println("\t"+Thread.currentThread().getName() + "
withdraws Rs."+amt);
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
            }
            acc.withdraw(amt);
            bal=acc.getBalance();
            System.out.println("\t The Balance: "+bal);

        } else {
            System.out.println("Insufficient Balance in account for " +
Thread.currentThread().getName() + " to withdraw " + acc.getBalance());
        }
    }
}
class Account {
    private int balance = 100;
    public int getBalance() {
        return balance;
    }
    public void withdraw(int amount) {
        balance = balance - amount;
    }
}

```

Review Questions

1. What is multithreading ? What are the methods available in Java for inter-thread communication ? Discuss with example. **AU : IT : Dec.-11, Marks 10**
2. Write a Java program to illustrate multithreaded programming. **AU : CSE : Dec.-11,12, Marks 16**
3. Write notes on multi-threaded programming. **AU : IT : May-12, Marks 16**
4. Write a java application that illustrate the use of multithreading. Explain the same with sample input. **AU : CSE : May-12, Marks 16**
5. Describe the creation of a single thread and multiple threads using an example. **AU : CSE : May-19, Marks 13**
6. Create a simple real life application program in Java to illustrate the use of multithreads. **AU : CSE : Dec.-19, Marks 15**

6.6 Synchronizing Threads

AU : Dec.-14, 19, Marks 13

- When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring

one access at a time by one thread is called synchronization. The synchronization is the concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When a thread owns this monitor at a time then the other threads can not access the resources. Other threads have to be there in waiting state.

- In Java every object has implicit monitor associated with it. For entering in object's monitor, the method is associated with a keyword synchronized. When a particular method is in synchronized state then all other threads have to be there in waiting state.
- There are two ways to achieve the synchronization -
 1. Using Synchronized Methods
 2. Using Synchronized Blocks (Statements).

Let us make the method synchronized to achieve the synchronization by using following Java program -

1. Using Synchronized Method

```
class Test
```

```
{
    synchronized void display(int num)
    {
        System.out.println("\nTable for "+num);
        for(int i=1;i<=10;i++)
        {
            System.out.print(" "+num*i);
        }
        System.out.print("\nEnd of Table");
        try
        {
            Thread.sleep(1000);
        }catch(Exception e){}
    }
}
```



This is a synchronized method

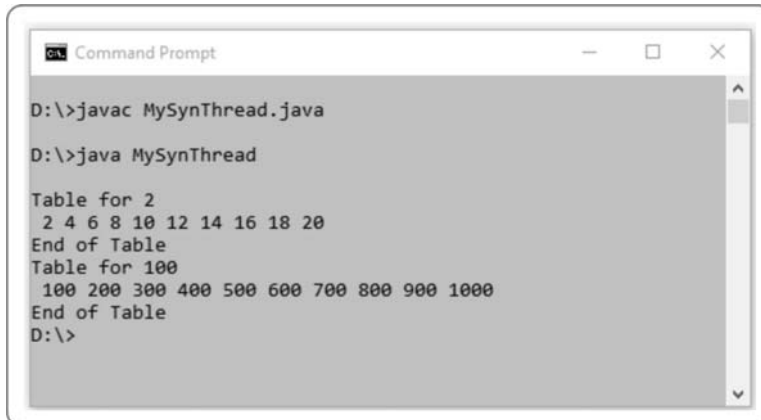
```
class A extends Thread
```

```
{
    Test th1;
    A(Test t)
    {
        th1=t;
    }
    public void run()
    {
        th1.display(2);
    }
}
```

```
class B extends Thread
```

```
{
    Test th2;
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}
class MySynThread
{
    public static void main(String args[])
    {
        Test obj=new Test();
        A t1=new A(obj);
        B t2=new B(obj);
        t1.start();
        t2.start();
    }
}
```

Output



```
Command Prompt

D:\>javac MySynThread.java

D:\>java MySynThread

Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

Program Explanation :

- In above program we have written one class named **Test**. Inside this class the synchronized method named **display** is written. This method displays the table of numbers.
- We have written two more classes named **A** and **B** for executing the thread. The constructors for class **A** and Class **B** are written and to initialize the instance variables of class **Test** as **th1** (as a variable for class **A**) and **th2** (as a variable for class **B**)
- Inside the **run** methods of these classes we have passed number 2 and 10 respectively and **display** method is invoked.

- The display method executes firstly for thread t1 and then for t2 in synchronized manner.

2. Using Synchronized Block

- When we want to achieve synchronization using the synchronized block then create a block of code and mark it as synchronized.
- Synchronized statements must specify the object that provides the intrinsic lock.

Syntax

The syntax for using the synchronized block is -

```
synchronized(object reference )  
{  
statement;  
statement; //block of code to be synchronized  
.  
.  
.  
}
```

Java Program

```
class Test  
{  
    void display(int num)  
    {  
        synchronized(this)  
        {  
            System.out.println("\nTable for "+num);  
            for(int i=1;i<=10;i++)  
            {  
                System.out.print(" "+num*i);  
            }  
            System.out.print("\nEnd of Table");  
            try  
            {  
                Thread.sleep(1000);  
            }catch(Exception e){}  
        }  
    }  
}  
}  
class A extends Thread  
{  
    Test th1;  
    A(Test t)
```

This is a synchronized Block

```
{
    th1=t;
}
public void run()
{
    th1.display(2);
}
}
class B extends Thread
{
    Test th2;
    B(Test t)
    {
        th2=t;
    }
    public void run()
    {
        th2.display(100);
    }
}
class MySynThreadBlock
{
    public static void main(String args[])
    {
        Test obj=new Test();
        A t1=new A(obj);
        B t2=new B(obj);
        t1.start();
        t2.start();
    }
}
```

Output



```
Command Prompt

D:\>javac MySynThreadBlock.java

D:\>java MySynThreadBlock

Table for 2
 2 4 6 8 10 12 14 16 18 20
End of Table
Table for 100
100 200 300 400 500 600 700 800 900 1000
End of Table
D:\>
```

Points to Remember about Synchronization

1. **Only methods** can be synchronized but the variables and classes can not be synchronized.
2. Each **object** has **one lock**.
3. A class contains several methods and all methods need not be synchronized.
4. If two threads in a class wants to execute synchronized methods and both the methods are using the same instance of a class then only one thread can access the synchronized method at a time.
5. We **can not synchronize the constructors**.
6. A thread can acquire more than one lock.

Review Questions

1. What is thread synchronization ? Discuss with an example.
2. What is t synchronization ? Explain the different types of synchronization in java.

AU : Dec.-14, Marks 8

AU : Dec.-19, Marks 13

6.7 Inter-thread Communication

AU : IT, May-13, 19, Dec.-18, Marks 16

- Two or more threads communicate with each other by exchanging the messages. This mechanism is called **interthread communication**.
- Polling is a mechanism generally implemented in a loop in which certain condition is repeatedly checked.
- To better understand the concept of polling, consider producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced.
- Both must work in co-ordination to avoid wastage of CPU cycles.
- But there are situations in which the producer has to wait for the consumer to finish consuming of data. Similarly the consumer may need to wait for the producer to produce the data.
- In Polling system either consumer will waste many CPU cycles when waiting for producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data.
- In order to avoid polling **there are three in-built methods** that take part in inter-thread communication -

notify()	If a particular thread is in the sleep mode then that thread can be resumed using the notify call.
----------	--

notifyall()	This method resumes all the threads that are in suspended state.
wait()	The calling thread can be send into a sleep mode.

- **Example Program**

Following is a simple Java program in which two threads are created one for producer and another is for consumer.

The producer thread **produces(writes)** the numbers from 0 to 9 and the consumer thread consumes(reads) these numbers.

The **wait** and **notify** methods are used to send particular thread to sleep or to resume the thread from sleep mode respectively.

Java Program[InterThread.java]

```
class MyClass
{
    int val;
    boolean flag = false;
    synchronized int get() //by toggling the flag synchronised read and write is performed
    {
        if(!flag)
            try
            {
                wait();
            }
            catch(InterruptedException e)
            {
                System.out.println("InterruptedException!!!");
            }
        System.out.println("Consumer consuming: " + val);
        flag = false;
        notify();
        return val;
    }
    synchronized void put(int val)
    {
        if(flag)
            try
            {
                wait();
            }
            catch(InterruptedException e)
            {
                System.out.println("InterruptedException!!!");
            }
    }
```

```
    this.val = val;
    flag = true;
    System.out.println("Producer producing " + val);
    notify();
}
}
class Producer extends Thread
{
    MyClass th1;
    Producer(MyClass t)
    {
        th1 = t;
    }
    public void run()
    {
        for(int i = 0;i<10;i++)
        {
            th1.put(i);
        }
    }
}
class Consumer extends Thread
{
    MyClass th2;
    Consumer(MyClass t)
    {
        th2 = t;
    }
    public void run()
    {
        for(int i = 0;i<10;i++)
        {
            th2.get();
        }
    }
}
class InterThread
{
    public static void main(String[] arg)
    {
        MyClass TObj = new MyClass();
        Producer pthread=new Producer(TObj);
        Consumer cthread=new Consumer(TObj);
        pthread.start();
        cthread.start();
    }
}
```

Product thread is writing the numbers from 0 to 9

Consumer thread is reading the numbers from 0 to 9

```
}  
}
```

Output

```
Producer producing 0  
Consumer consuming: 0  
Producer producing 1  
Consumer consuming: 1  
Producer producing 2  
Consumer consuming: 2  
Producer producing 3  
Consumer consuming: 3  
Producer producing 4  
Consumer consuming: 4  
Producer producing 5  
Consumer consuming: 5  
Producer producing 6  
Consumer consuming: 6  
Producer producing 7  
Consumer consuming: 7  
Producer producing 8  
Consumer consuming: 8  
Producer producing 9  
Consumer consuming: 9
```

Program Explanation

- **Inside get() -**

The **wait()** is called in order to suspend the execution by that time the producer writes the value and when the data gets ready it notifies other thread that the data is now ready.

Similarly when the consumer reads the data execution inside **get()** is suspended. After the data has been obtained, **get()** calls **notify()**. This tells producer that now the producer can write the next data in the queue.

- **Inside put() -**

The **wait()** suspends execution by that time the consumer removes the item from the queue. When execution resumes, the next item of data is put in the queue, and **notify()** is called. When the notify is issued the consumer can now remove the corresponding item for reading.

6.7.1 Producer Consumer Pattern

- The producer consumer pattern is a kind of pattern in which **producer thread** produces the data and **consumer thread** consumes it.

- The data is stored in the block of memory **shared** between producer and consumer thread.
- If the **producer thread** is producing the data at must **faster rate** then the consumer that consumes it then producer thread may **overwrite the previous result** before consumer thread receives it.
- If **consumer thread** receives data at much faster rate than producer deposits the data, then consumer can retrieve the **identical data or may attempt to retrieve data that is not been deposited yet**.
- The working of producer and consumer must be **synchronized**.
- For the implementation of producer-consumer pattern in Java, refer program in section 6.7.

Ex. 6.7.1 : Write a java program for inventory problem to illustrates the usage of thread synchronized keyword and inter thread communication process. They have three classes called consumer, producer and stock.

AU : IT, May-13, Marks 16



Sol. :

class Consumer implements Runnable

```

{
    Stock obj1;
    Thread t;
    Consumer (Stock obj1)
    {
        this.obj1 = obj1;
        t = new Thread(this,"Consumer thread");
        t.start();
    }
    public void run()
    {
        while (true)
        {
            try {
                t.sleep(900);
            } catch (InterruptedException e) { }
            obj1.getStock((int)(Math.random()*100));
        }
    }
    void stop()
    {
        t.stop();
    }
}

```

```
    }  
}  
class Producer implements Runnable  
{  
    Stock obj2;  
    Thread t;  
    Producer(Stock obj2)  
    {  
        this.obj2 = obj2;  
        t = new Thread(this, "Producer thread");  
        t.start();  
    }  
    public void run()  
    {  
        while(true)  
        {  
            try  
            {  
                t.sleep(900);  
            } catch (InterruptedException e) { }  
            obj2.addStock((int)(Math.random()*100));  
        }  
    }  
    void stop()  
    {  
        t.stop();  
    }  
}  
class Stock  
{  
    int items = 0;  
    public synchronized void addStock(int i)  
    {  
        items = items + i;  
        System.out.println("Stock added :"+ i);  
        System.out.println("present stock :"+ items );  
        notify();  
    }  
    public synchronized int getStock(int j)  
    {  
        while(true)  
        {  
            if(items >= j)  
            {
```

```
        items = items - j ;
        System.out.println("Item is removed from stock : " + j);
        System.out.println("Current stock is : " + items);
        break;
    }
    else
    {
        System.out.println("\tStock is not enough!!!" );
        System.out.println ( "\t Waiting for items to get added...");
        try {
            wait();
        } catch (InterruptedException e) { }
    }
}
return items;
}
public static void main(String args[])
{
    Stock j = new Stock();
    Producer p = new Producer(j);
    Consumer c = new Consumer(j);
    try
    {
        Thread.sleep(10000);
        p.stop();
        c.stop();
        p.t.join();
        c.t.join();
        System.out.println("Thread stopped");
    } catch (InterruptedException e) { }
    System.exit(0);
}
}
```

Review Questions

1. Demonstrate inter thread communication and suspending, resuming and stopping threads.
2. Using an example, explain inter-thread communication in Java.

AU : Dec.-18, Marks 13

AU : May-19, Marks 7

6.8 Daemon Threads

- Daemon thread is a low priority thread which runs in the back ground.
- For example the thread doing the garbage collection operation for the java runtime system is a daemon thread.
- **setDaemon** method is used to create a daemon thread.
- The daemon threads are also called **service provider** threads.
- The **run()** method is an infinite loop for the daemon thread in which it waits for servicing.

Java program[My Daemon.java]

```
/******  
Program to create and execute a daemon thread  
*****/  
class MyDaemon implements Runnable  
{  
    Thread t;  
    MyDaemon()  
    {  
        t = new Thread(this);  
        t.setDaemon(true);  
        t.start();  
    }  
    public void run()  
    {  
        try  
        {  
            while(true)  
            {  
                System.out.print("\n*");  
                Thread.sleep(100);  
            }  
        }  
        catch(Exception e)  
        {  
            System.out.println("MyDaemon interrupted.");  
        }  
    }  
}  
  
public class DaemonDemo  
{  
    public static void main(String args[]) throws Exception  
    {  
        MyDaemon t = new MyDaemon();  
        System.out.println("Daemon thread started.");  
        Thread.sleep(1000);  
        System.out.println("\nMain thread ending.");  
    }  
}
```

Output

Daemon thread started.

*

*
*
*
*
*
*
*
*
*
*
*

Main thread ending.

6.9 Thread Groups

- **Definition :** Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once.
- For example, you can start or suspend all the threads within a group with a single method call.
- The thread group can be implemented using **ThreadGroup** class. This class belongs to java.lang package.
- There are two **constructors** used for creating thread group :

ThreadGroup(String name)	Creates a thread group by some name.
ThreadGroup(ThreadGroup parent, String name)	Creates a thread group with given parent and for given name.

Methods used in ThreadGroup

Method	Description
int activeCount()	It returns number of active threads in the current thread group.
int activeGroupCount()	It returns number of active thread groups.
void checkAccess()	It checks whether the currently running thread is allowed to modify the current thread group or not.
String getName()	This method is used to obtain name of the working thread group.
ThreadGroup getParent()	This method is used to obtain name of the parent of working thread group.

void interrupt()	This method is used to interrupt all the threads in the group.
void list	This method is used to display the information about the thread group.

Example Program

public class ThreadGroupProg implements Runnable

```
{
    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main(String[] args)
    {
        ThreadGroupProg runnable = new ThreadGroupProg();
        ThreadGroup tg1 = new ThreadGroup("Colorful ThreadGroup");
        System.out.println("Thread Group Name: "+tg1.getName());
        Thread t1 = new Thread(tg1, runnable,"Red");
        t1.start();
        Thread t2 = new Thread(tg1, runnable,"Green");
        t2.start();
        Thread t3 = new Thread(tg1, runnable,"Blue");
        t3.start();
        Thread t4 = new Thread(tg1, runnable,"Yellow");
        t4.start();
        System.out.println("The active threads are: "+tg1.activeCount());
    }
}
```

Output

Thread Group Name: Colorful ThreadGroup

Red

Blue

Green

Yellow

The active threads are: 4

Two Marks Questions with Answers

Q.1 What do you mean by threads in Java ?

AU : CSE : Dec.-11; IT : May-12, Dec.-13

Ans. : Thread is tiny program running continuously. It is a light weight process in Java.

Q.2 Give the difference between process and thread.

AU : IT : Dec.-11

Ans. : Refer section 6.1.

Q.3 What are different stages in thread ?

AU : CSE : Dec.-10, 19

Ans. : Various stages in thread are -

- | | | |
|-----------------------|-------------------|---------------------|
| 1. New state | 2. Runnable state | 3. Waiting state |
| 4. Time waiting state | 5. Blocked state | 6. Terminated state |

Q.4 What do you mean by synchronization ?

AU : CSE : Dec.-10, May-12

Ans. : When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization.

Q.5 What are the three ways by which the thread can enter in waiting stage ?

AU : IT : Dec.-10

Ans. :

- i) **Waiting state :** Sometimes one thread has to undergo in waiting state because another thread starts executing. This can be achieved by using wait() state.
- ii) **Timed waiting state :** There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state. This can be achieved by using the sleep() command.
- iii) **Blocked state :** When a particular thread issues an input/output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state. This can be achieved by using suspend() method.

Q.6 What is multithreading ?

AU : CSE : Dec.-11, May-12; IT : Dec.-12, 18

Ans. : Multithreading is an environment in which multiple threads are created and they can execute simultaneously. The multiple threads can be created either by extending the thread class or by implementing the runnable interface.

Q.7 Mention two mechanisms for protecting a code block from concurrent access.

Ans. : There are two mechanisms for protecting a code block from concurrent access -

- | | |
|--------------------------|---------------------------|
| 1. Use of reentrant code | 2. Use synchronized block |
|--------------------------|---------------------------|

Q.8 What is meant by notify methods in multithreading ?

Ans. : The notify() method is used for inter thread communication. If a particular thread is in the sleep mode then that thread can be resumed by using the notify call.

Q.9 What are the two ways of creating a thread ?**AU : IT : Dec.-12**

Ans. : Thread can be created using

1. Thread class
2. runnable interface.

Q.10 Why do we need run() and start() method both ? Can we achieve it with only run() method ?**AU : IT : May-13**

Ans. : We use start() method to start a thread instead of calling run() method because the run() method is not a regular class method. It should only be called by the JVM. If we call the run() method directly then it will simply invoke the caller's thread instead of its own thread. Hence the start() method is called which schedules the thread with the JVM.

Q.11 What is the need for thread ?**AU : CSE : May-13**

Ans. : In Java threads are used to handle multiple tasks together. This kind of programming is called concurrent programming.

Q.12 Name any four thread constructor.**AU : CSE : May-13**

Ans. :

1. Thread()
2. Thread(String name)
3. Thread(Runnable target)
4. Thread(Runnable target, String name)

Q.13 When thread is initiated and created, what is its initial stage ?**AU : May-15**

Ans. : A thread is in "Ready" state after it has been created and started. This state signifies that the thread is ready for execution. From here, it can be in the running state.

Q.14 "Thread is a lightweight process" – Comment**AU : Dec.-19**

Ans. : Threads do not require separate address for its execution. It runs in the address space of the process to which it belongs to. Hence thread is a lightweight process.

Q.15 Sketch the life cycle of thread**AU : May-19**

Ans. : Refer Fig. 6.3.1



UNIT-IV

7

Generic Programming

Syllabus

Generic Programming - Generic classes - generic methods - Bounded Types - Restrictions and Limitations

Contents

7.1	What is Generic Programming ?	Dec.-13,	Marks 4
7.2	Generic Methods.....	May-15, 19	Marks 8
7.3	Generic Classes.....	May-14, CSE : Dec.-10,12,14, May-12, IT : May-13,	Marks 16
7.4	Bounded Types		
7.5	Restrictions and Limitations		

7.1 What is Generic Programming ?

AU : Dec.-13, Marks 4

Generic is a mechanism for creating a general model in which generic methods and generic classes enable programmers to specify a single method (or a set of related methods) and single class (or a set of related classes) for performing the desired task.

For example -

Suppose we want to create a stack and if we create a stack of integers then it will store only the integer elements, if we try to push any string or any double type element then a compile time error will occur. If we want to push any string then we need to create a separate stack, separate class and separate methods for handling the String elements. Same is true for the elements of any other data type. This results in complex code building. To avoid such complexity, a concept of generic programming is introduced.

What is the need for Generic?

Following features show the importance of generics in Java.

1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
2. It allows the code reusability.
3. Compact code can be created.

Review Question

1. *Mention the motivations of generic programming*

AU : Dec.-13, Marks 4

7.2 Generic Methods

AU : May-15, 19, Marks 8

Generic method allows a programmer to write a generalised method for the methods of different data types.

Suppose we want to print an array of integer, float and character type elements then normally we write three different methods performing the corresponding task. The object oriented feature of Java allows us to make use of same function name. This idea is illustrated in following Java program –

Java Program[OverloadProg1.java]

```
import java.io.*;
import java.util.*;
public class OverloadProg1
{
```

```

public static void display(float[] a)
{
    for(int i=0;i<5;i++)
        System.out.printf(" %.2f",a[i]);
}

public static void display(int[] a)
{
    for(int i=0;i<5;i++)
        System.out.printf(" %d",a[i]);
}

public static void display(char[] a)
{
    for(int i=0;i<5;i++)
        System.out.printf(" %c",a[i]);
}

public static void main(String[] args)
{
    float[] dbl_a={11,22,33,44,55};
    int[] int_a={1,2,3,4,5};
    char[] char_a={'A','B','C','D','E'};
    System.out.println("\n The Float elements are ...");
    display(dbl_a);
    System.out.println("\n The Integer elements are ...");
    display(int_a);
    System.out.println("\n The character elements are ...");
    display(char_a);
}
}

```

Note that the name of these three functions is same i.e. **display**

Output

```

The Float elements are ...
11.00 22.00 33.00 44.00 55.00
The Integer elements are ...
1 2 3 4 5
The character elements are ...
A B C D E

```

But if we override the generic methods then the code becomes more simplistic –

Java Program[OverloadProg2.java]

```

import java.io.*;
import java.util.*;
public class OverloadProg2
{
    public static < T > void display(T[] a)           //Generic Method is created
    {
        for(int i=0;i<5;i++)

```

```

        System.out.printf(" %s",a[i]);
    }
    public static void main(String[] args)
    {
        float[] dbl_a={11,22,33,44,55};
        int[] int_a={1,2,3,4,5};
        char[] char_a={'A','B','C','D','E'};
        System.out.println("\n The Float elements are ...");
        display( dbl_a );
        System.out.println("\n The Integer elements are ...");
        display( int_a );
        System.out.println("\n The character elements are ...");
        display( char_a );
    }
}

```

Output

```

The Float elements are ...
11.00 22.00 33.00 44.00 55.00
The Integer elements are ...
1 2 3 4 5
The character elements are ...
A B C D E

```

Ex. 7.2.1 : Write generic method for sorting an array of integer objects.

AU : May-19, Marks 6

Sol. :

```

private <E extends Comparable<E>> void bubbleSortG(E[] Arr) {
    E temp;
    for(int j = 1; j < Arr.length; j++) {
        for(int i = 0; i < Arr.length - j; i++) {
            if(Arr[i].compareTo(Arr[i+1]) > 0) {
                temp = Arr[i];
                Arr[i] = Arr[i+1];
                Arr[i+1] = temp;
            }
        }
    }
    for(E i: Arr) {
        System.out.print(" " + i);
    }
}

```

Review Question

1. Explain about generic method with suitable example.

AU : May-15, Marks 8

7.3 Generic Classes

AU : May-14, CSE : Dec.-10,12,14, May-12, IT : May-13, Marks 16

A generic class contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class .

```
public class Test<T>
{
    public Test(){val=null;}
    public Test(T val)
    {
        this.val=val;
    }
    public getVal()
    {
        return val;
    }
    public setVal()
    {
        val=newValue;
    }
    private T val; //variable defined as of generic type
}
```

The concept of generic class supports the idea of type-independency. Typical example of data structure is stack. We can create a generic class for stack which allows us to insert integer, character or any other data type elements using the same **push** and **pop** methods.

Ex. 7.3.1 : Create a generic class for the stack data structure. Your class must handle integer and character type elements. Show clearly how will you handle the stack empty condition

Sol. : We will create a generic class for stack data structure using following steps -

Step 1 : Create a Java file named Stack.java as follows -

Java Program[Stack.java]

```
import java.util.*; //Supports the ArrayList
public class Stack<T> //T denotes any data type
{
    public ArrayList<T> obj;
    public Stack(int size) //Constructor will be invoked from main
    {
        obj=new ArrayList<T>(size);
    }
    public void push(T item) //Generic method for PUSH operation
    {
        obj.add(item);
    }
}
```

```

public T pop()//Generic method for POP operation
{
    if(obj.isEmpty())
    {
        System.out.println("\n Stack is Empty");
        return null;
    }
    return obj.remove(obj.size()-1);
}
}

```

Step 2 : Create another Java program in a separate file named **StackGeneric.java**. It is as given below -

Java Program[StackGeneric.java]

```

import java.io.*;
import java.util.*;
public class StackGeneric
{

```

```

    public static void main(String[] args)
    {
        int[] iArray={1,2,3,4,5};
        char[] cArray={'A','B','C','D','E'};

```

Declaring integer and character values to be pushed onto the stack.

ist is an instance for integer stack and **cst** is an instance for character stack.
The Constructor **Stack(size)** will be invoked.

```

Stack<Integer> ist=new Stack<Integer>(5);
Stack<Character> cst=new Stack<Character>(5);

```

```

System.out.println("\n Pushing the elements in integer stack");
for(int i=0;i<5;i++)
    ist.push(iArray[i]);

```

Pushing onto the integer stack

```

System.out.println("\n Pushing the elements in character stack");
for(int i=0;i<5;i++)
    cst.push(cArray[i]);

```

Pushing onto the character stack

```

System.out.println("\n Popping two elements from character stack");
for(int i=0;i<2;i++)
    System.out.printf("\n%c",cst.pop());

```

Popping from the character stack

```

System.out.println("\n Popping all the elements from integer stack");
for(int i=0;i<5;i++)
    System.out.printf("\n%d",ist.pop());

```

Popping from the integer stack

```

        System.out.println("\n Performing one more pop for integer stack");
        System.out.printf("\n%d",ist.pop());
    }
}

```

Output

This pop operation
is to handle the
stack Empty
condition

```

F:\>javac StackGeneric.java
F:\>java StackGeneric

```

Pushing the elements in integer stack

Pushing the elements in character stack

Popping two elements from character stack

```

E
D
Popping all the elements from integer stack
5
4
3
2
1
Performing one more pop for integer stack

Stack is Empty

null

```

Program Explanation

In the step 1 we have created a separate Java file, in which a class is written for defining the generic methods push and pop. Note that it is necessary to define the constructor for this class because it will then allow to initialize the class of appropriate data type.

In step 2 we are first creating the separate instances for each of these classes say **cst** and **ist**. Then using these instances the generic push and pop methods will be invoked.

The output given in the step 2 is self explanatory.

Ex. 7.3.2 : Using generic classes, write a program to perform the following operations on an array i) Add an element in the beginning/middle/end ii) Delete an element from a given position

AU : May-14, Marks 16

Sol. :

```

import java.io.*;
import java.util.*; //Supports the ArrayList
class Arr<T> //T denotes any data type

```

```
{
    public ArrayList<T> obj;
    public Arr(int size) //Constructor will be invoked from main
    {
        obj=new ArrayList<T>(size);
    }
    public void insert(int index,T item) //Generic method for insert Operation
    {
        obj.add(index,item);
    }
    public void display()
    {
        System.out.print(" "+obj);
    }
    public T del(int index)//Generic method for delete Operation
    {
        return obj.remove(index);
    }
}

public class ArrayGeneric
{
    public static void main(String[] args)
    {
        int[] iArray={1,2,3,4,5};
        Arr<Integer> iobj=new Arr<Integer>(10);
        int i,index;
        System.out.println("\n Array of integers is ...");
        for(i=0;i<5;i++)
            iobj.insert(i,iArray[i]);
        iobj.display();
        System.out.println("\n Inserting the elements in integer Array");
        System.out.println("Enter the element to be inserted: ");
        Scanner sc=new Scanner(System.in);
        int item = sc.nextInt();
        System.out.println("Enter the index at which the element is to be inserted: ");
        index = sc.nextInt();
        iobj.insert(index,item);
        iobj.display();
        System.out.println("\n Enter the index of the element to be deleted: ");
        index = sc.nextInt();
        iobj.del(index);
        iobj.display();
        double[] dArray={11.11,22.22,33.33,44.44,55.55};
        Arr<Double> dobj=new Arr<Double>(10);
```

```
        System.out.println("\n Array of doubles is ...");
        for(i=0;i<5;i++)
            dobj.insert(i,dArray[i]);
        dobj.display();
        System.out.println("\n Inserting the elements in double Array");
        System.out.println("Enter the element to be inserted: ");
        sc=new Scanner(System.in);
        double ditem = sc.nextDouble();
        System.out.println("Enter the index at which the element is to be inserted: ");
        index = sc.nextInt();
        dobj.insert(index,ditem);
        dobj.display();
        System.out.println("\n Enter the index of the element to be deleted: ");
        index = sc.nextInt();
        dobj.del(index);
        dobj.display();
    }
}
```

Output

```
Array of integers is ...
[1, 2, 3, 4, 5]
Inserting the elements in integer Array
Enter the element to be inserted:
100
Enter the index at which the element is to be inserted:
2
[1, 2, 100, 3, 4, 5]
Enter the index of the element to be deleted:
4
[1, 2, 100, 3, 5]
Array of doubles is ...
[11.11, 22.22, 33.33, 44.44, 55.55]
Inserting the elements in double Array
Enter the element to be inserted:
111.222
Enter the index at which the element is to be inserted:
3
[11.11, 22.22, 33.33, 111.222, 44.44, 55.55]
Enter the index of the element to be deleted:
2
[11.11, 22.22, 111.222, 44.44, 55.55]
```

Review Questions

1. Explain the generic classes and generic methods with example.

AU : CSE : Dec.-10,12, May-12, IT : May-13, Marks 16

2. Explain in detail about generic classes and methods in java with suitable example.

AU : IT : May-13, Marks 16

3. Develop a Java program that will illustrate the use of Generic classes. Give self explanatory comments in your program

AU : Dec.-14, Marks 8

7.4 Bounded Types

- While creating objects to generic classes we can pass any derived type as type parameters.
- Many times it will be useful to limit the types that can be passed to type parameters. For that purpose, **bounded types** are introduced in generics.
- Using bounded types, we can make the objects of generic class to have data of specific derived types.
- For example, If we want a generic class that works only with numbers (like int, double, float, long) then declare type parameter of that class as a bounded type to **Number** class. Then while creating objects to that class you have to pass only Number types or it's subclass types as type parameters.
- The syntax for declaring Bounded type parameters is

`<T extends SuperClass>`

- This specifies that 'T' can only be replaced by 'SuperClass' or it's sub classes.
- **For example**

```
class Test<T extends Number>    //Declaring Number class as upper bound of T
{
    T t;
    public Test(T t)
    {
        this.t = t;
    }
    public T getT()
    {
        return t;
    }
}
public class BoundedTypeDemo
{
    public static void main(String[] args)
    {
        //Creating object by passing Number as a type parameter
```

```

    Test<Number> obj1 = new Test<Number>(123);
    System.out.println("The integer is: "+obj1.getT());
    //While Creating object by passing String as a type parameter, it gives compile time
    //error
    Test<String> obj2 = new Test<String>("I am string"); //Compile time error
    System.out.println("The string is: "+obj2.getT());
}
}

```

Review Question

1. Explain the use of bounded types in Generics with illustrative program.

7.5 Restrictions and Limitations

1. In Java, generic types are compile time entities. The runtime execution is possible only if it is used along with raw type.
2. Primitive type parameters is not allowed for generic programming.
For example: Stack<int> is not allowed.
3. For the instances of generic class throw and catch instances are not allowed.

For example :

```

    public class Test<T> extends Exception
    {
        //code    // Error:can't extend the Exception class
    }

```

4. Instantiation of generic parameter T is not allowed.

For example :

```

    new T() //Error
    new T[10]

```

5. Arrays of parameterized types are not allowed.

For example :

```

    new Stack<String>[10]; //Error

```

6. Static fields and static methods with type parameters are not allowed.

Two Marks Questions with Answers

Q.1 Why generic programming is required ?

OR

What is the need for generic programming ?

AU : CSE : May-13, Dec.-18

Ans. : Following some reasons for the need of generic programming -

1. It saves the programmers burden of creating separate methods for handling data belonging to different data types.

2. It allows the code reusability.
3. Compact code can be created.

Q.2 List out motivation needed in generic programming.

Ans. :

1. Suppose we want to create a stack and if we create a stack of integers then it will store only the integer elements, if we try to push any string or any double type element then a compile time error will occur. If we want to push any string then we need to create a separate stack, separate class and separate methods for handling the String elements. Same is true for the elements of any other data type. This results in complex code building. To avoid such complexity, a concept of generic programming is introduced.
2. It saves the programmers burden of creating separate methods for handling data belonging to different data types.
3. It allows the code reusability.
4. Compact code can be created.

Q.3 With an example define a generic class. Or Describe generic classes.

AU : CSE : Dec.-12, 13, May-19

Ans. : A generic class contains one or more variables of generic data type. Following is a simple example which shows how to define the generic class .

```
public class Test<T>
{
    public Test(){val=null;}
    public Test(T val)
    {
        this.val=val;
    }
    public getVal()
    {    return val;
    }
    public setVal()
    {
        val=newValue;
    }
    private T val; //variable defined as of generic type
}
```

Q.4 Can generic be used with inheritance in several ways ? What are they ?

Ans. : Following are the ways by which generic can be used with inheritance -

- Consider a class and a subclass such as **Employee** and **Trainee** . There are two pair classes **Pair<Employee>** and **Pair<Trainee>** which do not possess an inheritance relation. That is,

Pair<Trainee> is not a subclass of **Pair<Employee>** even if these two classes are related to each other.

- The parameterised raw type can be converted to a raw type.
- The generic classes can extend to implement other generic classes.

Q.5 List any two advantages of type parameters.

AU : May-11

Ans. :

1. Due to the use of type parameter it saves the programmers burden of creating separate methods for handling data belonging to different data types.
2. Due to type parameter the early error detection at compile time occurs. This avoids crashing of the code(due to type incompatibility) at run time.

Q.6 State any two challenges of generic programming in virtual machine.

AU : IT : May-13

Ans. :

- i) The virtual machine does not work with generic classes or generic methods. Instead it makes uses raw types in which the raw types are replaced with ordinary java types. Each type variable is replaced with its bound or with object, if it is not bounded. This technique is called type erasure.
- ii) In order to handle the type erasure methods the compiler has to generate bridge methods in corresponding class.



Notes

UNIT-V

8

Graphics Programming

Syllabus

Graphics programming - Frame - Components - working with 2D shapes - Using color, fonts and images.

Contents

8.1	Introduction to Graphics Programming	
8.2	Frame.....	IT : Dec.-11, Marks 8
8.3	Components.....	May-19, Marks 9
8.4	Working with 2D Shapes.....	CSE : Dec.-10, 12, IT : Dec.-11, May-12, 15,19, Marks 16
8.5	Applet	
8.6	Life Cycle of Applet.....	IT : Dec.-12, Marks 16
8.7	Executing an Applet	
8.8	Using Color in Applet.....	CSE : Dec.-10, 11, 13, IT : Dec.-13, Marks 16
8.9	Using Fonts in Applet.....	CSE : Dec.-11, Marks 8
8.10	Using Images in Applet	

8.1 Introduction to Graphics Programming

- The graphics programming in Java is supported by the AWT package. The AWT stands for Abstract Window Toolkit.
- The AWT contains large number of classes which help to include various graphical components in the Java program. These graphical components include text box, buttons, labels, radio buttons, list items and so on. We need to import **java.awt** package for using these components.
- These classes are arranged in hierarchical manner which is recognised as AWT hierarchy.

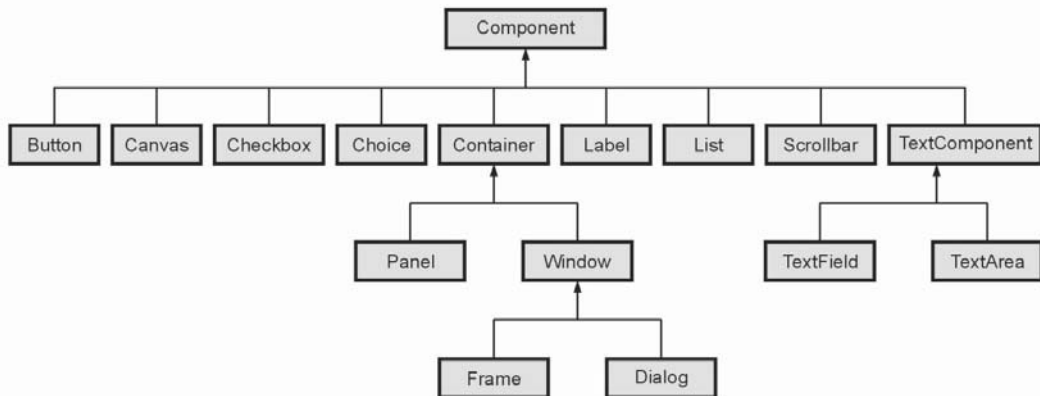


Fig. 8.1.1 AWT hierarchy

- The hierarchy components classes are -
 - **Component** : This is the super class of all the graphical classes from which variety of graphical classes can be derived. It helps in displaying the graphical object on the screen. It handles the mouse and keyboard events.
 - **Container** : This is a graphical component derived from the **component** class. It is responsible for managing the layout and placement of graphical components in the container.
 - **Window** : The top level window without border and without the menu bar is created using the window class. It decides the layout of the window.
 - **Panel** : The panel class is derived from the **container** class. It is just similar to window - without any border and without any menu bar, title bar.
 - **Frame** : This is a top-level window with a border and menu bar. It supports the common window events such as window open, close, activate and deactivate.

8.2 Frame**AU : IT : Dec.-11, Marks 8**

- In Java, Frame is a standard graphical window.
- The frame can be displayed using the **Frame** class.
- The frame drawn using this class has standard minimize, maximize and close buttons.
- The syntax of frame class is -

i) Frame()

This creates the new instance of frame which is invisible initially.

ii) Frame(String title)

This creates the new instance of frame which has some title.

- Following table enlists various methods of **Frame class**

Methods	Description
void setResizable(boolean resizable)	Sets frame to resizable
void setTitle(String Title)	Sets the title of the frame
void setSize(int width,int height)	Sets the width and height of a frame
String getTitle()	Obtains the title of the frame
void setVisible(boolean visible)	Set the frame visible or not.

A frame can be created by two ways -

- By **extending the Frame class**
- By creating an **instance of a Frame class**.

Ex. 8.2.1 : Create a java frame by extending the Frame class.

Sol. :

Java Program

```
import java.awt.*;  
class FrameDemo extends Frame  
{  
    public static void main(String[] args)  
    {
```

```
        FrameDemo fr=new FrameDemo();
        fr.setSize(300,300);
        fr.setVisible(true);
    }
}
```

Output



Note that the initially the frame will not be visible. Hence we need to set the visibility of the frame.

Ex. 8.2.2 : Create a java frame by using an instance of Frame class.

Sol. :

Java Program

```
import java.awt.*;
class FrameDemo1
{
    public static void main(String[] args)
    {
        Frame fr=new Frame();
        fr.setSize(300,300);
        fr.setVisible(true);
    }
}
```

Output will be the same frame as above.

Review Question

1. Write the short note on - Frames.

AU : IT : Dec.-11, Marks 8**8.3 Components****AU : May-19, Marks 9**

- There are various graphical components that can be placed on the frame. These components have the classes. These classes have the corresponding methods.
- When we place the components on the frame we need to set the layout of the frame.
- The commonly used layout is **FlowLayout**. The **FlowLayout** means the components in the frame will be placed from left to right in the same manner as they get added.
- Various components that can be placed for designing user interface are -
 1. Label
 2. Buttons
 3. Canvas
 4. Scroll bars
 5. Text Components
 6. Checkbox
 7. Choices
 8. Lists Panels
 9. Dialogs
 10. Menubar
- Let us discuss these components one by one, but before that let us understand how to create a frame on which we can place the components.

8.3.1 Labels

The syntax of this control is

Label (String s)

Label(String s, int style)

where the **s** of String type represent the string contained by the label similarly in the other label function style is a constant used for the style of label. It can be Label.LEFT, Label.RIGHT and Label.CENTER. Here is a JAVA program which makes use Label.

Ex. 8.3.1 : Write a simple Java program to demonstrate the use of label components.

Sol. :

Java Program[Use_Label.java]

```
import java.awt.*;
class Use_Label
{
    public static void main(String[] args)
    {
        Frame fr=new Frame("This Program is for Displaying the Label");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("OK");
```

```
        Label L2=new Label("CANCEL");
        fr.add(L1);
        fr.add(L2);
    }
}
```

Output

```
C:\>javac Use_label.java
C:\>java Use_label
```



8.3.2 Buttons

- Buttons are sometimes called as **push buttons**. This component contains a label and when it is pressed it generates an event.
- The syntax of this control is
Button (String s)

Java Program

```
import java.awt.*;
class Use_Button
{
    public static void main(String[] args)
    {
        Frame fr=new Frame("This Program is for Displaying the Button");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Button B1=new Button("OK");
        Button B2=new Button("CANCEL");
        fr.add(B1);
        fr.add(B2);
    }
}
```

Output

- We can create an array of buttons. Following is a simple Java program which illustrates this ideas -

Java Program[Use_Button_Arr.java]

```
import java.awt.*;
class Use_Button_Arr
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Buttons");
        fr.setSize(400,200);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Button buttons[]=new Button[5];
        String Fruits[]{"Mango","Orange","Banana","Apple","Strawberry"};
        for(i=0;i<5;i++)
        {
            buttons[i]=new Button(" "+Fruits[i]);
            fr.add(buttons[i]);
        }
    }
}
```

Output



8.3.3 Canvas

- Canvas is a **special area** created on the frame.
- The canvas is specially used for drawing the graphical components such as oval, rectangle, line and so on.
- Various methods of this class are

Method	Description
void setSize(int width, int height)	This method sets the size of the canvas for given width and height.
void setBackground(Color c)	This method sets the background color of the canvas.
void setForeground(Color c)	This method sets the color of the text.

- Following is a simple Java program which shows the use of canvas -

Java Program[Use_Canvas.java]

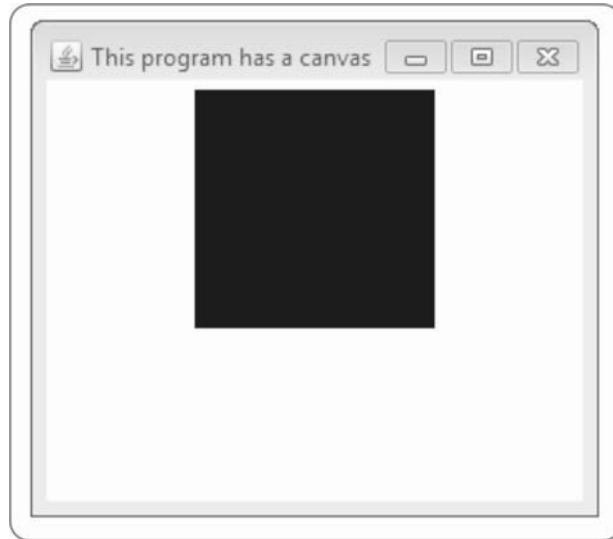
```
import java.awt.*;
class Use_Canvas
{
    public static void main(String[] args)
    {
        Frame Fr = new Frame("This program has a canvas");
        Canvas C1 = new Canvas();
        C1.setSize(120,120);
        C1.setBackground(Color.blue);
        Fr.setLayout(new FlowLayout());
```

```
Fr.setSize(250,250);  
Fr.setVisible(true);  
Fr.add(C1);  
}  
}
```

Output

C:\>javac Use_Canvas.java

C:\>java Use_Canvas



8.3.4 Scrollbars

- Scrollbar can be represented by the slider widgets.
- There are two styles of scroll bars - Horizontal scroll bar and vertical scroll bar.
- Following program shows the use of this component.

Java Program

```
import java.awt.*;  
class Use_ScrollBars  
{  
    public static void main(String[] args)  
    {  
  
        Frame Fr = new Frame("This program has a scrollbars");  
        Scrollbar HSelector = new Scrollbar(Scrollbar.HORIZONTAL);  
        Scrollbar VSelector = new Scrollbar(Scrollbar.VERTICAL);  
  
        Fr.setLayout(new FlowLayout());
```

```
Fr.setSize(300,300);  
Fr.setVisible(true);  
Fr.add(HSelector);  
Fr.add(VSelector);  
}  
}
```

Output



8.3.5 Text Components

- In Java there are two controls used for text box - One is **TextField** and the other one is **TextArea**.
- The **TextField** is a slot in which one line text can be entered. In the **TextField** we can enter the string, modify it, copy, cut or paste it. The syntax for the text field is
`int TextField(int n)`

where n is total number of characters in the string.

- The **TextArea** control is used to handle multi-line text. The syntax is -
`TextArea(int n,int m)`

where n is for number of lines and m is for number of characters.

Following is a Java program which illustrates the use of **TextField** and **TextArea**.

Java Program[Use_TxtFld.java]

```
import java.awt.*;
class Use_TxtFld
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the TextField");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Label L1=new Label("Enter your name here");
        TextField input1=new TextField(10);
        Label L2=new Label("Enter your address here");
        TextArea input2=new TextArea(10,20);
        fr.add(L1);
        fr.add(input1);
        fr.add(L2);
        fr.add(input2);
    }
}
```

Output**8.3.6 Checkbox**

- Checkbox is basically a small box which can be ticked or not ticked.
- In Java we can select particular item using checkbox control.

- This control appears as small box along with label. The label tells us the name of the item to be selected.
- The syntax of checkbox is as given below -
`Checkbox(String label)`

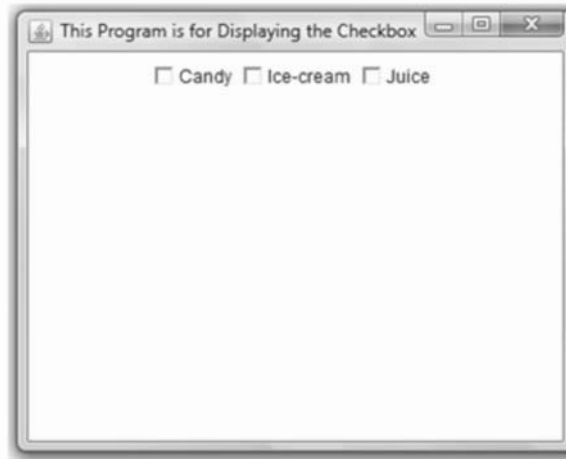
where *label* denotes the label associated with each checkbox.

- To get the state of the checkbox the **getState()** method can be used.

Java Program[Use_ChkBox.java]

```
import java.awt.*;
class Use_ChkBox
{
    public static void main(String[] args)
    {
        int i;
        Frame fr=new Frame("This Program is for Displaying the Checkbox");
        fr.setSize(350,300);
        fr.setLayout(new FlowLayout());
        fr.setVisible(true);
        Checkbox box1=new Checkbox("Candy");
        Checkbox box2=new Checkbox("Ice-cream");
        Checkbox box3=new Checkbox("Juice");
        fr.add(box1);
        fr.add(box2);
        fr.add(box3);
    }
}
```

Output



8.3.7 Checkbox Group

- The Checkbox Group component allows the user to make one and only one selection at a time.
- These checkbox groups is also called as **radio buttons**. The syntax for using checkbox groups is -

Checkbox(String str ,CheckboxGroup cbg , Boolean val);

- Following is a simple Java program which makes use of this control.

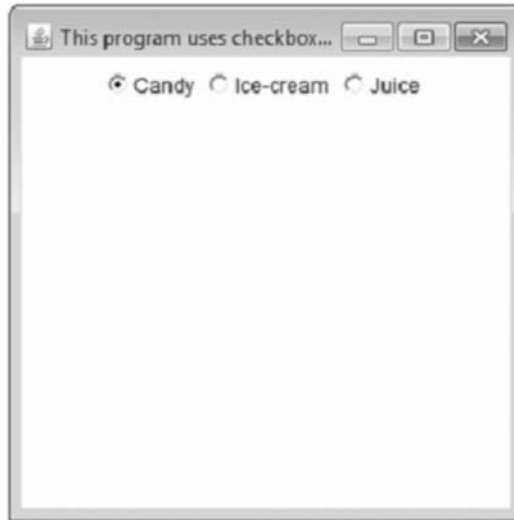
Java Program[Use_CheckBoxGr.java]

```
import java.awt.*;
class Use_CheckBoxGr
{
    public static void main(String[] args)
    {

        Frame Fr = new Frame("This program uses checkbox groups");
        Fr.setLayout(new FlowLayout());
        Fr.setSize(300,300);
        Fr.setVisible(true);
        CheckboxGroup cbg=new CheckboxGroup();
        Checkbox box1=new Checkbox("Candy",cbg,true);
        Checkbox box2=new Checkbox("Ice-cream",cbg,false);
        Checkbox box3=new Checkbox("Juice",cbg,false);
```

```
Fr.add(box1);  
Fr.add(box2);  
Fr.add(box3);  
}  
}
```

Output



8.3.8 Choices

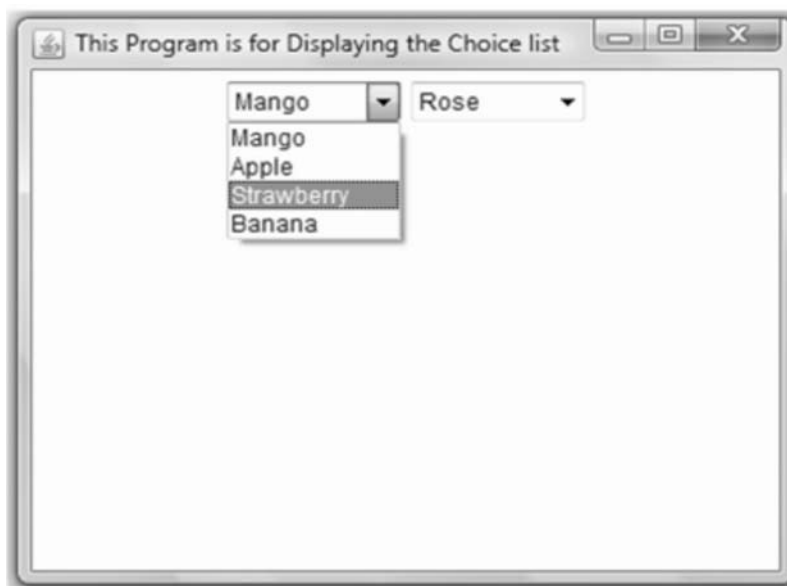
- This is a simple control which allows the popup list for selection.
- We have to create an object of type **choice** as follows -
Choice obj=new Choice();

Java Program[Use_Choice.java]

```
import java.awt.*;  
class Use_Choice  
{  
    public static void main(String[] args)  
    {  
        int i;  
        Frame fr=new Frame("This Program is for Displaying the Choice list");  
        fr.setSize(350,300);  
        fr.setLayout(new FlowLayout());  
        fr.setVisible(true);  
        Choice c1=new Choice();  
        Choice c2=new Choice();  
        c1.add("Mango");
```

```
c1.add("Apple");  
c1.add("Strawberry");  
c1.add("Banana");  
  
c2.add("Rose");  
c2.add("Lily");  
c2.add("Lotus");  
fr.add(c1);  
fr.add(c2);  
  
}  
}
```

Output



8.3.9 List Panels

- **List** is a collection of many items.
- By double clicking the desired item we can select it. Following Java program makes use of this control -

Java Program[Use_List.java]

```
import java.awt.*;  
class Use_List  
{  
    public static void main(String[] args)
```

```
{  
    int i;  
    Frame fr=new Frame("This Program is for Displaying the List");  
    fr.setSize(350,300);  
    fr.setLayout(new FlowLayout());  
    fr.setVisible(true);  
    List flower=new List(4,false);  
    flower.add("Rose");  
    flower.add("Jasmine");  
    flower.add("Lotus");  
    flower.add("Lily");  
    fr.add(flower);  
}
```

Output



Review Question

1. Write the short note on - Frames. **AU : IT : Dec.-11, Marks 8**
2. Use graphics objects to draw an arc and a semicircle inside a rectangular box. **AU : May-19, Marks 4**
3. Sketch the hierarchy of Java AWT classes and methods. Create a 'checkbox' using these classes and methods. **AU : May-19, Marks 4**

8.4 Working with 2D Shapes

AU : CSE : Dec.-10, 12, IT : Dec.-11, May-12, 15, 19, Marks 16

- Java has an ability to draw various graphical shapes. The applet can be used to draw these shapes.
- These objects can be colored using various colors.

- Every applet has its own area which is called **canvas** on which the display can be created.
- The co-ordinate system of Java can be represented by following Fig. 8.4.1.

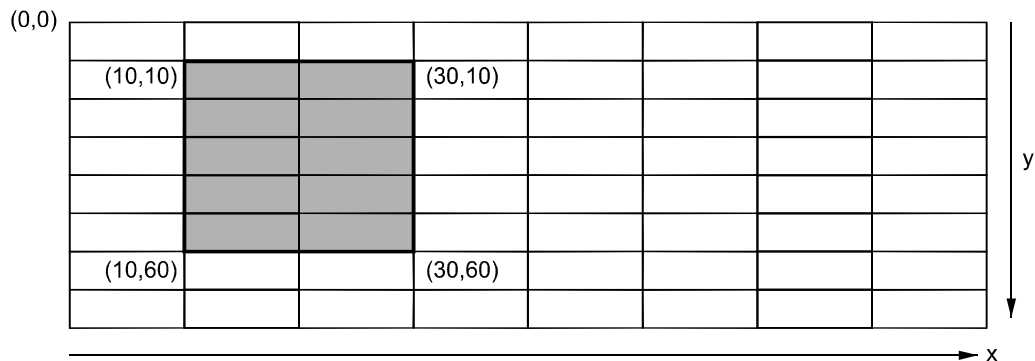


Fig. 8.4.1 Co-ordinate system

- Before drawing the 2D shapes we will need a special area on the frame. This area is called **canvas**. Various methods of this class are -

void setSize(int width,int height) - Sets the size of the canvas of given width and height.

void setBackground(Color c) - Sets the background color of the canvas.

void setForeground(Color c) - Sets the color of the text.

8.4.1 Lines

- Drawing the line is the simplest thing in Java. The syntax is,
`void drawLine(int x1,int y1,int x2,int y2);`

Where x1 and y1 represents the starting point of the line and x2 and y2 represents the ending point of the line.

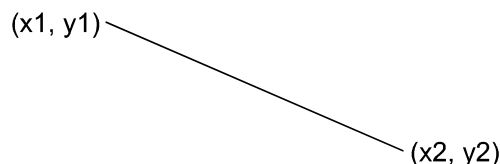


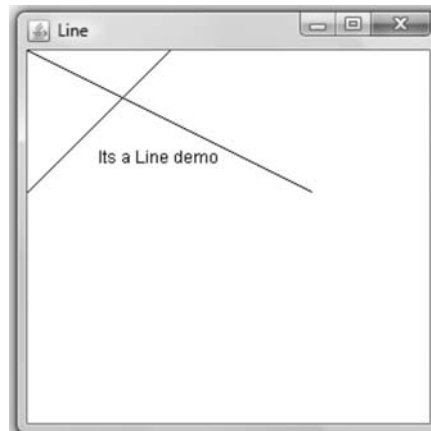
Fig. 8.4.2 Line

Here is a demonstration -

Java Program[LineDemo.java]

```
import java.awt.*;
class LineDemo extends Canvas
{
    public LineDemo()
```

```
{
    setSize(200,200);
    setBackground(Color.white);
}
public static void main(String[] args)
{
    LineDemo obj=new LineDemo();
    Frame fr=new Frame("Line");
    fr.setSize(300,300);
    fr.add(obj);
    fr.setVisible(true);
}
public void paint(Graphics g)
{
    g.drawLine(0,0,200,100);//diagonal line from top-left
    g.drawLine(0,100,100,0);//another diagonal line
    g.drawString("Its a Line demo",50,80);
}
}
```

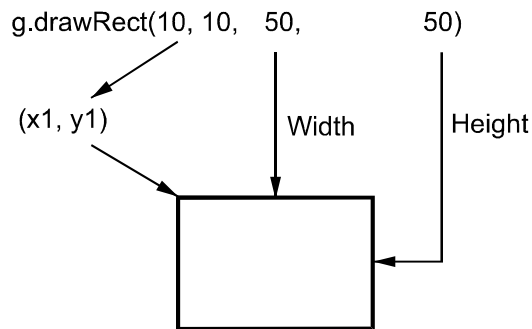
Output

8.4.2 Rectangle

In Java we can draw two types of rectangles: normal rectangle and the rectangle with round corners. The syntax of these methods are -

`void drawRect(int top,int left,int width,int height)`

`void drawRoundRect(int top,int left,int width,int height,int xdimeter,int ydimeter)`

For example**Fig. 8.4.3 Rectangle**

The rectangle can be filled up with following methods,

`void fillRect(int top,int left,int width,int height)`

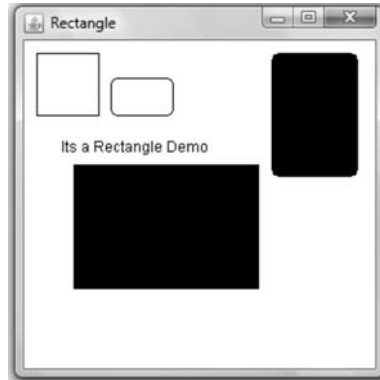
`void fillRoundRect(int top,int left,int width,int height,int xdimeter,int ydimeter)`

Java Program[RectangleDemo.java]

```
import java.awt.*;
class RectangleDemo extends Canvas
{
    public RectangleDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        RectangleDemo obj=new RectangleDemo();
        Frame fr=new Frame("Rectangle");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawRect(10,10,50,50);
        g.drawRoundRect(70,30,50,30,10,10);
        g.fillRect(40,100,150,100);
    }
}
```

```
g.fillRoundRect(200,10,70,100,10,10);  
g.drawString("Its a Rectangle Demo",30,90);  
}  
}
```

Output



8.4.3 Oval

To draw circle and ellipse we can use the function **drawOval()** method. The syntax of this method is ,

```
void drawOval(int top, int left, int width, int height)
```

To fill this oval we use **fillOval()** method. The syntax of this method is,

```
void fillOval(int top, int left, int width, int height)
```

We can specify the color for filling up the rectangle using **setColor** method. For example

```
g.drawOval(10,10,200,100);  
g.setColor(Color.blue);  
g.fillOval(30,50,200,100);
```

The *top* and *left* values specify the upper left corner and *width* and *height* is for specifying the width and heights respectively.

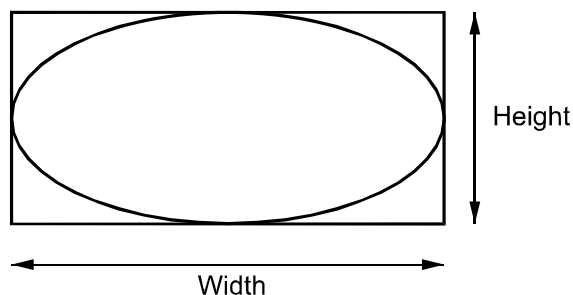
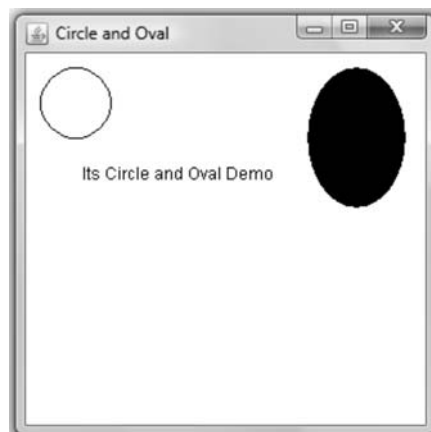


Fig. 8.4.4 Ellipse

Let us understand the functioning of these methods with the help of some example

Java Program[OvalDemo.java]

```
import java.awt.*;
class OvalDemo extends Canvas
{
    public OvalDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        OvalDemo obj=new OvalDemo();
        Frame fr=new Frame("Circle and Oval");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawOval(10,10,50,50);
        g.fillOval(200,10,70,100);
        g.drawString("Its Circle and Oval Demo",40,90);
    }
}
```

Output**8.4.4 Arc**

To draw an arc the **drawArc()** and to fill an arc **fillArc()** are the functions. The syntax of these methods is as follows -

```
void drawArc(int top,int left,int width,int height,int angle1,int angle2)
void fillArc(int top,int left,int width,int height,int angle1,int angle2)
```

The *angle1* represents the starting angle and the *angle2* represents the angular distance.

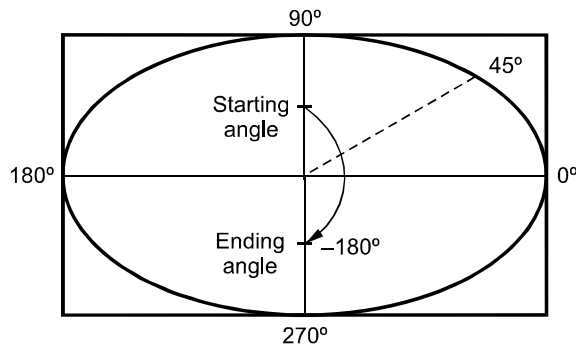
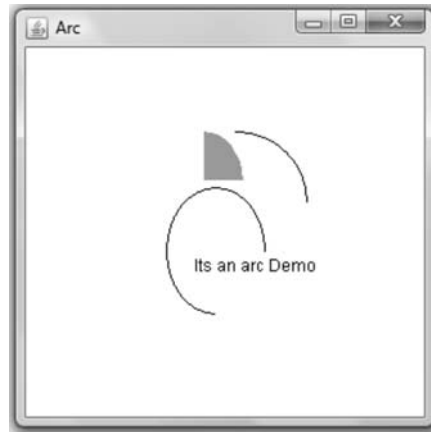


Fig. 8.4.5

Java Program[arcDemo.java]

```
import java.awt.*;
class arcDemo extends Canvas
{
    public arcDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        arcDemo obj=new arcDemo();
        Frame fr=new Frame("Arc");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawArc(100,60,100,100,0,90);
        g.setColor(Color.green);//fills the arc with green
        g.fillArc(100,60,55,70,0,90);
        g.setColor(Color.black);
        g.drawArc(100,100,70,90,0,270);
        g.drawString("Its an arc Demo",120,160);
    }
}
```

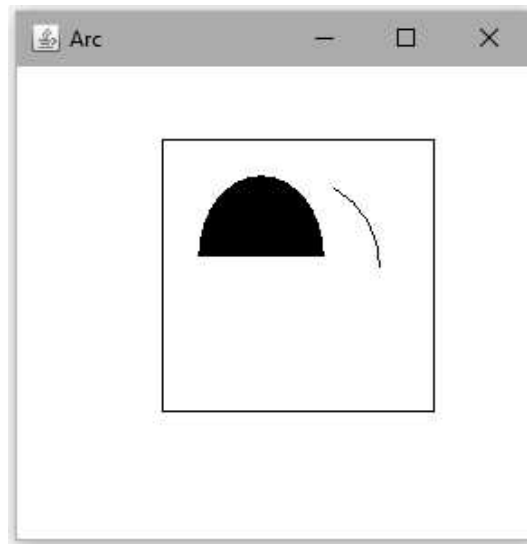
Output

Ex. 8.4.1 : Use graphics object method to draw an arc and semicircle inside a rectangular box

AU : May-19, Marks 4

Sol. :

```
import java.awt.*;
class arcDemo extends Canvas
{
    public arcDemo()
    {
        setSize(200,200);
        setBackground(Color.white);
    }
    public static void main(String[] args)
    {
        arcDemo obj=new arcDemo();
        Frame fr=new Frame("Arc");
        fr.setSize(300,300);
        fr.add(obj);
        fr.setVisible(true);
    }
    public void paint(Graphics g)
    {
        g.drawArc(100,60,100,100,0,60);//arc
        g.fillArc(100,60,70,90,0,180);//semicircle
        g.drawRect(80,40,150,150);
    }
}
```

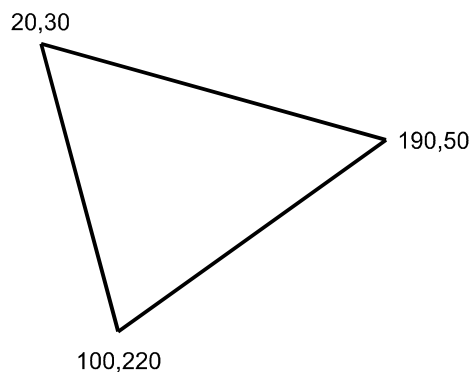
Output**8.4.5 Polygons**

In order to draw polygons following function is used

`Polygon(int[] xpoints,int[] ypoints,int npoints)`

The *xpoints* represent the array of *x* co-ordinates. The *ypoints* represent the array of *y* co-ordinates. And the *npoints* represents the number of points.

For filling up the polygon the **fillPolygon** method is used.

**Java Program[Polyg.java]**

```
import java.awt.*;
class Polyg extends Canvas
{
    public Polyg()
    {
        setSize(200,200);
    }
}
```

```
setBackground(Color.white);
}
public static void main(String[] args)
{
    Polyg obj=new Polyg();
    Frame fr=new Frame("Polygon");
    fr.setSize(300,300);
    fr.add(obj);
    fr.setVisible(true);
}
public void paint(Graphics g)
{
    int xpt[]={50,20,20,20,130};
    int ypt[]={80,30,200,200,30};
    int num=5;
    g.drawPolygon(xpt,ypt,num);
    g.setColor(Color.magenta);
    g.fillPolygon(xpt,ypt,num);
    g.setColor(Color.black);
    g.drawString("Its a polygon Demo",100,100);
}
}
```

Output



Review Questions

1. List the methods available in the draw shapes. **AU : CSE : Dec.-10, Marks 4**
2. Write the short note on - Graphics programming. **AU : IT : Dec.-11, Marks 8, CSE : Dec.-12, Marks 16**
3. How will you draw the following graphics in a window ? i) Arcs ii) Ellipses and circles in Java. **AU : IT : May-12, Marks 6**
4. With an example describe in detail about how to work with 2D Shapes in Java. **AU : May-15, Marks 8**

8.5 Applet

- Applets are the small Java programs that can be used in internetworking environment.
- These programs can be transferred over the internet from one computer to another and can be displayed on various web browsers.
- Various **applications of applets** are in performing arithmetic operations, displaying graphics, playing sounds, creating animation and so on.
- Following are the **situations in which we need to use applet** -
 1. For displaying the dynamic web pages we need an applet. The dynamic web page is a kind of web page on which the contents are constantly changing. For example an applet that can represent the sorting process of some numbers. During the process of sorting the positions of all the elements is changing continuously.
 2. If we want some special effects such as sound, animation and much more then the applets are used.
 3. If we want that the particular application should be used by any user who might be located remotely . Then in such situation the applets are embedded into the web pages and can be transferred over the internet.

8.6 Life Cycle of Applet

IT : Dec.-12, Marks 16

There are various methods which are typically used in applet for initialization and termination purpose. These methods are

1. Initialization
2. Running state
3. Idle state
4. Dead or destroyed state (Refer Fig. 8.6.1. for applet's life cycle)

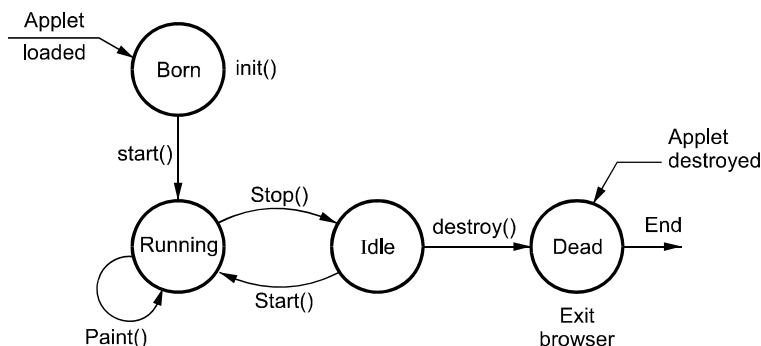


Fig. 8.6.1 Applet's life cycle

When applet begins, the AWT calls following methods in sequence -

- a) `init()` b) `start()` c) `paint()`

When applet is terminated following methods are invoked in sequence.

- a) `stop()` b) `destroy()`

1. Initialization state

When applet gets loaded it enters in the initialization state. For this purpose the **`init()`** method is used. In this method you can initialize the required variables. This method is called only once initially at the execution of the program. The syntax can be

```
public void init()
{
    //initialization of variables
}
```

In this method various tasks of initialization can be performed such as -

1. Creation of objects needed by applet
2. Setting up of initial values
3. Loading of image
4. Setting up of colors.

2. Running state

When the applet enters in the running state, it invokes the **`start()`** method of Applet class.

This method is called only after the `init` method. After stopping the applet when we restart the applet at that time also this method is invoked. The syntax can be

```
public void start()
{
    ...
}
```

3. Display state

Applet enters in the display state when it wants to display some output. This may happen when applet enters in the running state. The **`paint()`** method is for displaying or drawing the contents on the screen. The syntax is

```
public void paint(Graphics g)
{
    ...
}
```

An instance of Graphics class has to be passed to this function as an argument. In this method various operations such as display of text, circle, line are invoked.

4. Idle state

This is an idle state in which applet becomes idle. The **stop()** method is invoked when we want to stop the applet. When an applet is running if we go to another page then this method is invoked. The syntax is

```
public void stop()
{
    ...
}
```

5. Dead state

When applet is said to be dead then it is removed from memory. The method **destroy()** is invoked when we want to terminate applet completely and want to remove it from the memory.

```
public void destroy()
{
    ...
}
```

It is a good practice to call stop prior to destroy method.

Review Question

1. Explain about applet lifecycle ? How applets are prepared and executed.

AU : IT : Dec.-12, Marks 16

8.7 Executing an Applet

There are two methods to run the applet

1. Using web browser
- 2 Using Appletviewer

Let us learn both the methods with necessary illustrations

1. Using web browser

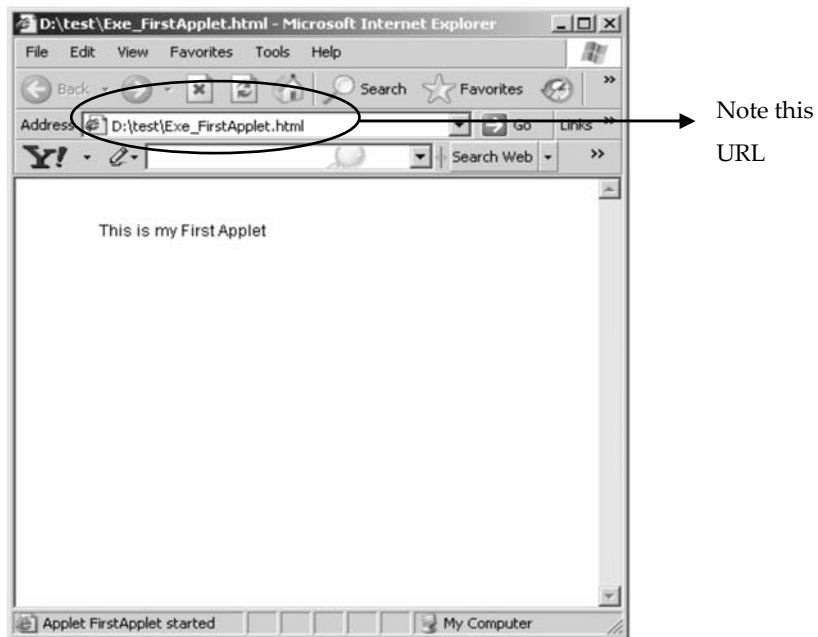
Step 1 : Compile your Applet source program using javac compiler, i.e.

```
D:\test>javac FirstApplet.java
```

Step 2 : Write following code in Notepad/Wordpad and save it with filename and extension **.html**. For example following code is saved as *Exe_FirstApplet.html*, The code is

```
<applet code="FirstApplet" width=300 height=100>
</applet>
```

Step 3 : Load html file with some web browser, This will cause to execute your html file. It will look this -



2. Using Appletviewer

Step 1 : To run the applet without making use of web browser or using command prompt we need to modify the code little bit. This modification is as shown below

```
/*
This is my First Applet program
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="FirstApplet" width=300 height=100>
</applet>
*/
public class FirstApplet extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("This is my First Applet",50,30);
    }
}
```

In above code we have added one more comment at the beginning of the program
<applet code="FirstApplet" width=300 height=100>

</applet>

This will help to understand Java that the source program is an applet with the name FirstApplet. By this edition you can run your applet program merely by **Appletviewer** command.

Step 2 :

```
D:\test>javac FirstApplet.java
```

```
D:\test>Appletviewer FirstApplet.java
```

And we will get



Note one fact about applet is that: it does not require **main** function.

8.8 Using Color in Applet

AU : CSE : Dec.-10, 11, 13; IT : Dec.-13, Marks 16

When you want to specify the color in order to make your applet colourful, there are some methods supported by AWT system. The syntax of specifying color is

```
Color(int R,int G,int B);
```

```
Color(int RGBval); //here RGBval denotes the value of color
```

Similarly to set the background color of applet window we use

```
void setBackground(Color colname)
```

where *colname* denotes the name of the background color. In the same manner we can also specify the foreground color i.e. color of the text by using method

```
void setForeground(Color colname)
```

We can specify the *colorname* using the **Color** object as follows -

Color.black	Color.lightGray
Color.blue	Color.magenta
Color.cyan	Color.orange
Color.darkGray	Color.pink
Color.gray	Color.red
Color.green	Color.white
	Color.yellow

Here is a simple demo -

Java Program[ColorDemo.java]

```
import java.awt.*;
import java.applet.*;

/*
<applet code="ColorDemo" width=300 height=100>
</applet>
*/
public class ColorDemo extends Applet
{
    public void paint(Graphics g)
    {
        setBackground(Color.cyan);
        setForeground(Color.red);
        g.drawString("Its a colorful Applet",50,30);
        Color newColor=new Color(255,255,0);
        //creating red+green=yellow color
        g.setColor(newColor);
        g.drawString("Its a colorful Applet",50,70);
    }
}
```

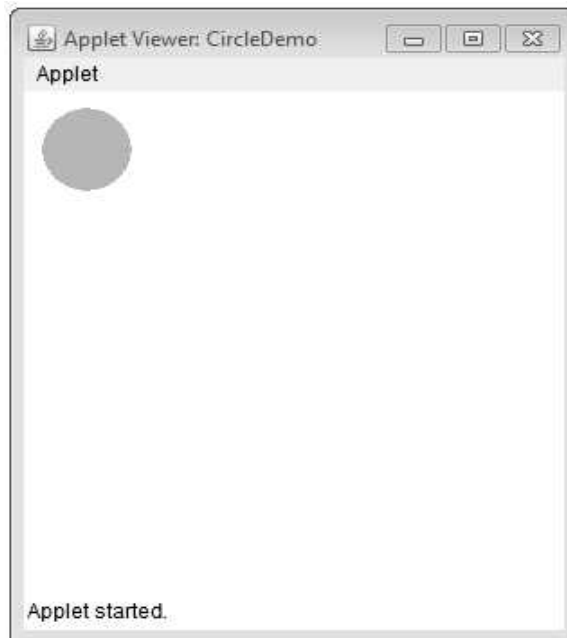
Output



In above program we see one more method related to color and that is **setColor**. To set some specific color this method is used. The color name has to be passed as a parameter to this method. We can create our own color by a method **Color(int R,int G,int B)**. In above program we have created yellow color(Red + Green = Yellow) and then using *newColor* in **setColor** method the string '*It's a colourful applet*' is written on the new line.

Ex. 8.8.1 : Write an applet program for displaying the circle in green color.**Sol. :**

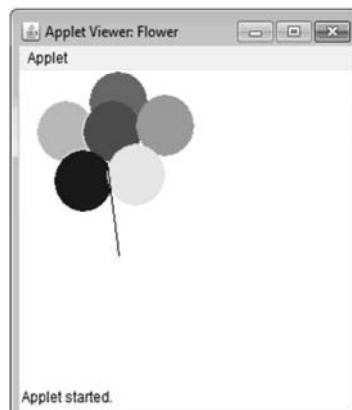
```
import java.awt.*;
import java.applet.*;
/*
<applet code="CircleDemo" width=300 height=300>
</applet>
*/
public class CircleDemo extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.green);
        g.fillOval(10,10,50,50);
    }
}
```

Output

Ex. 8.8.2 : Write an applet program to draw a flower with color packages.**AU : CSE : Dec.-11, Marks 16****Sol. :**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="Flower" width=300 height=300>
</applet>
*/
public class Flower extends Applet
{
    public void paint(Graphics g)
    {
        g.setColor(Color.magenta);
        g.fillOval(60,1,50,50);

        g.setColor(Color.pink);
        g.fillOval(15,25,50,50);
        g.setColor(Color.red);
        g.fillOval(55,25,50,50);
        g.setColor(Color.blue);
        g.fillOval(30,65,50,50);
        g.setColor(Color.green);
        g.fillOval(100,20,50,50);
        g.setColor(Color.yellow);
        g.fillOval(75,60,50,50);
        g.setColor(Color.black);
        g.drawLine(75,75,85,150);
    }
}
```

Output

Ex. 8.8.3 : An analysis of examination results at a school gave the following distribution of grades for all subjects taken in one year :

<i>Grade</i>	<i>Percentage</i>
<i>A</i>	<i>10</i>
<i>B</i>	<i>25</i>
<i>C</i>	<i>45</i>
<i>D</i>	<i>20</i>

Write a java program to represent the distribution of each grade in a pie chart, where each slice of pie is differently colored.

AU : IT : Dec.-13, Marks 10

Sol. :

```
import java.awt.*;
import java.applet.*;
/*
<applet code="ExaminationResults" width=300 height=300>
</applet>
*/
class Slice
{
    double value;
    Color color;
    public Slice(double value, Color color)
    {
        this.value = value;
        this.color = color;
    }
}
public class ExaminationResults extends Applet
{
    Slice[] slices = {new Slice(10, Color.magenta),new Slice(25, Color.green),new Slice(20,
    Color.red),new Slice(45, Color.blue)};

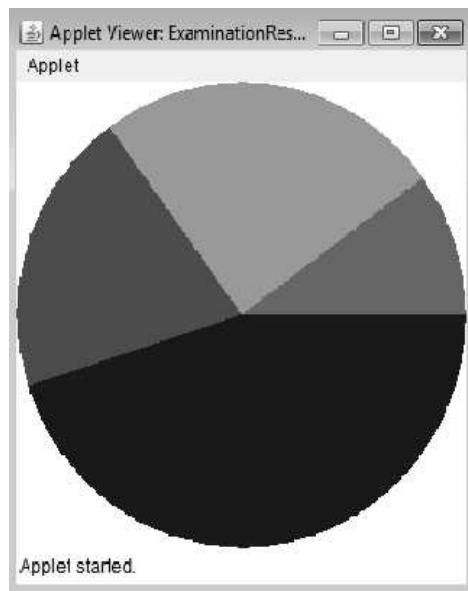
    public void paint(Graphics g)
    {
        drawPie((Graphics2D) g, getBounds(), slices);
    }

    void drawPie(Graphics2D g, Rectangle area, Slice[] slices)
    {
        double total = 0.0;
        for (int i = 0; i < slices.length; i++)
        {
            total += slices[i].value;
```

Storing grades and
colors in an array

Drawing the piechart with
each slice of different colors

```
}
double Value = 0.0;
int startAngle = 0;
for (int i = 0; i < slices.length; i++)
{
    startAngle = (int) (Value * 360 / total);
    int arcAngle = (int) (slices[i].value * 360 / total);
    g.setColor(slices[i].color);
    g.fillArc(area.x, area.y, area.width, area.height, startAngle, arcAngle);
    Value += slices[i].value;
}
}
```

Output

Ex. 8.8.4 : Write a Java program to plot the path of small circle moving around the circumference of a larger circle.

AU : IT : Dec.-13, Marks 16

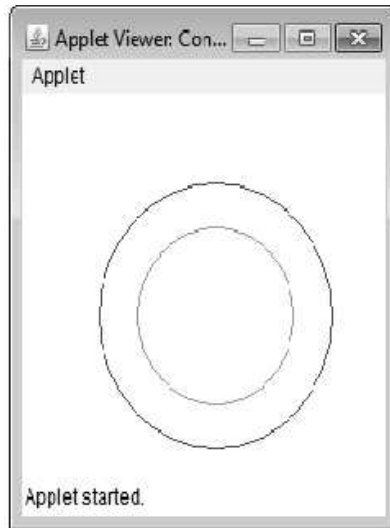
Sol. :

```
import java.applet.*;
import java.awt.*;
/*<applet code="ConcCircles" height=250 width=300>
  </applet>
*/
```

```
public class ConcCircles extends Applet
{
```

```
public void paint(Graphics g)
{
    g.setColor(Color.RED);           //Set the colour to red
    g.drawOval(50,50,150,150);
    g.setColor(Color.GREEN);         // Set the colour to green
    g.drawOval(75,75,100,100);       // 100 pixel diameter circle is drawn
}
}
```

Output



Review Question

1. List the methods available in the graphics for COLOR.

AU : CSE : Dec.-10, Marks 4, Dec.-11, Marks 8

8.9 Using Fonts in Applet

AU : CSE : Dec.-11, Marks 8

We can obtain the list of all the fonts that are present in our computer with the help of **getAvailableFontFamilyNames()**. This method is member of **GraphicsEnvironment**. First of all we need a reference to **GraphicsEnvironment**. For obtaining reference to **GraphicsEnvironment** we can call the **getLocalGraphicsEnvironment()** method. Then using this reference **getAvailableFontFamilyNames()** is invoked.

Following Java applet is for obtaining the list of available fonts

Java Program[FontListDemo.java]

```
import java.awt.*;
import java.applet.*;
```

```
/*
<applet code="FontListDemo" width=500 height=500>
</applet>
*/
public class FontListDemo extends Applet
{
    public void paint(Graphics g)
    {
        String Font_names="";
        String F[];
        int k=0;
        //getting reference to GraphicsEnvironment
        GraphicsEnvironment ge =
        GraphicsEnvironment.getLocalGraphicsEnvironment();
        F=ge.getAvailableFontFamilyNames();
        for(int i=0;i<F.length;i++)
        {
            Font_names=F[i];
            g.drawString(Font_names,0,10+k);
            k=k+15;
        }
    }
}
```

To print the message on next line y-co-ordinate is incremented.

Output

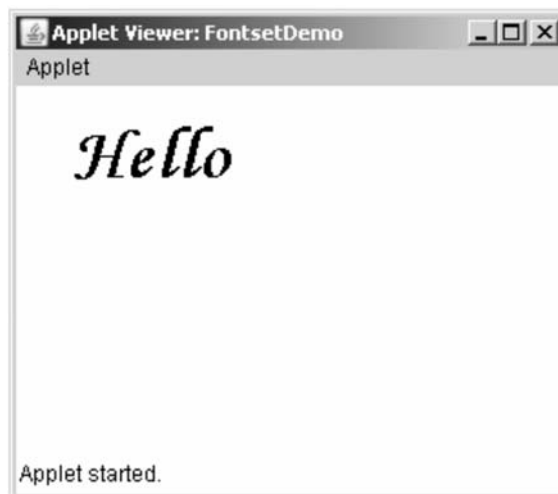


We can set the desired font using **setFont()** function. This function requires a reference parameter which can be created by the method **Font()**. In **Font()** method the first parameter is name of the font, second parameter denotes style of the font which can BOLD, ITALIC or PLAIN and third parameter denotes the size of font. Here is a simple program which is demonstrate the same

Java Program[FontsetDemo.java]

```
import java.awt.*;
import java.applet.*;
/*
<applet code="FontsetDemo" width=310 height=200>
</applet>
*/
public class FontsetDemo extends Applet
{
    String msg=" ";
    public void init()
    {
        Font f;
        f=new Font("Monotype Corsiva",Font.BOLD,40);
        msg="Hello";
        setFont(f);
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,30,50);
    }
}
```

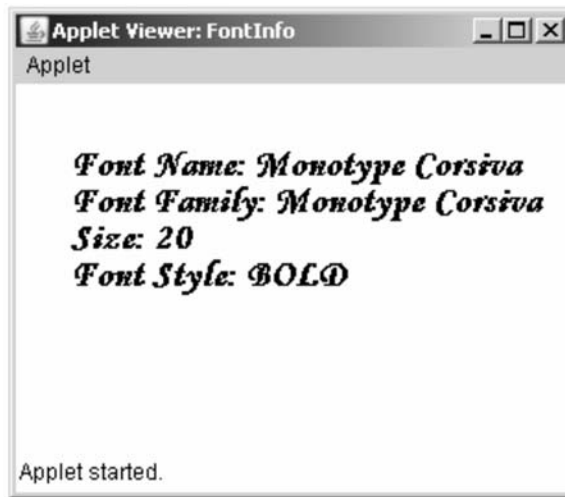
Output



Similarly we can get the information about the font. In the following Java program in the **init** method we have set a specific font and in the **paint** method we can get the font information using **getFont()** method.

Java Program[FontInfo.java]

```
import java.awt.*;
import java.applet.*;
/*
<applet code="FontInfo" width=300 height=200>
</applet>
*/
public class FontInfo extends Applet
{
    String msg=" ";
    Font f;
    public void init()
    {
        f=new Font("Monotype Corsiva",Font.BOLD,20);
        setFont(f);
    }
    public void paint(Graphics g)
    {
        f=g.getFont();
        String name=f.getName();
        msg="Font Name: "+name;
        g.drawString(msg,30,50);
        String family=f.getFamily();
        msg="Font Family: "+family;
        g.drawString(msg,30,70);
        int size= f.getSize();
        msg="Size: "+size;
        g.drawString(msg,30,90);
        int style=f.getStyle();//we int value of font Style
        if((style & Font.PLAIN)==Font.PLAIN)
            msg="Font Style: PLAIN";
        if((style & Font.BOLD)==Font.BOLD)
            msg="Font Style: BOLD";
        if((style & Font.ITALIC)==Font.ITALIC)
            msg="Font Style: ITALIC";
        g.drawString(msg,30,110);
    }
}
```

Output**Review Question**

1. Explain the usage of special fonts for text in Graphics programming.

AU : CSE : Dec.-11, Marks 8**8.10 Using Images in Applet**

- The image can be displayed in applet using **drawImage** method of **Graphics** class. Before using the **drawImage** we must get the image file. This can be obtained using the method **getImage**.

- **Syntax**

```
public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer)
public Image getImage(URL u, String image)
{
}
}
```

- **Example**

```
import java.awt.*;
import java.applet.*;
/*
<applet code="ImageDisplayDemo" width=300 height=300>
</applet>
*/
public class ImageDisplayDemo extends Applet
{
    Image Img;
    public void init()
    {
        Img = getImage(getDocumentBase(),"myimg.jpg");
    }
}
```

```
    }  
    public void paint(Graphics g)  
    {  
        g.drawImage(Img, 50,50, this);  
    }  
}
```

Review Question

1. Explain how to display an image in applet ?

Two Marks Questions with Answers

Q.1 What are the steps needed to show a Frame ?

AU : CSE : Dec.-10

OR How are frames created in Java ?

AU : CSE : Dec.-11

Ans. : Frame can be created by following two ways -

- 1) The frame can be created by extending the frame class. For example
class FrameDemo extends Frame

```
{  
    public void static main(String[] arg)  
    {  
        FrameDemo fr=new FrameDemo();  
        fr.setSize(300,300);  
        fr.setVisible(true);  
    }  
}
```

- 2) The frame can be created by using the instance of Frame class -
class FrameDemo1

```
{  
    public static void main(String[] arg)  
    {  
        Frame fr=new Frame();  
        fr.setSize(300,300);  
        fr.setVisible(true);  
    }  
}
```

Q.2 What is meant by frame window ?

AU : IT : May-12

Ans. : The frame window is a standard graphical window. It can be displayed by the class named **Frame**. This frame window has standard minimize, maximize and close buttons.

Q.3 What is AWT ?**AU : IT : May-12, Dec.-12**

Ans. : The AWT stands for Abstract Window Toolkit. It is a class library that contains various classes and interfaces that are required for graphical programming. The AWT provides the support for drawing graphical shapes, windows, buttons, text boxes, menus and so on.

Q.4 How do you manage the color and font of graphics in applet ?**AU : CSE : May-12**

Ans. : The color can be specified as
`color(int R,int G,int B);`

The background color can be set by
`setBackground(Color colname)`

The color of the text can be set by
`setForeground(Color colname)`

The font can be set by **setFont()** method. This function requires a reference parameter which can be created by the method **Font()**. The Font() method has first parameter as name of the font, second parameter denotes the style of the font which can be BOLD, ITALIC or PLAIN. The third parameter denotes the size of the font.

Q.5 Code a Graphics method in Java to draw the String "Hello World" from the coordinates(100,200)**AU : IT : Dec.-13**

Ans. :

```
Import java.applet;  
/*  
<applet code="MsgDemo" width=300 height=300>  
</applet>  
*/  
public class MsgDemo extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("Hello World",100,200);  
    }  
}
```

Q.6 How does radio button in Java differ from check box ?**AU : IT : Dec.-13**

Ans. : The radio buttons are mutually exclusive and user must select exactly one choice from the given list. Whereas the check box is for selecting any number options from the given choices.

Q.7 Enumerate features of AWT in Java**AU : CSE : Dec.-18**

Ans. : 1) AWT stands for Abstract Window Toolkit. It includes large number of classes to include **graphical components** such as text box, buttons, labels and radio buttons and so on.

2) AWT classes allows to **handle the events** associated with graphical components.

Q.8 Write the class hierarchy for Panel and Frame**AU : May 19**

Ans. : Refer fig.8.1.1

Q.9 State the purpose of getRed(), getBlue(), and getGreen() methods**AU : May 19**

Ans. : The getRed(), getBlue() and getGreen() returns the red component, blue component or green component of color in the range 0.0 – 1.0



Notes

UNIT-V

9

Event Handling

Syllabus

Basics of event handling - event handlers - adapter classes - actions - mouse events - AWT event hierarchy

Contents

9.1 Basics of Event Handling

9.2 Event handlers **IT : May-13, CSE : May-13, Dec.-18**.....Marks 16

9.3 Adapter Classes **IT : Dec.-11, CSE : Dec.-13, .Marks 8**

9.4 Actions

9.5 Mouse Events..... **Dec.-19**Marks 13

9.6 AWT Event Hierarchy..... **IT : Dec.-12, 13,**Marks 16

9.1 Basics of Event Handling

9.1.1 Event

- Event means any activity that interrupts the current ongoing activity.
- For example : when user clicks mouse or press some key from a keyboard during some processing, then it generates an event.
- In Java, events represent all activity that is carried out between the user and the application.
- Some specific actions are associated with these events. **Java's Abstract Windowing Toolkit (AWT)** conveys these actions to the programs.
- When the user interacts with a program let us say by clicking a button, the system creates an event representing the click action and passes it to the **event-handling code** within the program.
- This code is then responsible for giving appropriate response.

9.1.2 Event Delegation Model

Event delegation model is used for understanding the event and for processing it. The event-handler method takes the Event object as a parameter. For handling particular event specific object of event must be mentioned. There are four main components based on this model are

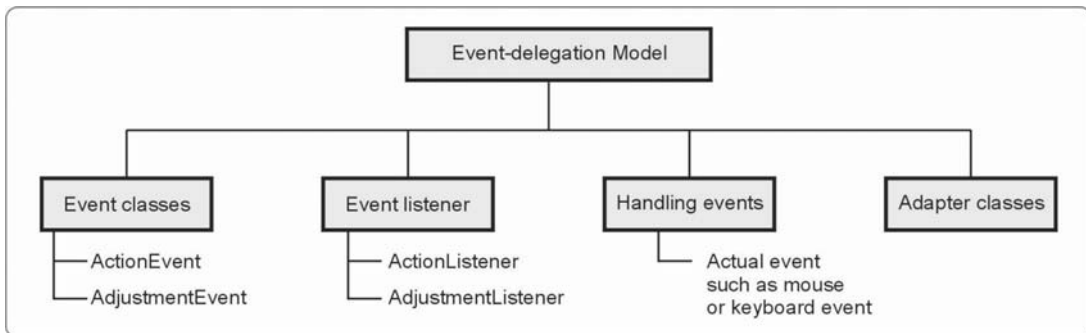


Fig. 9.1.1 Components of event delegation model

Advantages of Event Delegation Model

Following are the advantages of event delegation model -

1. In event delegation model the events are handled using objects. This allows a clear separation between the usage of the components and the design.
2. It accelerates the performance of the application in which multiple events are used.

9.1.2.1 Example : Handling Button Click

The buttons are sometimes called as push buttons. We can associate some event on button click. That means on clicking the button, certain event gets triggered to perform required task. Following example illustrates this idea.

Ex. 9.1.1 : Write a Java program to toggle the background color on every click of button.

Sol. :

Java Program[Button1.java]

```
//This program alternatively changes the background color
```

```
//After every click of button
```

```
import java.awt.*;
```

```
import java.awt.event.*;
```

```
import java.applet.*;
```

```
/*
```

```
<applet code="Button1" width=350 height=200>
```

```
</applet>
```

```
*/
```

```
public class Button1 extends Applet
```

```
implements ActionListener
```

```
{
```

Event Listener Interface

```
Button button=new Button("change the color");
```

```
boolean flag=true;
```

```
public void init()
```

```
{
```

```
add(button);
```

Placing the button control

```
button.addActionListener(this);
```

Invoking the button click Event

```
}
```

```
public void paint(Graphics g)
```

```
{
```

```
if(flag)
```

```
setBackground(Color.yellow);
```

```
else
```

```
setBackground(Color.red);
```

ActionListener Interface defines action. Performed method. Using this

```
}
```

```
public void actionPerformed(ActionEvent e)
```

```
{
```

```
String str=e.getActionCommand();
```

```
if(str.equals("change the color"))
```

```
{
```

```
flag=!flag;
```

```
//toggle the flag values on every click of button
```

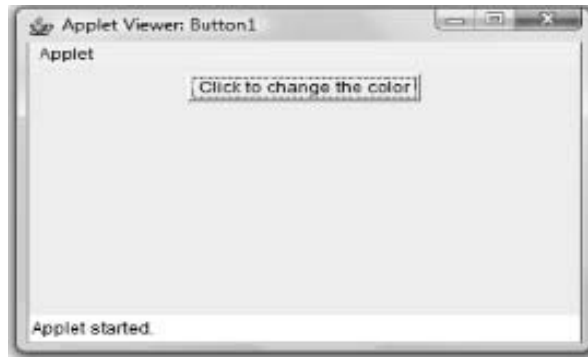
```
repaint();
```

Event gets handled here.

```
}  
}  
}
```

Output

D:\>Appletviewer Button1.java



Program Explanation

In above program,

1. When a button is clicked the action of changing the background color occurs. Hence this program must implement the **ActionListener** interface.
2. This listener class must have a method called **actionPerformed** which has a parameter an object-**ActionEvent**.
3. By invoking the method **getActionCommand** we can recognise that the event has occurred by clicking the button.

9.2 Event handlers

AU : IT : May-13, CSE : May-13, Marks 16; Dec.-18, Marks 13

9.2.1 Event Classes

- Event classes are the classes responsible for handling events in the event handling mechanism.
- The **EventObject** class is at the top of the event class hierarchy. It belongs to the **java.util** package. And other event classes are present in **java.awt.event** package.
- The **getSource()** and **toString()** are the methods of the **EventObject** class.
- There is **getId()** method belonging to **java.awt.event** package returns the **nature of the event**.
- For example, if a keyboard event occurs, you can find out whether the event was key press or key release from the event object.

- Various event classes that are defined in **java.awt.event** class are -
 1. An **ActionEvent** object is generated when a component is activated. For example if a button is pressed or a menu item is selected then this event occurs.
 2. An **AdjustmentEvent** object is generated when scrollbars are used.
 3. A **TextEvent** object is generated when text of a component or a text field is changed.
 4. A **ContainerEvent** object is generated when component are added or removed from container.
 5. A **ComponentEvent** object is generated when a component is resized, moved, hidden or made visible.
 6. An **ItemEvent** is generated when an item from a list is selected. For example a choice is made or if checkbox is selected.
 7. A **FocusEvent** object is generated when component receives keyboard focus for input.
 8. A **KeyEvent** object is generated when key on keyboard is pressed or released.
 9. A **WindowEvent** object is generated when a window activated, maximized or minimized.
 10. A **MouseEvent** object is generated when a mouse is clicked, moved, dragged, released.

ActionEvent Class

Description	When an event gets generated due to pressing of button, or by selecting menu item or by selecting an item, this event occurs.
Constructors	ActionEvent (Object <i>source</i> , int <i>id</i> , string <i>command</i> , long <i>when</i> , int <i>modifier</i>) The <i>source</i> indicates the object due to which the event is generated. The <i>id</i> which is used to identify the type of event. The <i>command</i> is a string that specifies the command that is associated with the event. The <i>when</i> denotes the time of event. The <i>modifier</i> indicates the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.
Methods	<ul style="list-style-type: none">• String getActionCommand(): This method is useful for obtaining the <i>command</i> string which is specified during the generation of event.• int getModifiers(): This method returns the value which indicates the type of key being pressed at the time of event.• long getWhen(): It returns the time at which the event occurs.
Constants	There are four constants that are used to indicate the modifier keys being pressed. These constants are CTRL_MASK, SHIFT_MASK, META_MASK and ALT_MASK.

AdjustmentEvent Class

Description	This type of event is generated when scrollbars are used.
Constructors	AdjustmentEvent (Adjustable <i>source</i> ,int <i>id</i> , int <i>type</i> , int <i>adjValue</i>) <ul style="list-style-type: none"> • The <i>source</i> indicates the object due to which the event is generated. • The <i>id</i> denotes the type of the event. • The <i>type</i> denotes the adjustment type. • The <i>adjValue</i> indicates the current value of adjustment.
Methods	<ul style="list-style-type: none"> • Adjustable getAdjustable(): It returns adjustable object when event occurs. • int getValue: It returns the integer value for the amount of adjustment. • int getAdjustmentType:It denotes the type of adjustment when the event occurs.
Constants	<ul style="list-style-type: none"> • ADJUSTMENT_FIRST: Marks the first integer value for various events. • ADJUSTMENT_LAST: Marks the last integer value for various events. • ADJUSTMENT_VALUE_CHANGED:It denotes the change that occurs when the scrollbar value changes. • UNIT_INCREMENT: It denotes that the button at start is pressed to increase the value. • UNIT_DECREMENT: It denotes that the button at the end is pressed to decrease the value. • BLOCK_INCREMENT: It denotes that the user clicks inside the scrollbar to increment the value. • BLOCK_DECREMENT: It denotes that the user clicks inside the scrollbar to decrement the value. • TRACK: It indicates when user drags the scroll.

Component Event Class

Description	This type of event is generated when a component is resized, moved, hidden or made visible.
Constructors	ComponentEvent (Component <i>Source</i> , int <i>type</i>) The <i>source</i> denotes the component that originates the event. The <i>type</i> denotes the type of event
Methods	component getComponent() : It returns the object of the component that causes the event.
Constants	<ul style="list-style-type: none"> • COMPONENT_HIDDEN: This constant indicates that the component is hidden. • COMPONENT_SHOWN: It indicates that the component is made visible. • COMPONENT_RESIZED: It indicates that the component's size is changed. • COMPONENT_MOVED: It indicates that the component is moves from its location.

ContainerEvent Class

Description	This event generated when component are added or removed from container.
Constructors	ContainerEvent (Component <i>Source</i> ,int <i>type</i> ,Component <i>c</i>) The <i>source</i> returns the reference of the container that causes an event. The <i>type</i> denotes the type of the event. The <i>c</i> denotes the component being added or removed from the container.
Methods	<ul style="list-style-type: none"> • Container getContainer(): It returns to the reference to the container in which the component is getting added or removed. • Component getChild():It returns the reference to the component which is being added or removed from the container.
Constants	<ul style="list-style-type: none"> • COMPONENT_ADDED: It represents the component which is added to the container. • COMPONENT_REMOVED:It represents the components which is removed from the container.

FocusEvent

Description	This event is generated when component receives keyboard focus for input.
Constructors	FocusEvent (Component <i>source</i> , int <i>type</i> ,Boolean <i>flag</i> ,Component <i>other</i>) The <i>source</i> is a reference to the component which generates the event. The <i>type</i> specifies the type of the event. If the focus of the event is temporary then the <i>flag</i> is true, otherwise the value of the flag is false. The focus gets changed to <i>other</i> component. This opposite component is represented by the reference <i>other</i> .
Methods	<ul style="list-style-type: none"> • Component getOppositeComponent() This method returns the opposite component. • Boolean isTemporary(): This method tells whether the focus is temporary or not
Constants	<ul style="list-style-type: none"> • FOCUS_GAINED:The component involved in the focus change • FOCUS_LOST:The other component that gains the focus will be referred.

KeyEvent Class

Description	This event is generated when key on key on the keyboard is pressed or released.
Constructors	KeyEvent (Component <i>source</i> ,int <i>type</i> ,long <i>t</i> , int <i>modifiers</i> , int <i>code</i>) The <i>source</i> is a reference to the component that generates the event. The <i>type</i> specifies the type of event The <i>t</i> denotes the system time at which the event occurs.

	<p>The <i>modifiers</i> represent the modifier keys such as ALT, CNTRL, SHIFT that are pressed when an event occurs.</p> <p>The <i>code</i> represents the virtual keycode such as VK_UP, VK_ESCAPE and so on.</p>
Methods	<ul style="list-style-type: none"> • char getKeyChar(): It returns the character when a key is pressed.
Constants	<ul style="list-style-type: none"> • KEY_PRESSED: This event is generated when the key is pressed. • KEY_RELEASED: This event is generated when the key is released • KEY_TYPED: : This event is generated when the key is typed

InputEvent Class

Description	This class is a subclass of ComponentEvent class. This class is responsible for handling the input events. Hence the subclasses of this class are KeyEvent and MouseEvent classes.
Constructors	<p>InputEvent(EventType t)</p> <p>The <i>t</i> denotes the type of the event.</p>
Methods	<ul style="list-style-type: none"> • Boolean isAltDown(): It denotes the presence of modifier ALT. • Boolean isShiftDown(): It denotes the presence of modifier Shift. • Boolean isControlDown(): It denotes the presence of modifier Control.
Constants	There are various constants that are used to indicate the modifier keys being pressed. These constants are - ALT_MASK, SHIFT_MASK, CTRL_MASK.

9.2.2 Event Listeners

- The task of handling an event is carried out by **event listener**.
- When an event occurs, first of all an event object of the appropriate type is created. This object is then passed to a **Listener**.
- A listener must **implement the interface** that has the method for event handling.
- The **java.awt.event** package contains definitions of all event classes and **listener interface**.
- **An Interface** contains constant values and method declaration.
- The methods in an interface are only declared and not implemented, i.e. the methods do not have a body.
- The interfaces are used to define behavior on occurrence of event that can be implemented by any class anywhere in the class hierarchy.

- Various event listener interfaces are -

1. ActionListener

This interface defines the method **actionPerformed()** which is invoked when an **ActionEvent** occurs. The **syntax** of this method is

```
void actionPerformed(ActionEvent act)
where act is an object of ActionEvent class.
```

2. AdjustmentListener

This interface defines the method **adjustmentValueChanged()** which is invoked when an **AdjustmentEvent** occurs. The **syntax** of this method is

```
void adjustmentValueChanged(ActionEvent act)
```

3. TextListener

It has a method **textChanged()** when a change in text area or text field occurs then this method is invoked.

```
void textChanged(TextEvent tx)
```

4. ContainerListener

When a component is added to container then this interface is required. There are two methods for this interface and those are -

```
void componentAdded(ContainerEvent ct)
void componentRemoved(ContainerEvent ct)
```

where ct represents the object of class ContainerEvent

5. ComponentListener

When component is shown, hidden, moved or resized then the corresponding methods are defined by **ContainerListener** interface. The **syntax** for the methods defined by this interface is

```
void componentShown(ComponentEvent co)
void componentHidden(ComponentEvent co)
void componentMoved(ComponentEvent co)
void componentResized(ComponentEvent co)
```

6. ItemListener

The **itemStateChanged()** is the only method defined by the **ItemListener** interface. The **syntax** is

```
void itemStateChanged(ItemEvent It)
```

7. FocusListener

By this interface the methods related to keyboard focus are used. These methods are -

```
void focusGained(FocusEvent fo)
void focusLost(FocusEvent fo)
```

8. WindowFocusListener

By this interface the methods related to windows focus are used. These methods are -

```
void windowGainedFocus(WindowEvent fo)
void windowLostFocus(WindowEvent fo)
```

These methods are called when window gains or loses focus.

9. KeyListener

This interface is defining the events such as **keyPressed()**, **keyReleased()** and **keyTyped()** are used. These methods are useful for key press, key release and when you type some characters.

```
void keyPressed(KeyEvent k)
void keyReleased(KeyEvent k)
void keyTyped(KeyEvent k)
```

10. MouseListener

This interface defines five important methods for various activities such as mouse click, press, released, entered or exited. These are

```
void mouseClicked(MouseEvent m)
void mousePressed(MouseEvent m)
void mouseReleased(MouseEvent m)
void mouseEntered(MouseEvent m)
void mouseExited(MouseEvent m)
```

11. MouseMotionListener

For handling mouse drag and mouse move events the required methods are defined by MouseMotionListener interface. These methods are

```
void mouseDragged(MouseEvent m)
void mouseMoved(MouseEvent m)
```

12. WindowsListener

There are seven methods in which are related to windows activation and deactivation.

```
void windowOpened(WindowEvent w)
void windowClosed(WindowEvent w)
void windowClosing(WindowEvent w)
void windowActivated(WindowEvent w)
void windowDeactivated(WindowEvent w)
void windowIconified(WindowEvent w)
void windowDeiconified(WindowEvent w)
```

Relationship between Event Sources and Event Listener

An event source is an object that can register listener objects. And the listener object is an instance of a class that can implement a special interface called **listener interface**

Following are the steps that denote how event handling in AWT works -

Step 1 : The **event source**(Such as button or checkbox) sends the **event objects** to all the **registered listeners** when an event occurs.

Step 2 : The listener objects will then use the information in the event object and then determine reaction.

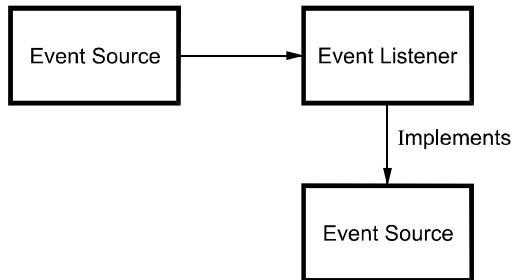


Fig. 9.2.1 Relationship between Event sources and even listener

Review Questions

1. What is event handling in java ? List out the available event classes and listener interfaces with suitable example.
2. How an application to respond to events in Java ? Write the steps and the example.
3. State and explain the basic of AWT event handling in detail.

AU : IT : May-13, Marks 16

AU : CSE : May-13, Marks 8

AU : Dec.-18, Marks 13

9.3 Adapter Classes

AU : IT : Dec.-11, CSE : Dec.-13, Marks 8

It is basically a class in Java that implements an interface with a set of dummy methods. The famous adapter classes in Java API are **WindowAdapter**, **ComponentAdapter**, **ContainerAdapter**, **FocusAdapter**, **KeyAdapter**, **MouseAdapter** and **MouseMotionAdapter**.

Whenever your class implements such interface, you have to implements all of the seven methods. **WindowAdapter** class implements **WindowListener** interface and make seven empty implementation. When you class subclass **WindowAdapter** class, you may choose the method you want without restrictions. The following give such an example.

```
public interface Windowlistener {
    public void windowClosed(WindowEvent e);
    public void windowOpened(WindowEvent e);
    public void windowIconified(WindowEvent e);
```

```

        public void windowDeiconified(WindowEvent e);
        public void windowActivated(WindowEvent e);
        public void windowDeactivated(WindowEvent e);
        public void windowClosing(WindowEvent e);
    }
    public class WindowAdapter implements WindowListner{
        public void windowClosed(WindowEvent e){}
    public void windowOpened(WindowEvent e){}
    public void windowIconified(WindowEvent e){}
    public void windowDeiconified(WindowEvent e){}
    public void windowActivated(WindowEvent e){}
    public void windowDeactivated(WindowEvent e){}
    public void windowClosing(WindowEvent e){}
    }

```

You can add a adapter class as subclass and override just the methods you need. Here is a simple Java program for handling mouse events. In this program we have used two interfaces **MouseListener** and **MouseMotionListener**. Two adapter classes are created. The **adapter1** class is for **MouseListener** interface and the **adapter2** class is for **MouseMotionListener** interface.

Java Program [Test.java]

```

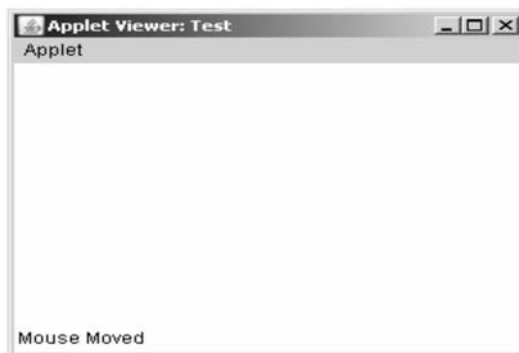
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="Test" width=300 height=200>
</applet>
*/
public class Test extends Applet
{
    public void init()
    {
        //defining the adapter classes
        //adapter1 is for MouseListener
        //adapter2 is for MouseMotionListener

        addMouseListener(new adapter1(this));
        addMouseMotionListener(new adapter2(this));
    }
}
class adapter1 extends MouseAdapter
{
    //object of main Test class
    Test obj;

```

```
public adapter1(Test obj)
{
    this.obj=obj;
}
//method belonging to MouseListener interface
public void mouseClicked(MouseEvent m)
{
    obj.showStatus("Mouse clicked");
}
}
class adapter2 extends MouseMotionAdapter
{
    Test obj;
    public adapter2(Test obj)
    {
        this.obj=obj;
    }
    //method belonging to MouseMotionListener interface
    public void mouseMoved(MouseEvent m)
    {
        obj.showStatus("Mouse Moved");
    }
}
```

Output



Note that, in the class **Test** we have written init method in which two adapter classes are created and registered as Listener using following statements

```
addMouseListener(new adapter1(this));
addMouseMotionListener(new adapter2(this));
```

Then we have defined **adapter1** class in which object for adapter class **adapter1** is initialised. Note that we have written **mouseClicked** method in adapter1 class because **mouseClicked** method is belonging to the **MouseListener** interface and the adapter class **adapter1** is for **MouseListener**. Same is true for the **adapter2** class which is an adapter class for

MouseMotionListener. Note that in the definition of **adapter2** class **mouseMoved** method is written because **mouseMoved** method is belonging to the **MouseMotionListener**. On running the above code, appropriate status will be shown on applet on encountering particular mouse event (either mouse click or mouse move).

Review Questions

1. What is an adapter class ? Explain its purpose and functionality.
2. Discuss the adapter classes using example.

AU : IT : Dec.-11, Marks 8**AU : CSE : Dec.-13, Marks 6****9.4 Actions**

Same command can be activated by multiple ways. That is, same command can be associated with a event handling function through menus, buttons, keystroke and so on. The **Action** interface is used for this purpose. This interface various methods such as -

`void actionPerformed(ActionEvent e)``void setEnabled(boolean flag)``Boolean isEnabled()``void putValue(String key_str, Object obj)``Object getValue(String key_str)``void addPropertyChangeListener(PropertyChangeListener listener)``void removePropertyChangeListener(PropertyChangeListener listener)`**9.5 Mouse Events****AU : Dec.-19, Marks 15**

While handling the mouse events we use two interfaces **MouseListener** and **MouseMotionListener**. These interfaces has certain methods such as `mouseClicked()`, `mousePressed()`, `mouseReleased()`, `mouseEntered()`, `mouseExited()` and `mouseDragged()` , `mouseMoved()` respectively. Let us see a simple Java program in which various mouse events are handled.

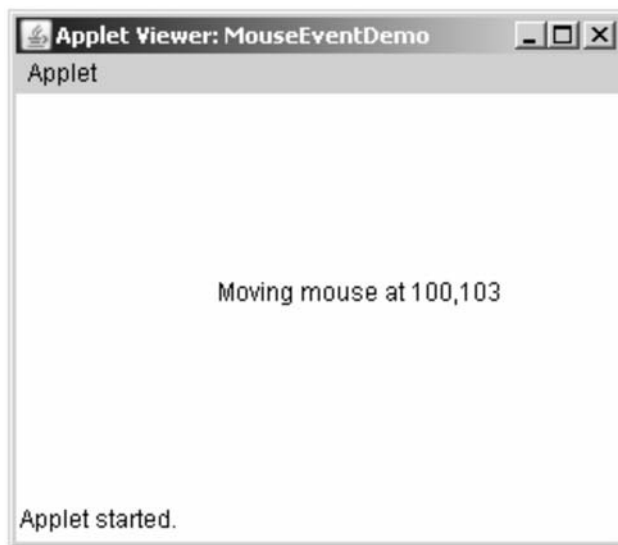
Ex. 9.5.1 : Write an applet program to handle all mouse events.**Sol. :****Java Program[MouseEventDemo.java]**

```
/*  
This is a Java program which is for handing mouse events  
*/  
import java.awt.*;  
import java.applet.*;  
import java.awt.event.*;  
/*
```

```
<applet code="MouseEventDemo" width=300 height=200>
</applet>
*/
public class MouseEventDemo extends Applet
implements MouseListener,MouseMotionListener
{
    String msg="";
    int xposition=0,yposition=0;
    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
    }
    public void mouseClicked(MouseEvent m)
    {
        xposition=m.getX();
        yposition=m.getY();
        msg="mouse Clicked";
        repaint();
    }
    public void mousePressed(MouseEvent m)
    {
        xposition=m.getX();
        yposition=m.getY();
        msg="Pressing mouse button";
        repaint();
    }
    public void mouseReleased(MouseEvent m)
    {
        xposition=m.getX();
        yposition=m.getY();
        msg="Releasing mouse button";
        repaint();
    }
    public void mouseEntered(MouseEvent m)
    {
        xposition=0;
        yposition=190;
        msg="mouse Entered";
        repaint();
    }
    public void mouseExited(MouseEvent m)
    {
        xposition=0;
        yposition=190;
    }
}
```

```
    msg="mouse Exited";
    repaint();
}
public void mouseDragged(MouseEvent m)
{
    xposition=m.getX();
    yposition=m.getY();
    msg="Dragging mouse at "+xposition+","+yposition;
    repaint();
}
public void mouseMoved(MouseEvent m)
{
    xposition=m.getX();
    yposition=m.getY();
    msg="Moving mouse at "+xposition+","+yposition;
    repaint();
}
public void paint(Graphics g)
{
    g.drawString(msg,xposition,yposition);
}
}
```

Output



In above program, we have used

```
import java.awt.event.*;
```

because various commonly used events are defined in the package **java.awt.event**.

The applet has to register itself as a listener to various events (*here the same applet acts as a event source as well as event listener*). Hence inside **init()** method applet registers itself as a listener by following statements

```
addMouseListener(this);  
addMouseMotionListener(this);
```

And then simple methods of mouse events are defined. The **getX()** and **getY()** methods return the current x and y positional values. To each of these methods an object of **MouseEvent** is passed which is shown by a variable 'm'.

Review Question

1. Discuss mouse listener and mouse motion listener. Give an example program.

AU : Dec.-19, Marks 13

9.6 AWT Event Hierarchy

AU : IT : Dec.-12, 13, Marks 16

In Java event handling is done using object oriented methodology. **EventObject** class is defined in java.util package and all the events are descendant of this class. The **AWTEvent** is a class derived from EventObject class.

The AWT Event Hierarchy is shown by following Fig. 9.6.1

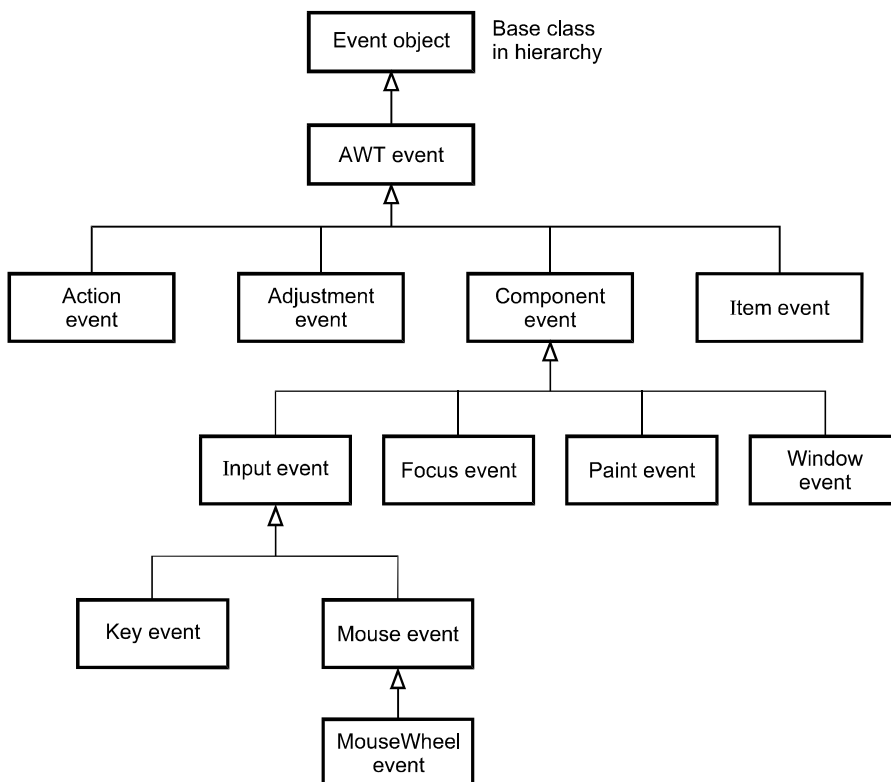


Fig. 9.6.1 AWT Event Hierarchy

Review Question

1. Explain in detail about AWT event hierarchy.

AU : IT : Dec.-12, Marks 16**Two Marks Questions with Answers****Q.1 What are the advantages of event delegation model ?****AU : IT : Dec.-10**

Ans. : Following are the advantages of event delegation model -

1. In this model, the events are handled using objects. This allows a clear separation between usage of the components and the design.
2. It accelerates the performance of the application in which multiple events are used.

Q.2 When do the following events occur ? i) AdjustmentEvent ii) ItemEvent**AU : IT : May-11**

Ans. :

(i) AdjustmentEvent : This event is generated when the scrollbars are used.

(ii) ItemEvent : This event is generated when an item from a list is selected. For example if choice is made to if a checkbox is selected.

Q.3 Which Java.util classes and interfaces support event handling ?**AU : IT : Dec.-11**

Ans. : EventObject is a class which is useful in event handling and this class belongs to java.util package.

Similarly EventListener interface of java.util package support event handling.

Q.4 Which class is the top of the AWT event hierarchy ?**AU : IT : Dec.-11**

Ans. : The **Component** class is at the top of AWT event hierarchy.

Q.5 What method is used to distinguish between single, double and triple mouse clicks ?

Ans. : public int **getClickCount()** : This methods returns the number of mouse clicks. Hence we can check -

```
if (mouseEvent.getClickCount() == 1)
{
    System.out.println("Single Click");
}
else if (mouseEvent.getClickCount() == 2)
{
    System.out.println("Double Click");
}
```

```
else if (mouseEvent.getClickCount() == 3)
{
    System.out.println("Triple Click");
}
```

Q.6 What is meant by window adapter classes ?

AU : Dec.18

Ans. : The window adapter class is for receiving the window event. The methods in this class are empty. This class exists as convenience for creating listener objects.

Q.7 Give the value for the following predefined actions. a) SMALL-ICON b) MNEMONIC-KEY

AU : CSE : Dec.-11

Ans. :

a) SMALL-ICON : The key used for storing a small Icon, such as ImageIcon. This is typically used with menus such as JMenuItem.

b) The MNEMONIC-KEY : The key used for storing an Integer that corresponds to one of the KeyEvent key codes. The value is commonly used to specify a mnemonic.

Q.8 What is the relationship between an event listener interface and an event adapter class ?

AU : IT : May-13

Ans. : The event adapter class implements the event listener interface in order to make use of methods defined by this interface.

Q.9 Name the Listener methods that must be implemented for the keyListener interface ?

AU : IT : Dec.-13

Ans. : The methods implemented for keyListener interface are -

```
void keyPressed(KeyEvent k)
void keyReleased(KeyEvent k)
void keyTyped(KeyEvent k)
```

Q.10 What is meant by event-driven programming ?

AU : CSE : Dec.-13

Ans. : Event driven programming is a technique in which using the graphical user interface user invokes some activity by causing some events. Typically mouse event and keyboard events are used to invoke some activity via GUI.



Notes

UNIT-V

10

Programming with Swing

Syllabus

Introduction to Swing - layout management - Swing Components - Text Fields, Text Areas - Buttons - Check Boxes - Radio Buttons - Lists - choices - scrollbars - Windows - Menus - Dialog Boxes.

Contents

10.1	Introduction to Swing.....	Dec.-14, May-19	Marks 16
10.2	Layout Management	CSE : May-13, Dec.-13,18,19	Marks 16
10.3	Creating Frames	CSE : Dec.-10, IT : May-13,	Marks 12
10.4	Using Swing Components		
10.5	Text Field		
10.6	Text Area		
10.7	Buttons		
10.8	Checkboxes	Dec.-18	Marks 15
10.9	Radio Buttons		
10.10	Lists		
10.11	Choices.....	May-11,19.....	Marks 10
10.12	Scrollbars		
10.13	Menus.....	May-14, 15,	Marks 16
10.14	Dialog boxes		

• 10.1 Introduction to Swing

AU : Dec.-14, Marks 16

- Swing is another approach of graphical programming in Java.
- Swing creates highly interactive GUI applications.
- It is the most flexible and robust approach.

10.1.1 Difference between AWT and Swing

Sr.No.	AWT	Swing
1.	The Abstract Window ToolKit is a heavy weight component because every graphical unit will invoke the native methods.	The Swing is a light weight component because it's the responsibility of JVM to invoke the native methods.
2.	The look and feel of AWT depends upon platform.	As Swing is based on Model View Controller pattern, the look and feel of swing components is independent of hardware and the operating system.
3.	AWT occupies more memory space.	Swing occupies less memory space.
4.	AWT is less powerful than Swing.	Swing is extension to AWT and many drawbacks of AWT are removed in Swing.

10.1.2 Limitations of AWT

Following are some limitations of AWT -

1. AWT supports limited number of GUI components.
2. The components defined by the AWT are heavy weight components.
3. The behavior of AWT components varies when the container operating system changes.
4. The AWT components are developed by using the platform specific code.
5. The AWT component is converted by the native code of the operating system.

10.1.3 Swing Components

The most commonly used component classes are -

Component	Purpose
AbstractButton	Abstract class for the buttons
ButtonGroup	It creates the group of buttons so that they behave in mutually exclusive manner
JApplet	The swing applet class
JButton	The swing button class
JCheckBox	The swing check box
JComboBox	The swing combo box
JLabel	The swing label component
JRadioButton	The radio button component
JTextArea	The text area component
TextField	The text field component
JSlider	The slider component

We will learn and understand these components with the help of programming examples .

10.1.3.1 Swing Class Component Hierarchy

In order to display any JComponent on the GUI, it is necessary to add this component to the container first. If you do not add these components to container then it will not be displayed on the GUI. The swing class component hierarchy is as shown by following Fig. 10.1.1.

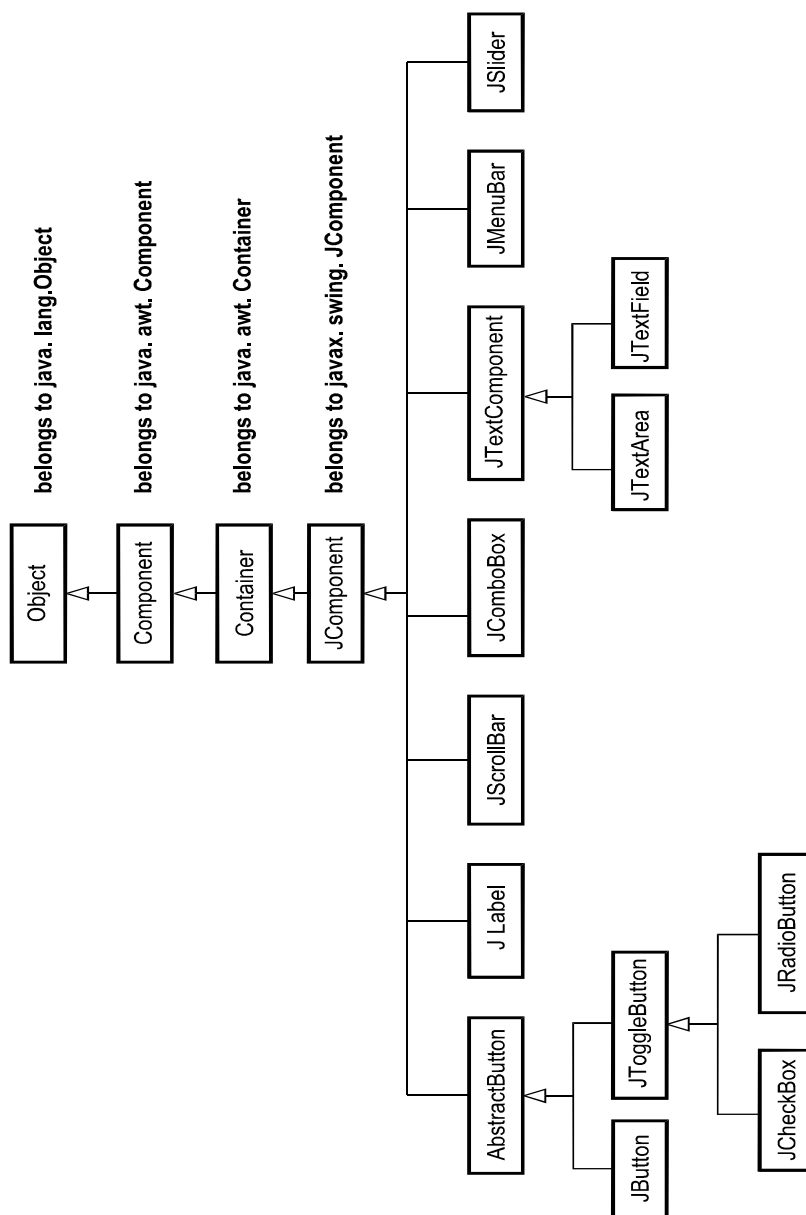


Fig. 10.1.1 Swing class component hierarchy

There are two important features of swing components - Firstly, all the component classes begin with the letter J and secondly all these GUI components are descendant of JComponent class. Let us now learn how to place these components on the GUI.

Review Questions

1. List and briefly discuss the swing components in Java.
2. State the difference between AWT and swing.

AU : Dec.-14, Marks 16

AU : May-19, Marks 4

10.2 Layout Management

AU : CSE : May-13, Dec.-13,19, Marks 16; Dec.-18, Marks 13

Definition :

- A **Layout manager** is an interface which automatically arranges the controls on the screen.
- Thus using layout manager the **symmetric** and **systematic arrangement** of the **controls** is possible. In this section we will discuss following layout managers.

10.2.1 FlowLayout

- FlowLayout manager is the simplest Layout manager.
- Using this Layout manager components are arranged from top left corner lying down from left to right and top to bottom.
- Between each component there is some space left.
- The **syntax** of FlowLayout manager is as given below -
`FlowLayout(int alignment)`

Where *alignment* denotes the alignment of the components on the applet windows. The alignment can be denoted as :

`FlowLayout.LEFT`

`FlowLayout.RIGHT`

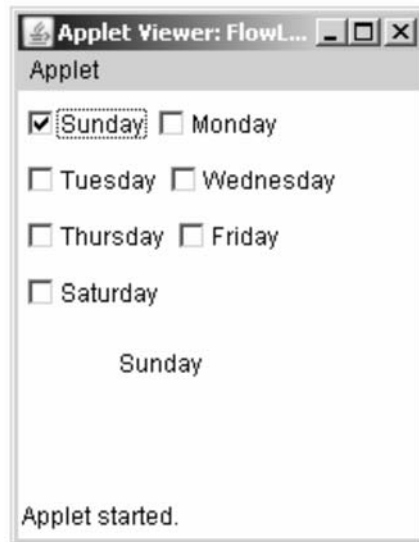
`FlowLayout.CENTER`

- Here is a Java program which makes use of seven checkboxes which are aligned on the applet window using FlowLayout manager.

Java Program[ItemListener.java]

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="FlowLDemo" width=200 height=200>
</applet>
*/
public class FlowLDemo extends Applet
implements ItemListener
{
    String msg=" ";
    Checkbox box1=new Checkbox("Sunday");
    Checkbox box2=new Checkbox("Monday");
    Checkbox box3=new Checkbox("Tuesday");
    Checkbox box4=new Checkbox("Wednesday");
    Checkbox box5=new Checkbox("Thursday");
    Checkbox box6=new Checkbox("Friday");
```

```
Checkbox box7=new Checkbox("Saturday");
public void init()
{
    //creating FlowLayout manager
    setLayout(new FlowLayout(FlowLayout.LEFT));
    //adding the components with Left alignment
    add(box1);
    add(box2);
    add(box3);
    add(box4);
    add(box5);
    add(box6);
    add(box7);
    //registering the checkboxes to EventListener
    box1.addItemListener(this);
    box2.addItemListener(this);
    box3.addItemListener(this);
    box4.addItemListener(this);
    box5.addItemListener(this);
    box6.addItemListener(this);
    box7.addItemListener(this);
}
public void paint(Graphics g)
{
    //if box1 checkbox is clicked
    if(box1.getState())
        msg="Sunday ";//then print the corresponding day
    if(box2.getState())
        msg="Monday ";
    if(box3.getState())
        msg="Tuesday ";
    if(box4.getState())
        msg="Wednesday ";
    if(box5.getState())
        msg="Thursday ";
    if(box6.getState())
        msg="Friday ";
    if(box7.getState())
        msg="Saturday ";
    g.drawString(msg,50,140);
}
public void itemStateChanged(ItemEvent e)
{
    repaint();
}
}
```

Output**Program Explanation :**

In above program, since we are using Check box control the **ItemListener** interface is used. And for registering these controls to receive the events we have written following lines in the **init** method.

```
box1.addItemListener(this);  
box2.addItemListener(this);  
box3.addItemListener(this);  
box4.addItemListener(this);  
box5.addItemListener(this);  
box6.addItemListener(this);  
box7.addItemListener(this);
```

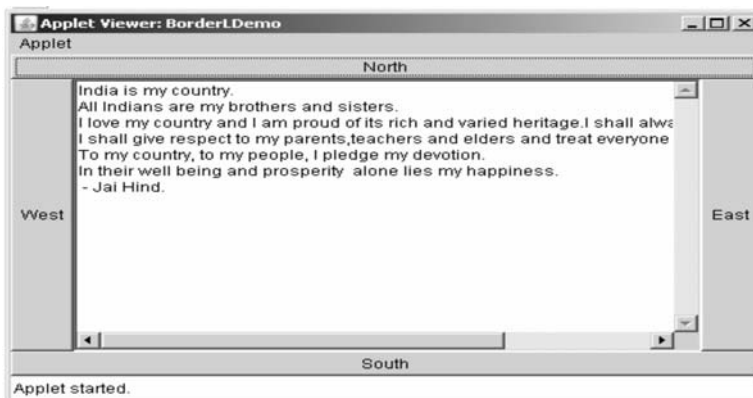
We have written a method **public void itemStateChanged (ItemEvent e)** because **ItemListener** interface is used.

10.2.2 BorderLayout

- In BorderLayout there are **four components** at the four sides and one component occupying large area at the centre.
- The central area is called CENTER and the components forming four sides are called LEFT, RIGHT, TOP and BOTTOM.
- Following program consists of one big message stored in variable **msg** which is to be displayed at the centre. And in the **init** method it shows that four sides are occupied by the **Buttons**.

Java Program[BorderLDemo.java]

```
import java.applet.*;
import java.awt.*;
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
String msg="India is my country.\n"+"All Indians are my
    brothers and sisters.\n"+
    "I love my country and I am proud of its rich
    and varied heritage."+
    "I shall always strive to be worthy of it.\n"+
    "I shall give respect to my parents,teachers and
    elders and treat everyone with courtesy.\n"+
    "To my country, to my people, I pledge my
    devotion.\n"+
    "In their well being and prosperity alone lies my
    happiness.\n"+
    "- Jai Hind.";
public void init()
{
    setLayout(new BorderLayout());
    add(new Button("North"),BorderLayout.NORTH);
    add(new Button("South"),BorderLayout.SOUTH);
    add(new Button("East"),BorderLayout.EAST);
    add(new Button("West"),BorderLayout.WEST);
    add(new TextArea(msg),BorderLayout.CENTER);
}
}
```

Output

Program Explanation :

In the above program, we have created object for BorderLayout manager using `setLayout(new BorderLayout());`

Then using

`BORDER.NORTH`

`BORDER.SOUTH`

`BORDER.EAST`

`BORDER.WEST`

- The four sides are set with the help of **Button** control. The central large area is formed using **TextArea** control which is called as `BORDER.CENTER`.
- The concept of BorderLayout can then be clearly understood with the help of above given output.
- In this Layout manager, we can add one more method called **getInsets()**. This method allows us to leave some space between underlying window on the applet and Layout manager.
- We have used this method in the following program. The syntax of **Insets** method is `Insets(int top,int left,int bottom,int right)`

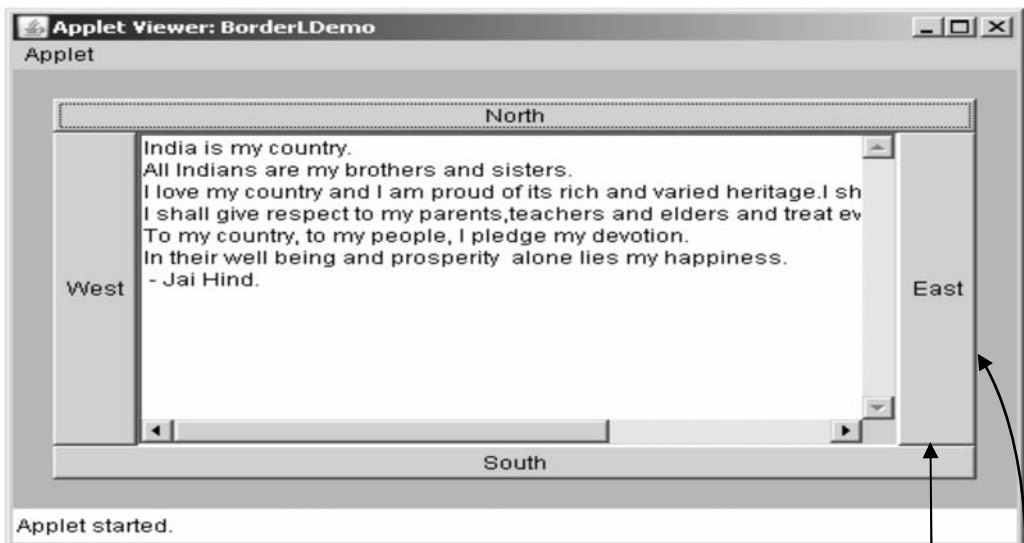
The *top, left, bottom* and *right* parameters specify the amount of space to be left.

Java Program

```
import java.applet.*;
import java.awt.*;
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
String msg="India is my country.\n"+"All Indians are my
    brothers and sisters.\n"+
        "I love my country and I am proud of its rich
    and varied heritage."+
        "I shall always strive to be worthy of it.\n"+
        "I shall give respect to my parents,teachers
    and elders and treat everyone with
    courtesy.\n"+
        "To my country, to my people, I pledge my
    devotion.\n"+
        "In their well being and prosperity alone lies
    my happiness.\n"+
```

```
" - Jai Hind.";
public void init()
{
    setBackground(Color.green);
    setLayout(new BorderLayout());
    add(new Button("North"),BorderLayout.NORTH);
    add(new Button("South"),BorderLayout.SOUTH);
    add(new Button("East"),BorderLayout.EAST);
    add(new Button("West"),BorderLayout.WEST);
    add(new TextArea(msg),BorderLayout.CENTER);
}
public Insets getInsets()
{
    return new Insets(20,20,20,20);
}
}
```

Output



This Space is left in between

Ex. 10.2.1 : Write a Java program which create border layout and adds two text boxes to it.

Sol. :

BorderLDemo.java

```
import java.applet.*;
import java.awt.*;
```

```
import java.util.*;
/*
<applet code="BorderLDemo" width=500 height=300>
</applet>
*/
public class BorderLDemo extends Applet
{
    public void init()
    {
        setLayout(new BorderLayout());
        add(new Button("Center"),BorderLayout.CENTER);
        add(new Button("East"),BorderLayout.EAST);
        add(new Button("West"),BorderLayout.WEST);
        add(new TextField("Technical"),BorderLayout.NORTH);
        add(new TextField("Books"),BorderLayout.SOUTH);
    }
}
```

Output



10.2.3 GridLayout

- GridLayout is a Layout manager used to arrange the components in a grid. The syntax of GridLayout manager is
`GridLayout(int n,int m)`

Where n represents total number of rows and m represents total number of columns.

- In the following program we have arranged **Buttons** in a grid form.

Java Program[GridLDemo.java]

```
import java.awt.*;
import java.applet.*;
/*
<applet code="GridLDemo" width=400 height=400>
</applet>
*/
public class GridLDemo extends Applet
{
    int n=4,m=3;
    public void init()
    {
        setLayout(new GridLayout(n,n));
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                switch(i)
                {
                    case 0:if(j==0) //Button[0,0]
                        add(new Button("Red"));
                        else if(j==1) //Button[0,1]
                            add(new Button("Green"));
                        else if(j==2) //Button[0,2]
                            add(new Button("Blue"));
                        break;
                    case 1:if(j==0) //Button[1,0]
                        add(new Button("Orange"));
                        else if(j==1) //Button[1,1]
                            add(new Button("Pink"));
                        else if(j==2) //Button[1,2]
                            add(new Button("Magenta"));
                        break;
                    case 2:if(j==0) //Button[2,0]
                        add(new Button("Cyan"));
                        else if(j==1) //Button[2,1]
                            add(new Button("Gray"));
                        else if(j==2) //Button[2,2]
                            add(new Button("Yellow"));
                        break;
                    case 3:if(j==0) //Button[3,0]
                        add(new Button("Black"));
                        else if(j==1) //Button[3,1]
                            add(new Button("White"));
                        else if(j==2) //Button[3,2]
```

```
        add(new Button("LightGray"));
        break;
    } //end of switch
} //end of inner for
} //end of outer for
} // end of function init
} //end for class
```

Output



10.2.4 CardLayout

- Sometimes we want to perform various sets of graphical controls at a time then CardLayout is used.
- Thus CardLayout manager allows us to have more than one layouts on the applet.
- The CardLayout is conceptually thought as a collection of cards lying on a **panel**.
- We have to follow following steps -

Step 1 : We have to create two objects

1. Panel object
2. CardLayout object

Step 2 : Then we have to add the cards on the panel using **add()** method. For example -
`panel_obj.setLayout(layout_obj);`

where *panel_obj* is an object of panel and *layout_obj* is an object of CardLayout

Step 3 : Finally we have to add the object of panel to main applet. For example -
`add(panel_obj);`

These all stages seem to be complicated. Hence let us understand following program which implements CardLayout.

Java Program[cardDemo.java]

```
//This program demonstrates cardLayout
//The dynamic selection of fruit/flower/colour can be made
//using cardlayout component
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="cardDemo" width=300 height=100>
</applet>
*/
public class cardDemo extends Applet implements
ActionListener,MouseListener
{
    Checkbox mango,apple,rose,lotus,Red,Green;
    Panel panel_obj;
    CardLayout layout_obj;
    Button fruit,flower,colour;
    public void init()
    {
        fruit=new Button("Fruit");
        flower=new Button("Flower");
        colour=new Button("Colour");
        //adding the button controls
        add(fruit);
        add(flower);
        add(colour);
        //getting object of Cardlayout
        layout_obj=new CardLayout();
        //getting object of Panel
        panel_obj=new Panel();
        panel_obj.setLayout(layout_obj);
```

```
//adding checkbox controls for fruits
mango=new Checkbox("Mango");
apple=new Checkbox("Apple");
//adding checkbox controls for flowers
rose=new Checkbox("Rose");
lotus=new Checkbox("Lotus");
//adding checkbox controls for colors
Red=new Checkbox("Red");
Green=new Checkbox("Green");

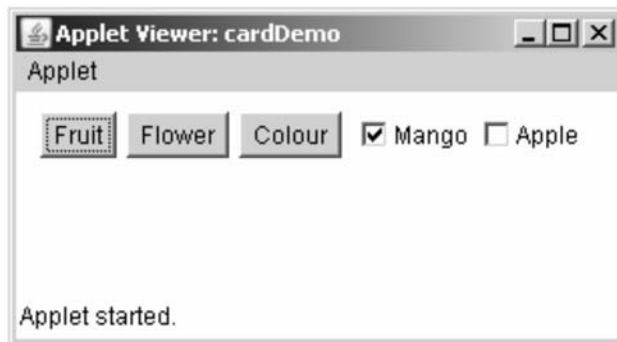
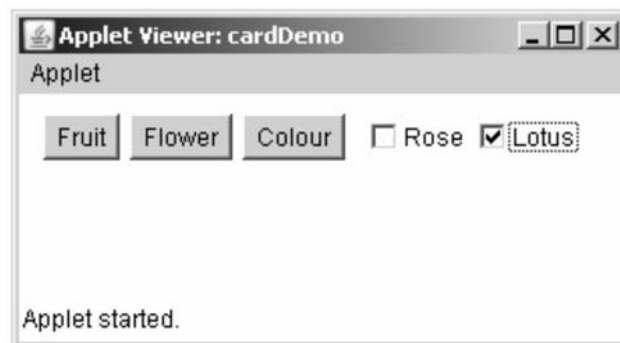
Panel fruit_pan=new Panel();
fruit_pan.add(mango);
fruit_pan.add(apple);

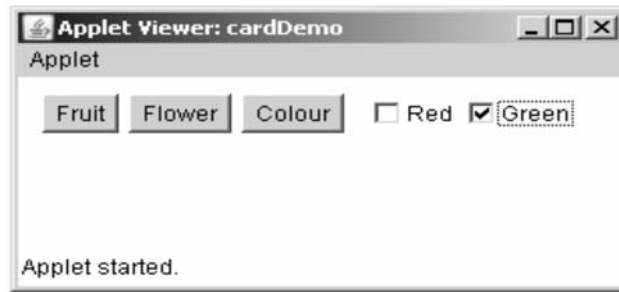
Panel flower_pan=new Panel();
flower_pan.add(rose);
flower_pan.add(lotus);

Panel colour_pan=new Panel();
colour_pan.add(Red);
colour_pan.add(Green);

panel_obj.add(fruit_pan,"Fruit");
panel_obj.add(flower_pan,"Flower");
panel_obj.add(colour_pan,"Colour");
add(panel_obj);
//register the components to event listener
fruit.addActionListener(this);
flower.addActionListener(this);
colour.addActionListener(this);
addMouseListener(this);
}
//following empty methods are necessary for mouse events
public void mousePressed(MouseEvent m)
{
    layout_obj.next(panel_obj);
}
public void mouseClicked(MouseEvent m)
{
}
public void mouseEntered(MouseEvent m)
{
}
public void mouseExited(MouseEvent m)
{
}
```

```
public void mouseReleased(MouseEvent m)
{
}
public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==fruit)
    {
        layout_obj.show(panel_obj,"Fruit");
    }
    else if(e.getSource()==flower)
    {
        layout_obj.show(panel_obj,"Flower");
    }
    else if(e.getSource()==colour)
    {
        layout_obj.show(panel_obj,"Colour");
    }
}
} //end for actionPerformed method
} //end of class
```

Output(Run 1)**Output(Run 2)**

Output (Run 3)**Program Explanation :**

- It is clear from the output that we can have a combination of various components lying on the same applet and can be invoked as per need. That mean if we click on *Fruit* button then we should get two checkboxes namely : *Mango* and *Apple*.
- If we click on *Flower* button then we should get two checkboxes namely : *Rose* and *Lotus*. Similarly, if we click on *Colour* button we should get two checkboxes namely : *Red* and *Green*.
- In above program, we have used to event listener interfaces : **ActionListener** and **MouseListener**.
- We have created **panel object** and **Layout object**.
- In the **init** method we have first created and added the **Button** controls. Then **CardLayout** is placed on the panel.
- The panel is then set for the applet window.
- Various components such as **checkboxes** and **Buttons** are added to the panel.
- In order to understand mouse events some necessary empty methods are written.
- In the **actionPerformed()** method, on the click of **Fruit** button, two corresponding check boxes are shown. Same is true for **Flower** and **Colour** buttons.

10.2.5 GridBagLayout

- The GridBagLayout is the most flexible and complex layout manager.
- The GridBagLayout manager places the components in rows and columns allowing the components to occupy multiple rows and columns. This is called **display area**.
- GridBagLayout performs three functions using values from the **GridBagConstraints** parameter in the add() method.
 1. Grid position, width and height describe the display area using **gridx**, **gridy**, **Gridwidth** and **gridheight** values.

2. Position within the display area using fill, **ipadx** and **ipady**.
3. Identifying rows and columns which receive extra space on expansion using **weightx**, and **weighty**.

- The layout can be set as follows

```
Container pane=frame.getContentPane();  
pane.setLayout(new GridBagLayout());
```

- The component can be added as

```
pane.add(component,constraintObject);
```

//pane is a container pane
- The following values are then set -
 - **gridx** and **gridy** : These values denote the integer column and row value of the component.
 - **gridwidth** and **gridheight** : They denote the number of columns and rows the component occupies.
 - **weightx** and **weighty** : They denote the extra space occupied by the component horizontally or vertically when the output window is resized.
 - **fill** : The fill value denotes how the component should expand within the display area. Typical values are -

```
GridBagConstraints.NONE    // Can not expand (Default)  
GridBagConstraints.VERTICAL // Expand vertically  
GridBagConstraints.HORIZONTAL // Expand horizontally  
GridBagConstraints.BOTH    // Expand vertically and horizontally
```
 - **ipadx** and **ipady** : These values denote increase and decrease in horizontal or vertical preferred size of the component. Default value is 0.

Ex. 10.2.2 : Write a simple Java program that illustrates the use of GridBagLayout.

Sol. :

```
import java.awt.*;  
import java.applet.*;  
/*  
<applet code="GridBagLayoutDemo" width=400 height=400>  
</applet>  
*/  
public class GridBagLayoutDemo extends Applet  
{  
    public void init()  
    {  
        Button B;  
        setLayout(new GridBagLayout());  
        GridBagConstraints gBC = new GridBagConstraints();  
        gBC.fill = GridBagConstraints.HORIZONTAL; //placing the components horizontally  
        B = new Button("Button 1"); //first component
```

```
gBC.weightx = 0.5;  
gBC.gridx = 0;  
gBC.gridy = 0;  
add(B, gBC);
```

```
B = new Button("Button 2");//second component  
gBC.gridx = 2;  
gBC.gridy = 0;  
    add(B, gBC);
```

```
B = new Button("Button 3"); //third component  
gBC.ipady = 40;    //This component is broad  
gBC.weightx = 0.0;  
gBC.gridwidth = 3;  
gBC.gridx = 0;  
gBC.gridy = 1;  
add(B, gBC);
```

```
TextField T = new TextField("Hello Friends!!!");//forth component  
gBC.ipady = 0;  
gBC.weightx = 0.0;  
gBC.gridx = 1;  
gBC.gridwidth = 2;  
gBC.gridy = 2;  
T.setEditable(false);//text field is not editable  
add(T, gBC);
```

```
}  
}
```

Output



Before Expanding



After Expanding

Review Questions

1. What is the function of layout manager ? Describe in detail about the different layout in Java GUI.

AU : CSE, May-13, Marks 8

2. What is layout management ? State the various types of layout supported by JAVA. Which layout is default one ? Discuss the components of swing.

AU : CSE, Dec.-13, Marks 16

3. Describe in detail about the different layout in Java GUI. Which layout is the default one ?

AU : Dec.-18, 19, Marks 13

10.3 Creating Frames

AU : CSE : Dec.-10, IT : May-13, Marks 12

- A Frame is a top-level window with a title and a border. The Frames in java works like the main window where the components like JButton, JLabel, JComboBox and so on can be placed. In Java swing the top-level windows are represented by the **JFrame** class. For creating the frame following statement can be used –

```
JFrame f=new JFrame("My First Frame Program");
```

object of JFrame class

Title to the Frame window.

- The frames are not visible initially, hence we have to make them visible by `f.setVisible(true);`
- The close button of the frame by default performs the hide operation for the JFrame. Hence we have to change this behavior to window close operation by setting the `setDefaultCloseOperation()` to **EXIT_ON_CLOSE** value.
- Any suitable size of the frame can be set by **setSize(int height,int width)** function.
- In the following program, we have created a frame on which two buttons and one textfield components are placed. The image icons are associated with the buttons. If the button having “apple” image is pressed then the string “Apple” will be displayed in the text field and if the button having “orange” image is pressed then the string “Orange” will be displayed in the text field.

Java Program[FrameProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrameProg extends JFrame
implements ActionListener
{
    JTextField T;
    public static void main(String[] args)
```

```
{
    new FrameProg();
}
FrameProg()
{
    JFrame f=new JFrame("Frame Demo");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setVisible(true);
    f.setSize(300,500);
    Container content=f.getContentPane();
    content.setLayout(new FlowLayout());
    content.add(new JLabel("Click on Any of the button"));

    ImageIcon apple=new ImageIcon("apple.gif");
    JButton B1=new JButton("Button1",apple);
    B1.setActionCommand("Apple");
    B1.addActionListener(this);
    content.add(B1);

    ImageIcon orange=new ImageIcon("orange.gif");
    JButton B2=new JButton("Button2",orange);
    B2.setActionCommand("Orange");
    B2.addActionListener(this);
    B2.setIcon(orange);
    content.add(B2);

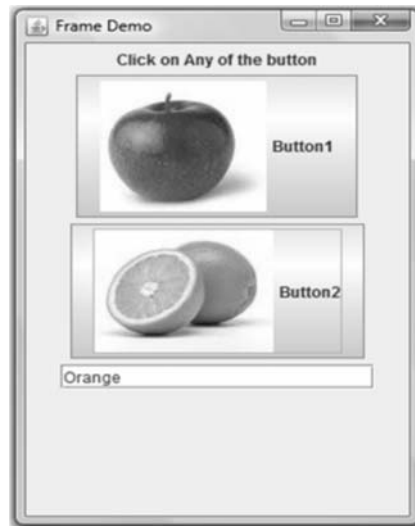
    T=new JTextField(20);
    content.add(T);

}
public void actionPerformed(ActionEvent e)
{
    T.setText(e.getActionCommand());
}
}
```

Output

F:\SwingProg>javac FrameProg.java

F:\SwingProg>java FrameProg



Program Explanation

To place various components on the Frame we will follow following steps -

Step 1 : Create a **JFrame** object. Set its visibility to true. Also set the default close operation to EXIT so that on closing the frame window the prompt can be displayed.

Step 2 : When ever we want to place some controls such as buttons, labels, textfields and so on the frame window then first of all the object for the **Container** class must be created using the **getContentPane** method. Then using **add** method these controls can be placed on the window.

Step 3 : In above program, two **JButtons** and one **JTextField** is placed using the **add** method.

Step 4 : The command strings "Apple" and "Orange" is associated with the buttons using **setActionCommand** method.

Step 5 : The button click is handled by the event **ActionListener**. Hence on the **actionPerformed** method, we are retrieving the command string and displaying it in the text field. As a result, if we click the *apple* button the string "Apple" will be displayed in the text field and if we click the *orange* button the string "Orange" will be displayed in the text field.

Ex. 10.3.1 : Write a JFrame with a hello world program.

Sol. :

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;
```

```
public class FrameProg extends JFrame
{
    public static void main(String[] args)
    {
        new FrameProg();
    }
    FrameProg()
    {
        JFrame f=new JFrame("Frame Demo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
        f.setSize(300,300);
        Container content=f.getContentPane();
        content.setLayout(new FlowLayout());
        content.add(new JLabel("Hello World"));
    }
}
```

Output



Ex. 10.3.2 : Write a program to create product enquiry form using frames.**AU : CSE : Dec.-10, Marks 12****Sol. :**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class FrameProg extends JFrame
{
    public static void main(String[] args)
    {
        new FrameProg();
    }
    FrameProg()
    {
        JFrame f=new JFrame("Enquiry Form");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
        f.setSize(800,300);
        Container content=f.getContentPane();
        content.setLayout(new FlowLayout(FlowLayout.CENTER));
        content.add(new JLabel("Name:"));
        content.add(new JTextField(10));
        content.add(new JLabel("Address:"));
        content.add(new JTextField(20));
        content.add(new JLabel("Product:"));
        JComboBox co=new JComboBox();
        co.addItem("Mobile Phones");
        co.addItem("Laptops");
        co.addItem("ipods");
        co.addItem("Tablet PC");
        content.add(co);
        JButton b=new JButton("Submit");
        content.add(b);
    }
}
```

Ex. 10.3.3 : How will you display an image on the frame in a window using java.**AU : IT : May-13, Marks 8****Sol. :**

```
import javax.swing.*;
import java.awt.*;
public class ImgFrameDemo extends JFrame
{
    public void display()
```

```
{  
  
    JFrame frame = new JFrame("Displaying Image On Frame");  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.setSize(400, 400);  
    frame.setResizable(false);  
    frame.setLocationRelativeTo(null);  
    // Inserts the image icon  
  
    ImageIcon image = new ImageIcon("img1.jpg");  
    JLabel label1 = new JLabel(" ", image,  
JLabel.CENTER);  
    frame.getContentPane().add(label1);  
    frame.validate();  
    frame.setVisible(true);  
}  
public static void main(String[] args)  
{  
    ImgFrameDemo obj = new ImgFrameDemo();  
    obj.display();  
}  
}
```



10.4 Using Swing Components

- Swing is a large system. It is one part of Java Foundation Classes(JFC).
- It has got good **look** and **feel**.
- There is a rich collection of swing components that can be used to provide the advanced user interface.
- Various classes that can be used for placing the components are JApplet, JLabel, JTextField, JButton, JCheckBox, JTree, JTabbedPane and so on.

10.4.1 Label and Image Icon

- The icons are encapsulated by **ImageIcon** class. The constructors are -
`ImageIcon(string fname)`
`ImageIcon(URL url)`

The *fname* denotes the name of the file which contains the icon. The *url* denotes the resource of image.

- The **JLabel** is a component for placing the label component. The **JLabel** is a subclass of **JComponent** class. The syntax for the JLabel is -

```
JLabel(Icon ic);  
Label(String s);
```

JLabel(String s, Icon ic, int alignment)

- The alignment can be LEFT, RIGHT, CENTER, LEADING and TRADING. The icons and text methods associated with the label are -

Icon getIcon()

void setIcon(Icon ic) ← ic represents the icon file

String getText();

void setText(String s); ← s represents the string

- Following simple Java Program illustrates the use of JLabel component -

Java Program[LabelProg.java]

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="LabelProg" width=300 height=200>
</applet>
*/
public class LabelProg extends JApplet
{
    public void init()
    {
        Container contentPane=getContentPane();
        ImageIcon i=new ImageIcon("innocence.gif");
        JLabel L1=new JLabel("Innocence",i,JLabel.LEFT);
        contentPane.add(L1);
    }
}
```

Output



10.5 Text Field

- The **JTextField** is extended from the **JComponent** class. The **JTextField** allows us to add a single line text.
- The syntax of using **JTextField** is -
`JTextField();`
`JTextField(int col_val);`
`JTextField(String s,int col_val);`
`JTextField(String s);`
- The program making use of **TextField** is as given below -

Java Program[TxtFieldProg.java]

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="TxtFieldProg" width=300 height=200>
</applet>
*/
public class TxtFieldProg extends JApplet
{
    public void init()
    {
        JTextField T;
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        T=new JTextField("Hello",20);
        contentPane.add(T);
        T=new JTextField(20);
        contentPane.add(T);
    }
}
```

- For getting the output of the above program following commands can be given on the command prompt -

```
F:\SwingProg>javac TxtFieldProg.java
```

```
F:\SwingProg>AppletViewer TxtFieldProg.java
```

The Applet Viewer will display the GUI as follows -

Output



Program Explanation

To place the control textfield on the GUI we have followed following steps -

1. Using the `getContentPane` method an object for the `Container` class is created. This object is taken in the variable `contentPane`.
2. The Layout is set using the `contentPane` object by the statement
`contentPane.setLayout(new FlowLayout());`

The default layout is `FlowLayout` but you can set other layout managers such as `GridLayout`, `BorderLayout` and so on.

3. After setting the layout manager, the `TextField` component can be created and placed using `add` method. In above program, we have created two text fields. In the first text field, the string "Hello" is already written during its creation but the second text field is kept blank and some string can be written into it during the execution.

10.6 Text Area

The `JTextArea` is a GUI control which allows us to write multi-line text. We can set the desired number of rows and number of columns so that a long big message can be written using this control.

The syntax for `JTextArea` is

```
JTextArea(String str,int rows,int cols);
```

In the following Java program, there are two GUI controls on the window - the text area and the push button. The idea is that: just type something within the text area and then click the push button. Whatever text you type will get displayed on the console window.

Java Program[TextAreaProg.java]

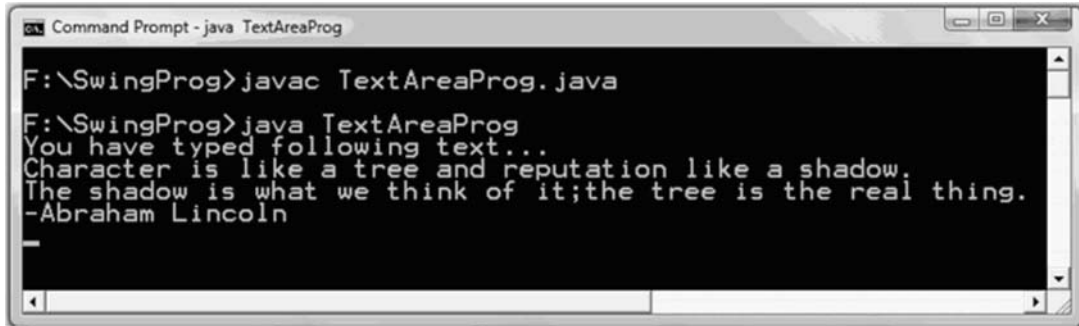
```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
class TextAreaProg
```

```
implements ActionListener
{
    JTextArea TA;
    public static void main(String[] args)
    {
        new TextAreaProg(); //invoke the function for demonstrating TextArea control
    }
    TextAreaProg() //definition of the function
    {
        JFrame f=new JFrame(); //create a Frame window
        Container content=f.getContentPane();//get the Container class object
        content.setLayout(new FlowLayout());//set the layout manager
        JLabel L=new JLabel("Enter some text here...");//create a Label control
        content.add(L); //using add method add the label on the GUI
        TA=new JTextArea("",10,20); //create a TextArea control with rows=10 and col=20
        content.add(TA);//using add method add the TextArea control on GUI
        JButton B=new JButton("Submit");//create a push button control
        content.add(B);//place this control on the GUI using add method
        B.addActionListener(this);//invoke action listener for button click event
        f.setSize(300,300);//set the size of the frame window
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);//exit on closing
        f.setVisible(true);//set the visibility of frame as true
    }
    public void actionPerformed(ActionEvent e)//handling the event when button is clicked
    {
        String s=TA.getText();//get the string typed in textarea
        System.out.println("You have typed following text...\n"+s);//display it on the consol
    }
}
```

Output



Now if we click **Submit** button then on the command-prompt window we can see following text-



```

Command Prompt - java TextAreaProg
F:\SwingProg>javac TextAreaProg.java
F:\SwingProg>java TextAreaProg
You have typed following text...
Character is like a tree and reputation like a shadow.
The shadow is what we think of it;the tree is the real thing.
-Abraham Lincoln

```

10.7 Buttons

- The swing push button is denoted by using **JButton** class.
- The swing button class provides the facilities that can not be provided by the applet button class. For instance you can associate some image file with the button.
- The swing button classes are subclasses of **AbstractButton** class.
- The **AbstractButton** class generates action events when they are pressed. These events can be associated with the Push buttons.
- We can associate an icon and/or string with the **JButton** class. The syntax of JButton is
 JButton(Icon ic);
 JButton(String s);
 JButton(String s,Icon ic);
- Following program illustrates the use of Icon and the string for the JButton class -

Java Program[ButtonProg.java]

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="ButtonProg" width=500 height=300>
</applet>
*/
public class ButtonProg extends JApplet//inherited from JApplet
implements ActionListener
{
    JTextField T;
    public void init()
    {
        Container contentPane=getContentPane();

```

Creating object of
Container class

```
contentPane.setLayout(new FlowLayout());
ImageIcon apple=new ImageIcon("apple.gif");
JButton B1=new JButton(apple);
B1.setActionCommand("Apple");
B1.addActionListener(this);
contentPane.add(B1);//adding button on GUI
```

Creating Button
component

```
ImageIcon orange=new ImageIcon("orange.gif"); //Creating image icon
JButton B2=new JButton(orange); //associating image with button
B2.setActionCommand("Orange");
B2.addActionListener(this); //button pressed event
contentPane.add(B2);
```

```
ImageIcon grapes=new ImageIcon("grapes.gif");
JButton B3=new JButton(grapes);
B3.setActionCommand("Grapes");
B3.addActionListener(this);
contentPane.add(B3);
T=new JTextField(20);
contentPane.add(T);//placing the textfield on the GUI
```

```
}
public void actionPerformed(ActionEvent e)
{
    T.setText(e.getActionCommand()); //retrieving the text associated with button
}
}
```

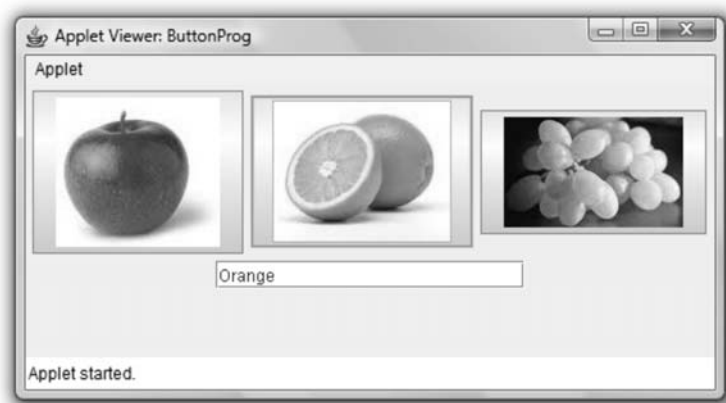
For getting the output open the **command-prompt** and give the following commands -

```
F:\SwingProg>javac ButtonProg.java
```

```
F:\SwingProg>AppletViewer ButtonProg.java
```

and you will get the applet as follows -

Output



Program Explanation

The above program, is inherited from the **JApplet** class. In the init method, we have written the code -

1. Create a container object by using the **getContentPane** method. This object is now in the variable **contentPane**.
2. A layout manager such as **FlowLayout** is used to set the layout of the GUI.
3. The button components are then placed on the GUI using the **add** method.
4. These push buttons are associated with some images using **ImageIcon** class. To this **ImageIcon** class appropriate gif file is passed as an argument. Thus corresponding image gets associated with each corresponding push button.
5. We have associated an **ActionListener** event with this program. This is necessary to handle the events when a push button gets pressed. We have associated some command string when the particular button is pressed. This can be done using **setActionCommand** method. The event handler can be invoked using the methods **addActionListener(this)**. When this a call to this method is given the control goes to the function **actionPerformed**. In this method we are simply displaying the appropriate command string in the textbox.

10.8 Checkboxes

AU : Dec.-18, Marks 15

- The Check Box is also implementation of **AbstractButton** class. But the immediate superclass of **JCheckBox** is **JToggleButton**.
- The **JCheckBox** supports two states true or false.
- We can associate an icon, string or the state with the checkboxes. The syntax for the Check Box will be -

```
JCheckBox(Icon ic);  
JCheckBox(Icon ic, boolean state);  
JCheckBox(String s);  
JCheckBox(String s,boolean state);  
JCheckBox(String s,Icon ic,boolean state);
```

- Following program illustrates the use of check box -

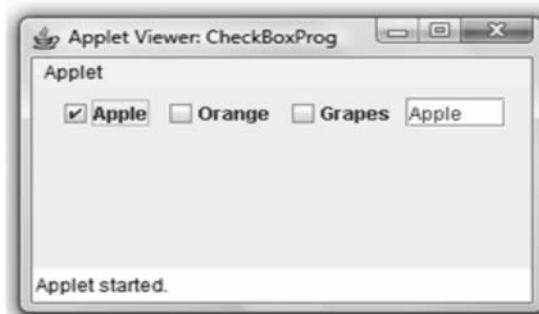
Java Program[CheckBoxProg.java]

```
import java.awt.*;  
import java.awt.event.*;  
import javax.swing.*;  
/*  
<applet code="CheckBoxProg" width=300 height=300>  
</applet>  
*/
```

```
public class CheckBoxProg extends JApplet
implements ItemListener
{
    JTextField T;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        JCheckBox chk1=new JCheckBox("Apple");
        chk1.addItemListener(this);// invoking the event handler
        contentPane.add(chk1);
        JCheckBox chk2=new JCheckBox("Orange");
        chk2.addItemListener(this); // invoking the event handler
        contentPane.add(chk2);
        JCheckBox chk3=new JCheckBox("Grapes");
        chk3.addItemListener(this); // invoking the event handler
        contentPane.add(chk3);
        T=new JTextField(5);
        contentPane.add(T);
        ButtonGroup bg=new ButtonGroup();
        bg.add(chk1);
        bg.add(chk2);
        bg.add(chk3);
    }
    public void itemStateChanged(ItemEvent e)
    {
        JCheckBox chk=(JCheckBox)e.getItem();
        T.setText(chk.getText());//displaying the appropriate string in textbox
    }
}
```

} ButtonGroup helps the objects to behave mutually exclusive.

Output



Program Explanation

In above program, the **ItemListener** event is implemented. Using the **addItemListener(this)** method the event handler function **itemStateChanged** is invoked. Thus on the clicking corresponding checkbox , the appropriate string will be displayed in the text field.

Ex. 10.8.1 : Code a Java program to implement the following : Create four check boxes. The initial state of the first box should be in checked state. The status of each check box should be displayed. When we change the state of a check box, the status should be display us updated.

AU : Dec.-18, Marks 15

Sol. :

```
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.*;
public class FourCheckBox extends JFrame
{
    JCheckBox c1,c2,c3,c4;
    JLabel l1,l2,l3,l4;
    FourCheckBox()
    {
        JFrame f=new JFrame("Frame Demo");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
        f.setSize(100,500);
        Container content=f.getContentPane();
        content.setLayout(new FlowLayout());
        c1=new JCheckBox("Red",true);
        l1=new JLabel();
        f.add(c1);
        f.add(l1);
        c2=new JCheckBox("Green");
        l2=new JLabel();
        f.add(c2);
        f.add(l2);
        c3=new JCheckBox("Blue");
        l3=new JLabel();
        f.add(c3);
        f.add(l3);
        c4=new JCheckBox("Yellow");
```

```
        l4=new JLabel();
        f.add(c4);
        f.add(l4);
        c1.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                l1.setText("Red CheckBox: " +
(e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        c2.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                l2.setText("Green CheckBox: " +
(e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        c3.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                l3.setText("Blue CheckBox: " +
(e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        c4.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                l4.setText("Yellow CheckBox: " +
(e.getStateChange()==1?"checked":"unchecked"));
            }
        });

        }
        public static void main(String[] args)
        {
            new FourCheckBox();
        }
    }
```

Output**10.9 Radio Buttons**

- The JRadioButton is a subclass of **JToggleButton**. This control is similar to the checkboxes.

- The syntax for the Radio Box will be -

JRadioButton(Icon ic);

JRadioButton (Icon ic, boolean state);

JRadioButton (String s);

JRadioButton (String s,boolean state);

JRadioButton (String s,Icon ic,boolean state);

- Following program illustrates the use of radio buttons.

Java Program[RadioButProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="RadioButProg" width=300 height=300>
</applet>
*/
public class RadioButProg extends JApplet
implements ActionListener
{
    JTextField T;
    public void init()
    {
```

```
Container contentPane=getContentPane();
contentPane.setLayout(new FlowLayout());

JRadioButton rb1=new JRadioButton("Apple");
rb1.addActionListener(this);
contentPane.add(rb1);

JRadioButton rb2=new JRadioButton("Orange");
rb2.addActionListener(this);
contentPane.add(rb2);
JRadioButton rb3=new JRadioButton("Grapes");
rb3.addActionListener(this);

contentPane.add(rb3);
T=new JTextField(5);
contentPane.add(T);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);
bg.add(rb2);
bg.add(rb3);
}
public void actionPerformed(ActionEvent e)
{
    T.setText(e.getActionCommand());
}
}
```

Output



Program Explanation

We have used **ActionListener** event to handle the radio button click. The **addActionListener** method is invoked when the button gets clicked. The appropriate command string can be displayed in the textfield.

10.10 Lists

JList is a component that displays list of text items. User can select one or multiple items from this list.

Various components of JList component are

JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

Various methods of this component are -

Method	Description
int getSelectedIndex()	It returns the index of selected item of the list.
void setListData(Object[] list)	It is used to create a read-only ListModel from an array of objects

Java Program

```
import javax.swing.*.*;
import java.awt.*.*;
/*
<applet code="ListDemo" width=200 height=200>
</applet>
*/
public class ListDemo extends JApplet
{
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new BorderLayout());
        String []
str={"Windows98\n","Windows2000\n","Windows7\n","Windows8\n","Windows10"};
```

```
JList list = new JList(str);
list.setBounds(100,100, 75,75);
contentPane.add(list);

}
}
```

Output



10.11 Choices

AU : May-11,19, Marks 10

- **JList** and **JCombobox** are very similar components but the **JList** allows to select multiple selections whereas the **JCombobox** allows only one selection at a time.
- A combo box is a combination of text field and the drop down list. The **JComboBox** is a subclass of **JComponent** class.

Java Program[ComboBoxProg.java]

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="ComboBoxProg" width=300 height=300>
</applet>
*/
public class ComboBoxProg extends JApplet
implements ItemListener
{
    JLabel L;
    public void init()
    {
```

```
Container contentPane=getContentPane();
contentPane.setLayout(new FlowLayout());
JComboBox co=new JComboBox();
co.addItem("apple");
co.addItem("orange");
co.addItem("grapes");
co.addItemListener(this);
contentPane.add(co);

L=new JLabel(new ImageIcon("apple.gif"));
contentPane.add(L);

}
public void itemStateChanged(ItemEvent e)
{
    String str=(String)e.getItem();
    L.setIcon(new ImageIcon(str+".gif")); //invokes the appropriate image file
}
}
```

Output



Program Explanation

Using the **new JComboBox()** the object for the combo box can be created. This object then invokes the method **addItem** for adding the items in the combo box. In the event handler function **itemStateChanged** the **getItem** method is used to retrieve the selected string.

Ex. 10.11.1 : Create a phonebook look-up application that displays the phone number for the person selected and looks up the name for an entered phone number. Create a GUI that consists of a pull-down list of names, a text field for entry of phone numbers to look up, and a command button to activate the look-up operation. It is your job to add the event handling to the application. There are three different events that must be handled. When the user changes the name in the pull-down list, the application should look up the appropriate telephone number in the phonebook. When the user enters a phone number in the text field and presses return or selects the button, the application should look up the number in the reverse phonebook and change the name in the pull-down list. Finally, the application should restrict input in the text field to digits and the '-' character. If a look-up operation fails, the application should simply beep.

AU : May-11, Marks 10

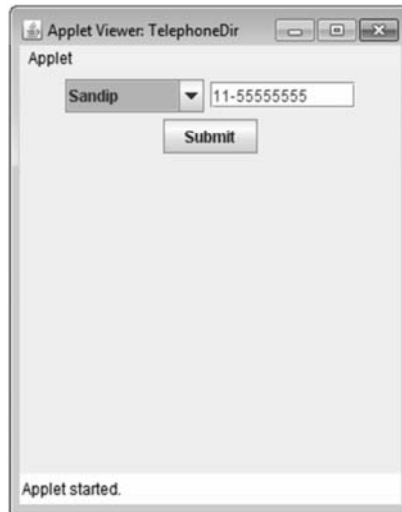
Sol. :

//Program for handling the telephone Directory using event handling

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
<applet code="TelephoneDir" width=300 height=300>
</applet>
*/
public class TelephoneDir extends JApplet
implements ItemListener,ActionListener
{
    JTextField T;
    JComboBox co;
    JButton B;
    public void init()
    {
        Container contentPane=getContentPane();
        contentPane.setLayout(new FlowLayout());
        co=new JComboBox();
        co.addItem("Select an Item");
        co.addItem("Archana");
        co.addItem("Supriya");
        co.addItem("Jayashree");
```

```
        co.addItem("Shivraj");
        co.addItem("Sandip");
        contentPane.add(co);
        co.addItemListener(this);
        T=new JTextField(10);
        contentPane.add(T);
        T.addActionListener(this);
        B=new JButton("Submit");
        contentPane.add(B);
        B.addActionListener(this);
    }
    public void itemStateChanged(ItemEvent e)
    {
        String str=(String)e.getItem();
        if(str=="Archana")
            T.setText("11-11111111");
        else if(str=="Supriya")
            T.setText("11-22222222");
        else if(str=="Jayashree")
            T.setText("11-33333333");
        else if(str=="Shivraj")
            T.setText("11-44444444");
        else if(str=="Sandip")
            T.setText("11-55555555");
        else
        {
            if((str=="") || (T.getText()==""))//generating Beep
                System.out.println("\007");
        }
    }
    public void actionPerformed(ActionEvent e)
    {
        String str=(String)T.getText();
        co.setSelectedItem(str);
        if(co.getSelectedItem()=="Select an Item");//no item selected
            System.out.println("\007\007");//generating Beep
    }
}
```

Output



Review Question

1. Present the hierarchy of Java swing classes and methods of component class. Create a simple 'combo box; in Java swing using the classes and methods.

AU : IT : May-19, Marks 9

10.12 Scrollbars

AU : May-14, 15, Marks 16

- The scroll pane is a rectangular areas in which some component can be placed.
- The component can be viewed with the help of **horizontal** and **vertical** scroll bars.
- Using the **JScrollPane** class the component can be added in the program.
- The **JScrollPane** class extends the **JComponent** class.
- There are **three constructors** that can be used for this component -

`JScrollPane(Component component)`

`JScrollPane(int vscrollbar, int hscrollbar)`

`JScrollPane(Component component, int vscrollbar, int hscrollbar)`

The *component* represents the reference to the component. The *vscrollbar* and *hscrollbar* are the integer values for the vertical and horizontal scroll bars. These values can be defined by the constants such as

Constant	Meaning
HORIZONTAL_SCROLLBAR_ALWAYS	It always displays the horizontal scroll bar.
HORIZONTAL_SCROLLBAR_NEEDED	It displays the horizontal scrollbar if required.
VERTICAL_SCROLLBAR_ALWAYS	It always displays the vertical scroll bar.
VERTICAL_SCROLLBAR_NEEDED	It displays the vertical scrollbar if required.

- Following is a simple program which illustrates the use of scrollpane.

Step 1 : Create a label

Step 2 : Create a panel

Step 3 : Use an image with the help of label

Step 4 : Add the label on the panel

Step 5 : Create a content pane.

Step 6 : Create a scrollpane component by passing the image (within a panel) as a *component* to it.

Step 7 : Add the the scrollpane component to the content pane component.

Java Program

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="ScrollPaneDemo" width=150 height=150>
</applet>
*/
public class ScrollPaneDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new BorderLayout());
        JPanel mypanel = new JPanel();
        JLabel L1 = new JLabel();
        ImageIcon i = new ImageIcon("img.jpg");
        L1.setLocation(20, 100);
        L1.setSize(120, 120);
        L1.setIcon(i);
        mypanel.add(L1);

        int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;
        JScrollPane jsp = new JScrollPane(mypanel, vscrollbar, hscrollbar);
        contentPane.add(jsp, BorderLayout.CENTER);
    }
}
```

Output

Ex. 10.12.1 : List does not support scrolling. Why ? How this can be remedied ? Explain with an example.

Sol. : List is a component which simply displays the list of items. But this component does not support scrolling. The scrolling facility can be added to this component by using the scroll pane. Following are the steps that can be carried out for that purpose -

Step 1 : Create the content pane.

Step 2 : Create a list of items. (Preferably list of lots of items)

Step 3 : Create a scroll pane component by setting the values to vertical and horizontal component as `ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED` and `ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED`;

Step 4 : Then pass the JList component to scroll pane component.

Step 5 : Then add this scroll pane component to content pane.

Following program illustrates this idea -

Java Program

```
import javax.swing.*.*;
import java.awt.*.*;
import javax.swing.tree.*.*;
/*
<applet code="ListDemo" width=150 height=90>
</applet>
*/
```

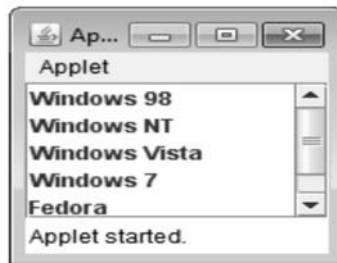
```
public class ListDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();

        contentPane.setLayout(new BorderLayout());
        String[] str = { "Windows 98\n", "Windows NT\n", "Windows Vista\n", "Windows
7\n", "Fedora\n", "Ubuntu"};
        JList list = new JList(str);

        int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

        JScrollPane mypane = new JScrollPane(list, vscrollbar, hscrollbar);
        contentPane.add(mypane, BorderLayout.CENTER);
    }
}
```

Output



Ex. 10.12.2 : List does not support scrolling. Why ? How this can be remedied ? Explain with an example.

Sol. : JList is a component which simply displays the list of items. But this component does not support scrolling. The scrolling facility can be added to this component by using the scroll pane. Following are the steps that can be carried out for that purpose -

Step 1 : Create the content pane.

Step 2 : Create a list of items. (preferably list of lots of items)

Step 3 : Create a scroll pane component by setting the values to vertical and horizontal component as ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED and ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

Step 4 : Then pass the JList component to scroll pane component.

Step 5 : Then add this scroll pane component to content pane.

Following program illustrates this idea -

Java Program

```
import javax.swing.*.*;
import java.awt.*.*;
import javax.swing.tree.*.*;
/*
<applet code="ListDemo" width=150 height=90>
</applet>
*/

public class ListDemo extends JApplet
{
    public void init()
    {
        Container contentPane = getContentPane();

        contentPane.setLayout(new BorderLayout());
        String[] str = { "Windows 98\n", "Windows NT\n", "Windows Vista\n",
            "Windows 7\n", "Fedora\n", "Ubuntu" };
        JList list = new JList(str);

        int vscrollbar = ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED;
        int hscrollbar = ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED;

        JScrollPane mypane = new JScrollPane(list, vscrollbar, hscrollbar);
        contentPane.add(mypane, BorderLayout.CENTER);
    }
}
```

Output



10.13 **Menus**

- The menu bar is the most standard GUI which is present in almost all the applications.
- We can create a menu bar which contains several menus.
- Each menu can have several menu items.
- The separator can also be used to separate out these menus.
- Various methods and APIs that can be used for creating the Menus in Swing are -

JMenuBar

This class creates a menu bar. A menu bar normally contains several menus.

JMenu(String *str*)

This class creates several menus. The menus contain the menu items. The string *str* denotes the names of the menus.

JMenuItem(String *str*)

The menu items are the parts of menus. The string *str* denotes the names of the menu items.

setMnemonic(char *ch*)

The mnemonic key can be set using this method. The character which is passed to it as an argument becomes the mnemonic key. Hence using alt + ch you can select that particular menu.

JSeparator

This is the constructor of the class JSeparator which adds separating line between the menu items.

setJMenuBar

This method is used to set menu bar to a specific frame.

In the following Java program, we have created a frame on which a menu bar is placed. The menu bar contains three menus - File, Edit and Help. Each of these menus have their own set of menu items. The separators and mnemonic keys are also used in this program.

Java Program

```
import java.awt.*;  
import javax.swing.*;  
class MenuProg
```

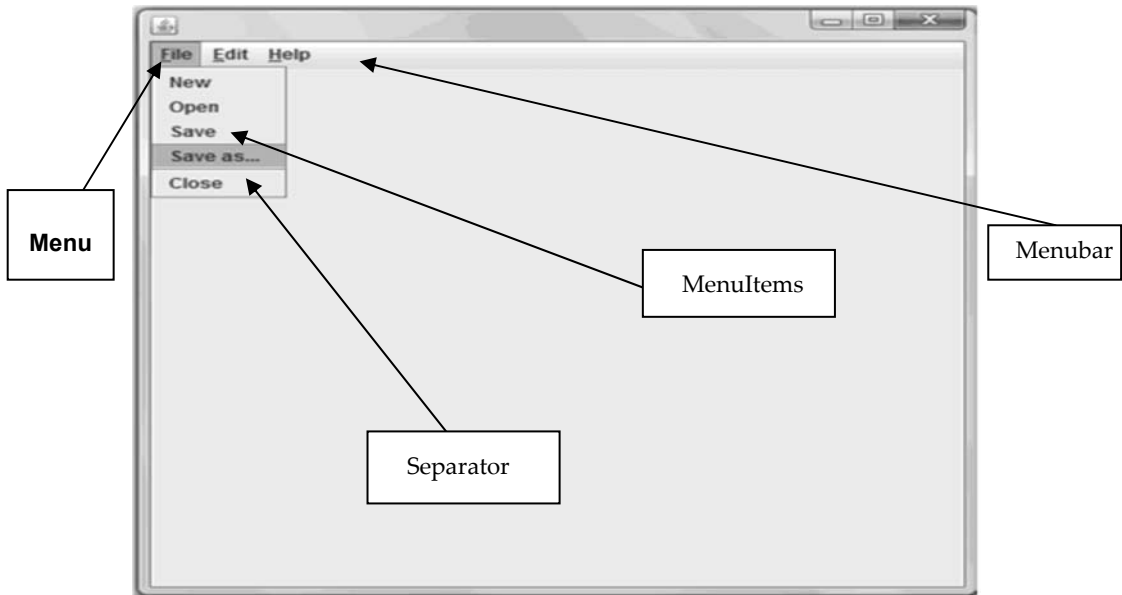
```
{
    public static void main(String[] args)
    {
        JMenu menu1=new JMenu("File");←Creating Menu
        menu1.setMnemonic('F');
        menu1.add(new JMenuItem("New"));
        menu1.add(new JMenuItem("Open"));
        menu1.add(new JMenuItem("Save"));
        menu1.add(new JMenuItem("Save as..."));
        menu1.add(new JSeparator());
        menu1.add(new JMenuItem("Close"));
        JMenu menu2=new JMenu("Edit");
        menu2.setMnemonic('E');
        menu2.add(new JMenuItem("Undo"));
        menu2.add(new JMenuItem("Redo"));
        menu2.add(new JSeparator());
        menu2.add(new JMenuItem("Cut"));
        menu2.add(new JMenuItem("Copy"));
        menu2.add(new JMenuItem("Paste"));

        JMenu menu3=new JMenu("Help");
        menu3.setMnemonic('H');
        JMenuBar menubar=new JMenuBar();←Creating menubar
        menubar.add(menu1);
        menubar.add(menu2);
        menubar.add(menu3);

        JFrame f=new JFrame();←Creating frames
        f.setJMenuBar(menubar);←placing menubar on the frame
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
        f.setSize(200,200);
    }
}
```

Adding menu items to menus

Adding menus to menubar

Output

In order to try out the mnemonic keys just press Alt+f key and the File menu gets selected.

Ex. 10.13.1 : Write a program to create a frame with the following menus, such that the corresponding geometric object is created when a menu is clicked. (i) circle (ii) Rectangle (iii) Line (iv) Diagonal for the rectangle

AU : May-14, Marks 16

Sol. :

```
import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.*;
class ShapesMenu extends JFrame implements ActionListener
{
    JMenuBar mb;
    JMenu menu;
    JMenuItem rect,line,oval;
    ShapesMenu()
    {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        mb=new JMenuBar();

        menu=new JMenu("Shapes");
        mb.add(menu);

        rect=new JMenuItem("Rectangle");
```

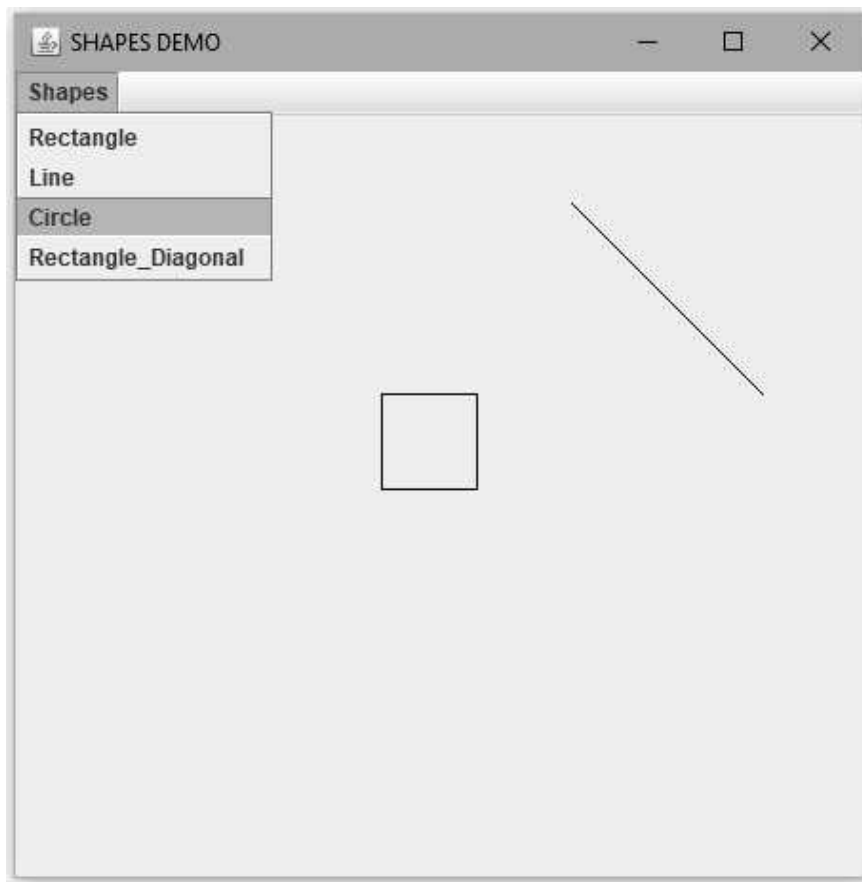
```
    rect.addActionListener(this);
    menu.add(rect);

    line=new JMenuItem("Line");
    line.addActionListener(this);
    menu.add(line);

    oval=new JMenuItem("Circle");
    oval.addActionListener(this);
    menu.add(oval);

    line=new JMenuItem("Rectangle_Diagonal");
    line.addActionListener(this);
    menu.add(line);

    setJMenuBar(mb);
}
public void actionPerformed(ActionEvent ae)
{
    String str=ae.getActionCommand();
    Graphics g=getGraphics();
    if(str=="Rectangle")
        g.drawRect(200,200,50,50);
    if(str=="Line")
        g.drawLine(300,100,400,200);
    if(str=="Circle")
        g.drawOval(200,300,100,100);
    if(str=="Rectangle_Diagonal")
        g.drawLine(200,200,250,250);
}
public static void main(String args[])
{
    ShapesMenu f=new ShapesMenu();
    f.setTitle("SHAPES DEMO");
    f.setSize(500,500);
    f.setVisible(true);
}
}
```



Ex. 10.13.2 : Create a simple menu application that enables a user to select one of the following items :

Radio 1

Radio 2

Radio 3

Radio 4

Radio 5

Red Dragon Radio

(i) From the menu bar of the application (ii) From a pop up menu (iii) From a toolbar

Add tooltips to each menu item that indicates some information about the Radio station such as type of music and its broadcast frequency.

AU : May-15, Marks 16

Sol. :

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class MenuApplication
{
    public static void main(String args[])
    {
        JFrame f = new JFrame("Simple Menu Application");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        /* Creating Menu Bar */
        JMenuBar bar = new JMenuBar();
        JMenu menu = new JMenu("Options");
        menu.setMnemonic(KeyEvent.VK_O);
        ButtonGroup group = new ButtonGroup();
        JRadioButtonMenuItem menuItem = new JRadioButtonMenuItem("Radio 1");
        menuItem.setToolTipText("Music Masala 94.5KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 2");
        menuItem.setToolTipText("Retro Melody 100.1KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 3");
        menuItem.setToolTipText("Classical Khajana 98.3KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 4");
        menuItem.setToolTipText("Music on Demand 91.1KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Radio 5");
        menuItem.setToolTipText("Disco Station 95.3KHz");
        group.add(menuItem);
        menu.add(menuItem);

        menuItem = new JRadioButtonMenuItem("Red Dragon Radio");
        group.add(menuItem);
        menu.add(menuItem);
    }
}
```

```
bar.add(menu);
f.setJMenuBar(bar);

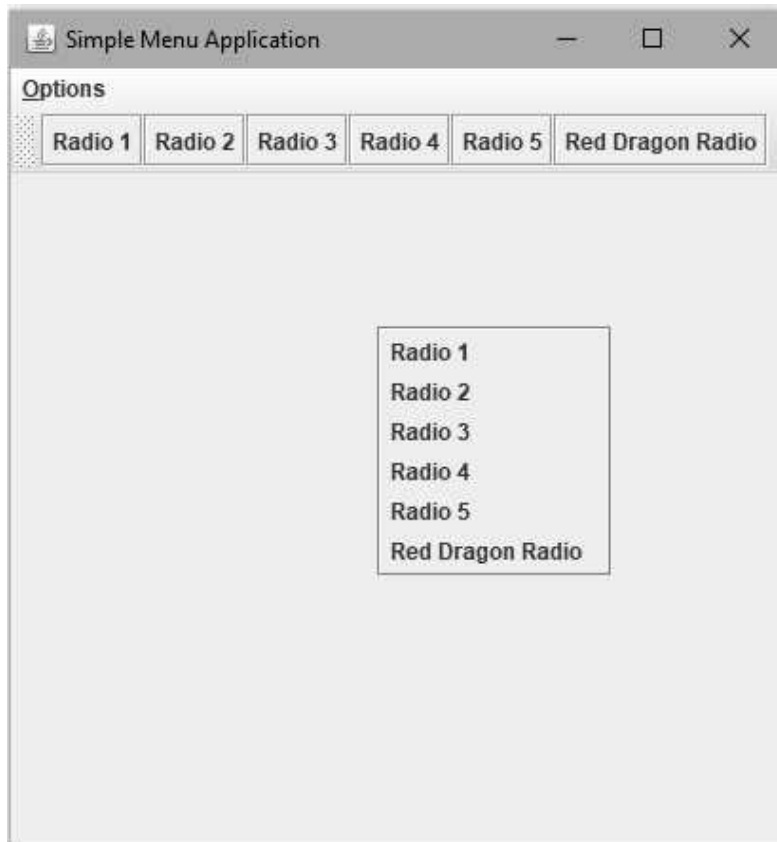
/* Creating ToolBar */
JToolBar toolbar = new JToolBar();
toolbar.setRollover(true);
JButton button = new JButton("Radio 1");
toolbar.add(button);
button = new JButton("Radio 2");
toolbar.add(button);
button = new JButton("Radio 3");
toolbar.add(button);
button = new JButton("Radio 4");
toolbar.add(button);
button = new JButton("Radio 5");
toolbar.add(button);
button = new JButton("Red Dragon Radio");
toolbar.add(button);
Container contentPane = f.getContentPane();
contentPane.add(toolbar, BorderLayout.NORTH);

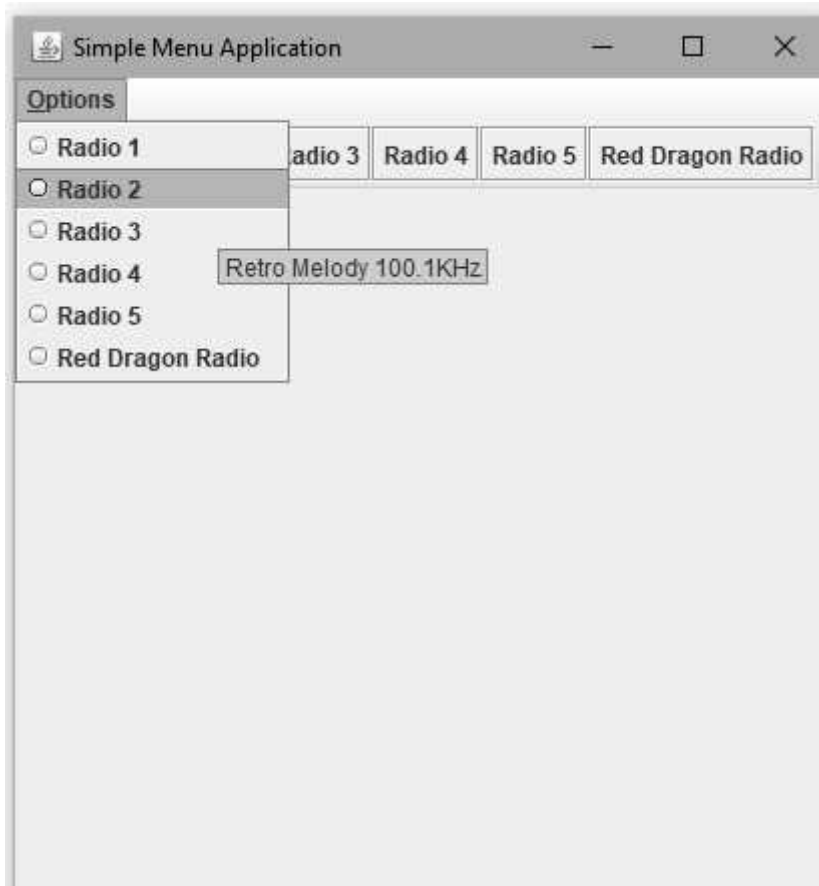
/* Creating Popup menu */
JPopupMenu popup = new JPopupMenu();
JMenuItem mItem=new JMenuItem("Radio 1");
popup.add(mItem);
mItem=new JMenuItem("Radio 2");
popup.add(mItem);
mItem=new JMenuItem("Radio 3");
popup.add(mItem);
mItem=new JMenuItem("Radio 4");
popup.add(mItem);
mItem=new JMenuItem("Radio 5");
popup.add(mItem);
mItem=new JMenuItem("Red Dragon Radio");
popup.add(mItem);

f.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        showPopup(e);
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        showPopup(e);
    }
    private void showPopup(MouseEvent e) {
```

```
        if (e.isPopupTrigger()) {  
            popup.show(e.getComponent(),  
                e.getX(), e.getY());  
        }  
    }  
});  
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
f.setSize(500, 500);  
f.setVisible(true);  
}  
}
```

Output





10.14 Dialog boxes

- The dialog box is useful to display some useful messages to the user. Using Swing we can create three types of dialog boxes -
 1. Simple message dialog box
 2. Confirm message dialog box
 3. Input dialog box
- These dialog boxes are created using **JOptionPane** class. This class is present in the **javax.swing.*** package. Hence we must import this file at the beginning.

Let us understand these dialog boxes with the help of programming examples -

Simple message dialog box

- This type of message box simply displays the informative message to the user.
- It has only OK button.

- It is expected that after reading out the message the user must click the OK button to return.
- The **JOptionPane** class provides the method **showMessageDialog()** in order to display the simple message box.
- Following is the simple Java program which shows how to create simple message dialog box -

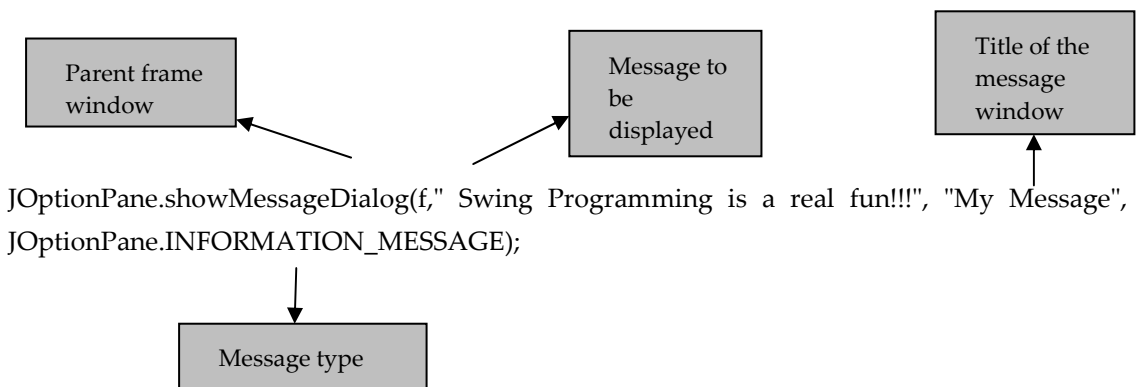
Java Program[DialogBox1Prog.java]

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class DialogBox1Prog
implements ActionListener
{
    JFrame f;
    public static void main(String[] args)
    {
        new DialogBox1Prog();
    }
    public DialogBox1Prog()
    {
        f=new JFrame("Dialog Box Demo");
        JButton B=new JButton("Click Me!!");
        Container container=f.getContentPane();
        container.setLayout(new FlowLayout());
        container.add(B);
        B.addActionListener(this);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        JOptionPane.showMessageDialog(f,"Swing Programming is a real fun!!!","My
        Message",JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Output**Program Explanation**

In above Java program, we have created one frame window, on which one push button is placed. When we click the button, then the simple message box will be displayed. This message box has only one OK button.

To display the dialog box we have

**Confirm message dialogbox**

- This type of dialog box will display some message to the user and will get the confirmation from him.
- User can give his confirmation either by clicking OK, CANCEL, YES or NO button.

- The confirm message dialog box will display the message either along with the OK and CANCEL button or with the YES, NO or CANCEL button.
- In the following Java program we have used three types of message boxes - simple message box, confirm message box with OK and CANCEL and the confirm message box with YES, NO and CANCEL.

Java Program[DialogBox2Prog.java]

```
import java.awt.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class DialogBox2Prog
implements ActionListener
{
    JFrame f;
    public static void main(String[] args)
    {
        new DialogBox2Prog();
    }
    public DialogBox2Prog()
    {
        f=new JFrame("Dialox Box Demo");
        Container container=f.getContentPane();
        container.setLayout(new FlowLayout());
        JButton B=new JButton("Simple Message DialogBox");
        container.add(B);
        B.addActionListener(this);

        B=new JButton("OK and Cancel DialogBox");
        container.add(B);
        B.addActionListener(this);

        B=new JButton("Yes/No/Cancel DialogBox");
        container.add(B);
        B.addActionListener(this);

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public void actionPerformed(ActionEvent e) //event handler function
```

On this button click simple message box will be displayed

On this button click OK/Cancel dialog box will be displayed

On this button click Yes/No/Cancel dialog box will be displayed

```
{
String s=e.getActionCommand(); //getting the string present on the button
if(s.equals("Simple Message DialogBox"))
    JOptionPane.showMessageDialog(f,"Simple Message");
else if(s.equals("OK and Cancel DialogBox"))
{
    if(JOptionPane.showConfirmDialog(f,"This is OK/Cancel Message","Confirm Dialog
        Box",JOptionPane.OK_CANCEL_OPTION)==0)
        JOptionPane.showMessageDialog(f,"You have clicked OK button");
    else
        JOptionPane.showMessageDialog(f,"You have clicked Cancel button");
}
else if(s.equals("Yes/No/Cancel DialogBox"))
{
    if(JOptionPane.showConfirmDialog(f,"This is Yes/No/Cancel Message","Confirm Dialog
        Box",JOptionPane.YES_NO_CANCEL_OPTION)==0)
    {
        JOptionPane.showMessageDialog(f,"You have clicked Yes button");
    }
    else
    {
        JOptionPane.showMessageDialog(f,"You have clicked OK or Cancel button");
    }
}

}

}
}
```

Output



Program Explanation

For the confirm dialog box, the option type could be OK_CANCEL, YES_NO_OPTION, OK_OPTION, NO_OPTION which are associated with some constant values.

In above program, when the user clicks the button for OK and Cancel message then appropriate message will be displayed. If user clicks the OK button then the message "You have clicked OK button will be displayed" otherwise the message "You have clicked Cancel button" will be displayed.

Two Marks Questions with Answers

Q.1 List any four features of swing not present in java .

Ans. :

1. Swing has a rich set of GUI components such as buttons and check boxes, trees, tabbed panes and so on.
2. It has got pluggable look and feel support.
3. Support for data transfer is built into Swing and works between Swing components within an application, between Java applications, and between Java and native applications.
4. If you want your program to run within a browser window, you can create it as an applet and run it.

Q.2 What is the function of a) SetLayout b) FlowLayout ?

AU : CSE : Dec.-10

Ans. : The setLayout is used for setting the layout of the device. The parameter to this function is an object of the layout manager.

The FlowLayout manager is the simplest layout manager. Using this layout manager the components are arranged from top left corner lying down from left to right and from top to bottom.

Q.3 Write down the statements used to create a JFrame and b JButton and then add the JButton to the JFrame.

AU : IT : May-11

Ans. : The code segment is -

```
JFrame f=new JFrame("Frame Demo");
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
f.setVisible(true);
f.setSize(300,300);
Container content=f.getContentPane();
```

```
content.setLayout(new FlowLayout(FlowLayout.CENTER));
JButton b=new JButton("Submit");
content.add(b);
```

Q.4 Differentiate Gridbaglayout from GridLayout.

Ans. : The GridbagLayout manager is a flexible layout manager which aligns components vertically and horizontally, without requiring that the components be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.

GridLayout is a kind of layout manager which arranges the components into a grid of equal sized rows and columns.

Q.5 List out the Swing components**AU : CSE : Dec.-12**

Ans. : Various most commonly used components of swing are -

1. AbstractButton
2. ButtonGroup
3. JApplet
4. JButton
5. JCheckBox
6. JComboBox
7. JRadioButton
8. JLabel
9. JTextArea
10. JTextField
11. JSlider

Q.6 Why are swing components called lightweight components ?**AU : IT : May-13**

Ans. : Swing components are considered as lightweight components because they are fully implemented in Java, without calling the native operating system for drawing the graphical user interface components.

On the other hand, AWT (Abstract Window Toolkit) are heavyweight components, as AWT merely makes calls to the operating system in order to produce its GUI components.

Q.7 How can you prevent overwriting of a displayed text in a TextField of a Java program ?**AU : IT : Dec.-13**

Ans. : By passing the desired string as a first parameter to the JTextField during its creation one can prevent the overwriting of the displayed text in a text field. Refer section 10.5.

Q.8 What are the purposes of JPanel ?**AU : Dec 19**

Ans. : The JPanel is a container class that can store a group of components. The JPanel is used by various layout managers to organize components.



Object Oriented Programming Laboratory

Contents

1. Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, type of EB connection (i.e domestic or commercial). Compute the bill amount using the following tariff.....L - 2
2. Develop a java application to implement currency converter (Dollar to INR, EURO to INR, Yen to INR and vice versa), distance converter (meter to KM, miles to KM and vice versa) , time converter (hours to minutes, seconds and vice versa) using packages.L - 4
3. Develop a java application with Employee class with Emp_name, Emp_id, Address, Mail_id, Mobile_no as members. Inherit the classes, Programmer, Assistant Professor, Associate Professor and Professor from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary.L - 13
4. Design a Java interface for ADT Stack. Implement this interface using array. Provide necessary exception handling in both the implementations.L - 17
5. Write a program to perform string operations using ArrayList. Write functions for the following a. Append - add at end b. Insert - add at particular index c. Search d. List all string starts with given letter.....L - 19
6. Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named print Area(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape. ...L - 24
7. Write a Java program to implement user defined exception handling.L - 26
8. Write a Java program that reads a file name from the user, displays information about whether the file exists, whether the file is readable, or writable, the type of file and the length of the file in bytes.....L - 26
9. Write a java program that implements a multi-threaded application that has three threads. First thread generates a random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.L - 27
10. Write a java program to find the maximum value from the given type of elements using a generic function.L - 30
11. Design a calculator using event-driven programming paradigm of Java with the following options. a) Decimal manipulations b) Scientific manipulations.....L - 31

Experiment 1 : Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, type of EB connection (i.e domestic or commercial). Compute the bill amount using the following tariff.

If the type of the EB connection is domestic, calculate the amount to be paid as follows:

- **First 100 units - Rs. 1 per unit**
- **101-200 units - Rs. 2.50 per unit**
- **201 -500 units - Rs. 4 per unit**
- **> 501 units - Rs. 6 per unit**

If the type of the EB connection is commercial, calculate the amount to be paid as follows :

- **First 100 units - Rs. 2 per unit**
- **101-200 units - Rs. 4.50 per unit**
- **201 -500 units - Rs. 6 per unit**
- **> 501 units - Rs. 7 per unit**

Implementation:

```
import java.util.Scanner;
class ElectBill
{
    int ConsumerNo;
    String ConsumerName;
    int PrevReading;
    int CurrReading;
    String EBConn;
    double Bill;
    void input_data()
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("\n Enter Consumer Number: ");
        ConsumerNo = sc.nextInt();
        System.out.println("\n Enter Consumer Name: ");
        ConsumerName = sc.next();
        System.out.println("\n Enter Previous Units: ");
        PrevReading = sc.nextInt();
        System.out.println("Enter Current Units consumed:");
        CurrReading = sc.nextInt();
        System.out.println("Enter the types of EB Connection(domestic or commercial)");
```

```
        EBConn = sc.next();
    }
    double calculate_bill()
    {
        int choice;
        if(EBConn=="domenstic")
            choice=1;
        else
            choice=2;
        switch(choice)
        {
            case 1:
                if(CurrReading>=0 && CurrReading<=100)
                    Bill=CurrReading*1;
                else if(CurrReading>100 && CurrReading <= 200)
                    Bill=(100*1)+((CurrReading-100)*2.50);
                else if(CurrReading>200 && CurrReading <= 500)
                    Bill=(100*1)+(100*2.50)+((CurrReading-200)*4);
                else
                    Bill=(100*1)+(100*2.50)+(300*4)+((CurrReading-500)*6);
                break;
            case 2:
                if(CurrReading>=0 && CurrReading<=100)
                    Bill=CurrReading*2;
                else if(CurrReading>100 && CurrReading <= 200)
                    Bill=(100*1)+((CurrReading-100)*4.50);
                else if(CurrReading>200 && CurrReading <= 500)
                    Bill=(100*1)+(100*2.50)+((CurrReading-200)*6);
                else
                    Bill=(100*1)+(100*2.50)+(300*4)+((CurrReading-500)*7);

                break;
        }
        return Bill;
    }
    void display()
    {
        System.out.println("-----");
        System.out.println("ELCTRICITY BILL");
        System.out.println("-----");
        System.out.println("Consumer Number: "+ConsumerNo);
        System.out.println("Consumer Name: "+ConsumerName);
        System.out.println("Consumer Previous Units: "+PrevReading);
        System.out.println("Consumer Current Units: "+CurrReading);
    }
}
```

```
        System.out.println("Type of EBConnection: "+EBConn);
        System.out.println("-----");
        System.out.println("Total Amount(Rs.): "+Bill);
    }
}
class ElectBillGen
{
    public static void main (String[] args)
    {
        ElectBill b=new ElectBill();
        b.input_data();
        b.calculate_bill();
        b.display();
    }
}
```

Output

```
Enter Consumer Number:
101
Enter Consumer Name:
AAA
Enter Previous Units:
310
Enter Current Units consumed:
480
Enter the types of EB Connection(domestic or commercial)
domestic
-----
ELCTRICITY BILL
-----
Consumer Number: 101
Consumer Name: AAA
Consumer Previous Units: 310
Consumer Current Units: 480
Type of EBConnection: domestic
-----
Total Amount(Rs.): 2030.0
```

Experiment 2 : Develop a java application to implement currency converter (Dollar to INR, EURO to INR, Yen to INR and vice versa), distance converter (meter to KM, miles to KM and vice versa) , time converter (hours to minutes, seconds and vice versa) using packages.

Implementation :

Step 1 : Create a folder named ConversionDemo. This is the name of the package. Following 3 Java files are stored in this folder.

CurrencyConverter.java

```
package ConversionDemo;
public class CurrencyConverter
{
    double ER = 0;
    public CurrencyConverter(double CurrentExchange)
    {
        ER = CurrentExchange;
    }
    public double DollarToINR(double Dollars)
    {
        double INR = 0;
        INR = Dollars * ER;
        return INR;
    }
    public double INRToDollar(double INR)
    {
        double Dollars = 0;
        Dollars = INR / ER;
        return Dollars;
    }
    public double EuroToINR(double Euros)
    {
        double INR = 0;
        INR = Euros * ER;
        return INR;
    }
    public double INRToEuro(double INR)
    {
        double Euros = 0;
        Euros = INR / ER;
        return Euros;
    }
    public double YenToINR(double Yens)
    {
        double INR = 0;
        INR = Yens * ER;
        return INR;
    }
    public double INRToYen(double INR)
    {
        double Yens = 0;
        Yens = INR / ER;
    }
}
```

```
        return Yens;
    }
}
```

DistanceConverter.java

```
package ConversionDemo;
public class DistanceConverter
{
    public double MeterToKM(double Meters)
    {
        double KM = 0;
        KM = Meters / 1000;
        return KM;
    }
    public double KMToMeter(double KM)
    {
        double Meters = 0;
        Meters = KM * 1000;
        return Meters;
    }
    public double MileToKM(double Miles)
    {
        double KM = 0;
        KM = Miles / 0.621371;
        return KM;
    }
    public double KMToMile(double KM)
    {
        double Miles = 0;
        Miles = KM * 0.621371;
        return Miles;
    }
}
```

TimeConverter.java

```
package ConversionDemo;
public class TimeConverter
{
    public double HrToMin(double Hours)
    {
        double Minutes = 0;
        Minutes = Hours * 60;
        return Minutes;
    }
}
```

```
public double MinToHour(double Minutes)
{
    double Hours = 0;
    Hours = Minutes / 60;
    return Hours;
}
public double HrToSec(double Hours)
{
    double Seconds = 0;
    Seconds = Hours * 3600;
    return Seconds;
}
public double SecToHour(double Seconds)
{
    double Hours = 0;
    Hours = Seconds / 3600;
    return Hours;
}
}
```

Step 2 : Now go to the parent directory of **ConversionDemo** directory and store following Java program under this parent directory

Converter.java

```
import ConversionDemo.CurrencyConverter;
import ConversionDemo.DistanceConverter;
import ConversionDemo.TimeConverter;
import java.util.Scanner;
class Converter
{
    public static void main(String[] args) throws ClassNotFoundException
    {
        double CurrentExchange;
        int choice,choice1,choice2,choice3;
        double inr;
        double km;
        double hr;
        char ans='y';
        do
        {
            System.out.println("\n Main Menu");
            System.out.println("1.Currency Converter \n2.Distance Converter \n3. Time Converter");
            System.out.println("Enter your choice: ");
```

```
Scanner input = new Scanner(System.in);
choice = input.nextInt();
switch(choice)//outer Switch
{
    case 1: System.out.println("\tCurrency Conversion");
    do
    {
        System.out.println("Menu For Currency Conversion");
        System.out.println("1. Dollar to INR");
        System.out.println("2. INR to Dollar");
        System.out.println("3. Euro to INR");
        System.out.println("4. INR to Euro");
        System.out.println("5. Yen to INR");
        System.out.println("6. INR to Yen");

        System.out.println("Enter your choice: ");
        choice1 = input.nextInt();
        System.out.println("Please enter the current exchange rate: ");
        CurrentExchange = input.nextDouble();
        CurrencyConverter cc=new CurrencyConverter(CurrentExchange);
        switch(choice1)//inner switch
        {
            case 1:
                System.out.print("Enter Dollars :");
                double dollar=input.nextDouble();
                System.out.println (dollar+" dollars are converted to
                                "+ cc.DollarToINR(dollar)+" Rs.");
                break;
            case 2:
                System.out.print("Enter INR :");
                inr=input.nextDouble();
                System.out.println(inr+" Rs. are converted to
                                "+ cc.INRToDollar(inr)+" Dollars");
                break;
            case 3:
                System.out.print("Enter Euro :");
                double euro=input.nextDouble();
                System.out.println(euro+" Euros are converted to
                                "+ cc.EuroToINR(euro)+" Rs.");
                break;
            case 4:
                System.out.print("Enter INR :");
                inr=input.nextDouble();
                System.out.println(inr+" Rs. are converted to
```

```
                "+cc.INRToEuro(inr)+" Euros");
break;
case 5:
    System.out.print("Enter Yen :");
    double yen=input.nextDouble();
    System.out.println(yen+" Yens are converted to
                "+cc.YenToINR(yen)+" Rs.");
break;
case 6:
    System.out.print("Enter INR :");
    inr=input.nextDouble();
    System.out.println(inr+" Rs. are converted to
                "+cc.INRToYen(inr)+" Yens");
break;

}
System.out.println("Do You want to go to Currency Conversion
                Menu?(y/n) ");
ans = input.next().charAt(0);
}while(ans!='y');
break;
case 2: System.out.println("\tDistance Conversion");
do
{
    System.out.println("Menu For Distance Conversion");
    System.out.println("1. Meter to Km");
    System.out.println("2. Km to Meter");
    System.out.println("3. Miles to Km");
    System.out.println("4. Km to Miles");
    System.out.println("Enter your choice: ");
    choice2 = input.nextInt();
    DistanceConverter dc=new DistanceConverter();
    switch(choice2)
    {
        case 1:
            System.out.print("Enter meters to convert to Km:");
            double meter=input.nextDouble();
            System.out.println(meter+" Meters are converted to
                    "+dc.MeterToKM(meter)+" Km.");
            break;
        case 2:
            System.out.print("Enter Km to convert to meters:");
            km=input.nextDouble();
            System.out.println(km+" Km. are converted meters
```

```
                "+dc.KMToMeter(km)+" Meters");
        break;
    case 3:
        System.out.print("Enter Miles to convert to Km:");
        double miles=input.nextDouble();
        System.out.println(miles+" Miles are converted to
                "+dc.MileToKM(miles)+" Km.");
        break;
    case 4:
        System.out.print("Enter Km to convert to miles:");
        km=input.nextDouble();
        System.out.println(km+" Km. are converted miles
                "+dc.KMToMile(km)+" Miles");
        break;

    }
    System.out.println("Do You want to go to Distance Conversion
            Menu?(y/n) ");
    ans = input.next().charAt(0);
}while(ans!='y');
break;
case 3:System.out.println("\tTime Conversion");
do
{
    System.out.println("Menu For Time Conversion");
    System.out.println("1. Hour to Minutes");
    System.out.println("2. Minutes to Hour");
    System.out.println("3. Hour to Seconds");
    System.out.println("4. Seconds to Hour");
    System.out.println("Enter your choice: ");
    choice3 = input.nextInt();
    TimeConverter tc=new TimeConverter();
    switch(choice3)
    {
        case 1:
            System.out.print("Enter hours to convert to Minutes:");
            hr=input.nextDouble();
            System.out.println(hr+" Hours are converted to
                    "+tc.HrToMin(hr)+" min.");

            break;
        case 2:
            System.out.print("Enter Minutes to convert to hours:");
            double minutes=input.nextDouble();
            System.out.println(minutes+" Minutes. are converted hours
```

```
        "+tc.MinToHour(minutes)+" Hours");
    break;
    case 3:
        System.out.print("Enter Hours to convert to Seconds:");
        hr=input.nextDouble();
        System.out.println(hr+" Hours are converted to
        "+tc.HrToSec(hr)+" Seconds.");
    break;
    case 4:
        System.out.print("Enter Seconds to convert to hours:");
        double seconds=input.nextDouble();
        System.out.println(seconds+" Seconds. are converted hours
        "+tc.SecToHour(seconds)+" Hours");
    break;
}
System.out.println("Do You want to go to Time Conversion Menu?(y/n) ");
ans = input.next().charAt(0);
}while(ans=='y');
break;
} //end of outer switch
System.out.println("Do you want to go back to Main Menu?(y/n)");
ans=input.next().charAt(0);
}while(ans=='y');
} //end of main
} //end of class
```

Step 3 : Open the Command prompt and issue following commands

Output

```
D:\>cd ConversionDemo
D:\ConversionDemo>javac CurrencyConverter.java
D:\ConversionDemo>javac DistanceConverter.java
D:\ConversionDemo>javac TimeConverter.java
D:\ConversionDemo>cd..
D:\>javac Converter.java
D:\>java Converter
Main Menu
1.Currency Converter
2.Distance Converter
3. Time Converter
Enter your choice:
1
    Currency Conversion
Menu For Currency Conversion
1. Dollar to INR
```

2. INR to Dollar

3. Euro to INR

4. INR to Euro

5. Yen to INR

6. INR to Yen

Enter your choice:

1

Please enter the current exchange rate:

66.21

Enter Dollars :25

25.0 dollars are converted to 1655.2499999999998 Rs.

Do You want to go to Currency Conversion Menu?(y/n)

y

Menu For Currency Conversion

1. Dollar to INR

2. INR to Dollar

3. Euro to INR

4. INR to Euro

5. Yen to INR

6. INR to Yen

Enter your choice:

4

Please enter the current exchange rate:

81.1

Enter INR :5000

5000.0 Rs. are converted to 61.652281134401974 Euros

Do You want to go to Currency Conversion Menu?(y/n)

n

Do you want to go back to Main Menu?(y/n)

y

Main Menu

1.Currency Converter

2.Distance Converter

3. Time Converter

Enter your choice:

2

Distance Conversion

Menu For Distance Conversion

1. Meter to Km

2. Km to Meter

3. Miles to Km

4. Km to Miles

Enter your choice:

1

Enter meters to convert to Km:4000

4000.0 Meters are converted to 4.0 Km.

Do You want to go to Distance Conversion Menu?(y/n)

n

Do you want to go back to Main Menu?(y/n)

N

Experiment 3 : Develop a java application with Employee class with Emp_name, Emp_id, Address, Mail_id, Mobile_no as members. Inherit the classes, Programmer, Assistant Professor, Associate Professor and Professor from employee class. Add Basic Pay (BP) as the member of all the inherited classes with 97% of BP as DA, 10 % of BP as HRA, 12% of BP as PF, 0.1% of BP for staff club fund. Generate pay slips for the employees with their gross and net salary.

Implementation:

```
import java.util.Scanner;
class Employee
{
    int Emp_id;
    String Emp_name;
    String Address;
    String Mail_Id;
    String Mobile_no;
    Employee(){}
    Employee(int id,String name,String addr,String mail,String mob)
    {
        this.Emp_id=id;
        this.Emp_name=name;
        this.Address=addr;
        this.Mail_Id=mail;
        this.Mobile_no=mob;
    }
}
class Programmer extends Employee
{
    double BP,Gross_salary,Net_salary;
    public Programmer(int id,String name,String addr,String mail,String mob)
    {
        super(id,name,addr,mail,mob);
    }
    void computePay()
    {
        System.out.print("Enter Basic Pay:");
        Scanner input=new Scanner(System.in);
        BP=input.nextDouble();

        double DA,HRA,PF,Fund;
```

```
        DA=(BP*97/100);
        HRA=(BP*10/100);
        PF=(BP*12/100);
        Fund=(BP*0.1/100);
        Gross_salary=BP+DA+HRA;
        Net_salary=BP+DA+HRA-(PF+Fund);
        System.out.println("Emp_ID: "+Emp_id);
        System.out.println("Emp_Name: "+Emp_name);
        System.out.println("Address: "+Address);
        System.out.println("Mail_ID: "+Mail_Id);
        System.out.println("Mobile Number: "+Mobile_no);
        System.out.println("Gross Pay: "+Gross_salary);
        System.out.println("Net Pay: "+Net_salary);
    }
}

class Asst_Professor extends Employee
{
    double BP,Gross_salary,Net_salary;
    public Asst_Professor(int id,String name,String addr,String mail,String mob)
    {
        super(id,name,addr,mail,mob);
    }
    void computePay()
    {
        System.out.print("Enter Basic Pay:");
        Scanner input=new Scanner(System.in);
        BP=input.nextDouble();
        Gross_salary=BP;
        double DA,HRA,PF,Fund;
        DA=(BP*97/100);
        HRA=(BP*10/100);
        PF=(BP*12/100);
        Fund=(BP*0.1/100);
        Net_salary=BP+DA+HRA-(PF+Fund);
        System.out.println("Emp_ID: "+Emp_id);
        System.out.println("Emp_Name: "+Emp_name);
        System.out.println("Address: "+Address);
        System.out.println("Mail_ID: "+Mail_Id);
        System.out.println("Mobile Number: "+Mobile_no);
        System.out.println("Gross Pay: "+Gross_salary);
        System.out.println("Net Pay: "+Net_salary);
    }
}

class Associate_Professor extends Employee
```

```
{
    double BP,Gross_salary,Net_salary;
    public Associate_Professor(int id,String name,String addr,String mail,String mob)
    {
        super(id,name,addr,mail,mob);
    }
    void computePay()
    {
        System.out.print("Enter Basic Pay:");
        Scanner input=new Scanner(System.in);
        BP=input.nextDouble();
        Gross_salary=BP;
        double DA,HRA,PF,Fund;
        DA=(BP*97/100);
        HRA=(BP*10/100);
        PF=(BP*12/100);
        Fund=(BP*0.1/100);
        Net_salary=BP+DA+HRA-(PF+Fund);
        System.out.println("Emp_ID: "+Emp_id);
        System.out.println("Emp_Name: "+Emp_name);
        System.out.println("Address: "+Address);
        System.out.println("Mail_ID: "+Mail_Id);
        System.out.println("Mobile Number: "+Mobile_no);
        System.out.println("Gross Pay: "+Gross_salary);
        System.out.println("Net Pay: "+Net_salary);
    }
}

class Professor extends Employee
{
    double BP,Gross_salary,Net_salary;
    public Professor(int id,String name,String addr,String mail,String mob)
    {
        super(id,name,addr,mail,mob);
    }
    void computePay()
    {
        System.out.print("Enter Basic Pay:");
        Scanner input=new Scanner(System.in);
        BP=input.nextDouble();
        Gross_salary=BP;
        double DA,HRA,PF,Fund;
        DA=(BP*97/100);
        HRA=(BP*10/100);
        PF=(BP*12/100);
```

```
Fund=(BP*0.1/100);
Net_salary=BP+DA+HRA-(PF+Fund);
System.out.println("Emp_ID: "+Emp_id);
System.out.println("Emp_Name: "+Emp_name);
System.out.println("Address: "+Address);
System.out.println("Mail_ID: "+Mail_Id);
System.out.println("Mobile Number: "+Mobile_no);
System.out.println("Gross Pay: "+Gross_salary);
System.out.println("Net Pay: "+Net_salary);
}
}
public class PaySlip
{
    public static void main(String[] args)
    {
        Programmer p=new Programmer(10,"AAA","xxx","aaa_xxx@gmail.com","1111111111");
        System.out.println("----- Programmer -----");
        p.computePay();
        Asst_Professor Ap=new
        Asst_Professor(20,"BBB","yyy","bbb_yyy@gmail.com","2222222222");
        System.out.println("----- Assistant Professor -----");
        Ap.computePay();
        Associate_Professor As=new
        Associate_Professor(30,"CCC","zzz","ccc_zzz@gmail.com","3333333333");
        System.out.println("----- Associate Professor -----");
        As.computePay();
        Professor pf=new
        Professor(40,"DDD","www","ddd_www@gmail.com","444444444444");
        System.out.println("----- Professor -----");
        pf.computePay();
    }
}
```

Output

```
----- Programmer -----
Enter Basic Pay:5000
Emp_ID: 10
Emp_Name: AAA
Address: xxx
Mail_ID: aaa_xxx@gmail.com
Mobile Number: 1111111111
Gross Pay: 10350.0
Net Pay: 9745.0
----- Assistant Professor -----
Enter Basic Pay:10000
```

```
Emp_ID: 20
Emp_Name: BBB
Address: yyy
Mail_ID: bbb_yyy@gmail.com
Mobile Number: 222222222
Gross Pay: 10000.0
Net Pay: 19490.0
----- Associate Professor -----
Enter Basic Pay:15000
Emp_ID: 30
Emp_Name: CCC
Address: zzz
Mail_ID: ccc_zzz@gmail.com
Mobile Number: 3333333333
Gross Pay: 15000.0
Net Pay: 29235.0
----- Professor -----
Enter Basic Pay:20000
Emp_ID: 40
Emp_Name: DDD
Address: www
Mail_ID: ddd_www@gmail.com
Mobile Number: 44444444444
Gross Pay: 20000.0
Net Pay: 38980.0
```

Experiment 4 : Design a Java interface for ADT Stack. Implement this interface using array. Provide necessary exception handling in both the implementations.

Implementation:

Step 1 : We will create an Java interface for stack operations. The interface is as follows –

interface1.java

```
interface interface1
{
    void push(int item);
    int pop();
}
```

Step 2 : The stack implementation using arrays with the help of the above defined interface is as follows –

StackDemo.java

```
import java.io.*;
import java.util.*;
class ArrStack implements interface1
{
```

```
int size;
int[] st;
private int top;
private int items_count;

public ArrStack(int Max)    // constructorx
{
    size = Max;
    st = new int[size];
    top = -1;
    //items_count = 0;
}
public void push(int item)
{
    try{
        st[++top] = item;    // increment top and insert
        //items_count++;    // one more item
    }
    catch(Exception e){
        System.out.println("\n Satck Full!! Can not insert element");
    }
}
public int pop()    // take item from top
{
    int item;
    try{
        item = st[top];
        top--;
        //items_count--;
        return item;
    }
    catch(Exception e){
        System.out.println("\n Satck Empty!!Can not pop the element");
        return -1;
    }
}
public void display()
{
    System.out.println("\n Stack contains:");
    for(int i=top;i>=0;i--)
    {
        if(top== -1)
            System.out.println("Stack is empty");
        else if(size==top)
```

```
        System.out.println("Stack is Full");
    else
        System.out.println(" "+st[i]);
    }
}
}
class StackDemo
{
    public static void main(String[] args)
    {
        ArrStack obj = new ArrStack(10); // stack holds 10 items
        System.out.println("\n-----");
        System.out.println("\n\t Stack using Array");
        System.out.println("\n-----");
        obj.push(10);
        obj.push(20);
        obj.push(30);
        obj.display();
        System.out.println("\n The Popped item is "+obj.pop());
        obj.display();
    }
}
```

Output

```
D:\Prog_Anna>javac StackDemo.java
D:\Prog_Anna>java StackDemo
```

```
-----
      Stack using Array
-----
```

```
Stack contains:
```

```
30
```

```
20
```

```
10
```

```
The Popped item is 30
```

```
Stack contains:
```

```
20
```

```
10
```

Experiment 5 : Write a program to perform string operations using ArrayList. Write functions for the following

a. Append - add at end

b. Insert - add at particular index

c. Search

d. List all string starts with given letter

Java Program

```
import java.util.*;
class Lst
{
    Scanner sc;
    String str;
    public void append(ArrayList AL)
    {
        char ans;
        do
        {
            System.out.println("Enter string: ");
            sc=new Scanner(System.in);
            str=sc.next();
            AL.add(str);
            System.out.println("Do you want to append more string: ");
            ans=sc.next().charAt(0);
        }while(ans=='y');
        System.out.println("The array elements are... "+AL);
        System.out.println("The array size is... "+AL.size());
    }
    public void insert(ArrayList AL)
    {
        System.out.println("Enter string: ");
        sc=new Scanner(System.in);
        str=sc.next();
        System.out.println("Enter index at which the string is to be inserted: ");
        sc=new Scanner(System.in);
        int index=sc.nextInt();
        AL.add(index,str);
        System.out.println("The array elements are... "+AL);
    }
    public void search(ArrayList AL)
    {
        append(AL);
        String searchStr;
        System.out.println("Enter string for searching ");
        sc=new Scanner(System.in);
        searchStr=sc.next();
        boolean found = false;
        Iterator<String> iter = AL.iterator();
        String curItem="";
```

```
int pos=0;
while ( iter.hasNext() == true )
{
    pos=pos+1;
    curItem =(String) iter.next();
    if (curItem.equals(searchStr))
    {
        found = true;
        break;
    }
}
if ( found == false )
{
    pos=0;
}
if (pos!=0)
{
    System.out.println(searchStr + " String Found in position : " + pos);
}
else
{
    System.out.println(searchStr + " String not found");
}
}
public void find(ArrayList AL)
{
    append(AL);
    /*ArrayList to Array of objects Conversion */
    Object [] objs = AL.toArray(new String[0]);
    /*Array of objects to Array of String Conversion*/
    String[] str_list=new String[objs.length];
    System.arraycopy(objs,0,str_list,0,objs.length);
    /*Enter the starting letter */
    System.out.println("Enter starting letter: ");
    sc=new Scanner(System.in);
    String searchChar=sc.next();
    System.out.println("The strings starting with letter "+searchChar+" are...");
    for(int i=0;i<str_list.length;i++)//Iterate through array of strings
    {
        if(str_list[i].startsWith(searchChar))//checking for first letter match
            System.out.println(str_list[i]);//if found display the strings
    }
}
}
```

```
class ArrayListProg
{
    public static void main(String[] args)
    {
        System.out.println("\n\t\t Program for Implementing Array List for List of Strings");
        ArrayList AL=new ArrayList<String>();           //creation of ArrayList
        Lst obj=new Lst();
        char ans;
        do
        {
            System.out.println("Main Menu");
            System.out.println("1. Append \n2. Insert at particular Index \n3. Search \n4. List
                                strings");
            System.out.print("Enter your choice: ");
            Scanner sc=new Scanner(System.in);
            int choice=sc.nextInt();
            switch(choice)
            {
                case 1:obj.append(AL);
                    break;
                case 2:obj.insert(AL);
                    break;
                case 3:obj.search(AL);
                    break;
                case 4:obj.find(AL);
                    break;

            }
            System.out.println("Do you want to go to Main Menu?(y/n): ");
            ans=sc.next().charAt(0);
        }while(ans!='y');
    }
}
```

Output

Program for Implementing Array List for List of Strings

Main Menu

1. Append

2. Insert at particular Index

3. Search

4. List strings

Enter your choice: 1

Enter string:

aaa

Do you want to append more string:

```
y
Enter string:
bbb
Do you want to append more string:
y
Enter string:
ccc
Do you want to append more string:
y
Enter string:
ddd
Do you want to append more string:
n
The array elements are... [aaa, bbb, ccc, ddd]
The array size is... 4
Do you want to go to Main Menu?(y/n):
y
Main Menu
1. Append
2. Insert at particular Index
3. Search
4. List strings
Enter your choice: 2
Enter string:
xxx
Enter index at which the string is to be inserted:
3
The array elements are... [aaa, bbb, ccc, xxx, ddd]
Do you want to go to Main Menu?(y/n):
y
Main Menu
1. Append
2. Insert at particular Index
3. Search
4. List strings
Enter your choice: 3
Enter string for searching
ccc
ccc String Found in position : 3
Do you want to go to Main Menu?(y/n):
y
Main Menu
1. Append
2. Insert at particular Index
3. Search
4. List strings
Enter your choice: 1
Enter string:
bmw
```

Do you want to append more string:

n

The array elements are... [aaa, bbb, ccc, xxx, ddd, bmw]

The array size is... 6

Do you want to go to Main Menu?(y/n):

y

Main Menu

1. Append

2. Insert at particular Index

3. Search

4. List strings

Enter your choice: 4

Enter starting letter:

b

The strings starting with letter b are...

bbb

bmw

Do you want to go to Main Menu?(y/n):

n

Experiment 6 : Write a Java Program to create an abstract class named Shape that contains two integers and an empty method named print Area(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contains only the method print Area () that prints the area of the given shape.

Implementation:

```
import java.util.*;
abstract class Shapes
{
    double a,b;
    abstract void printArea();
}
class Rectangle extends Shapes
{
    void printArea()
    {
        System.out.println("\t\tCalculating Area of Rectangle");
        Scanner input=new Scanner(System.in);
        System.out.print("Enter length: ");
        a=input.nextDouble();
        System.out.print("\nEnter breadth: ");
        b=input.nextDouble();
        double area=a*b;
        System.out.println("Area of Rectangle: "+area);
    }
}
```

```
class Triangle extends Shapes
{
    void printArea()
    {
        System.out.println("\t\tCalculating Area of Triangle");
        Scanner input=new Scanner(System.in);
        System.out.print("Enter height: ");
        a=input.nextDouble();
        System.out.print("\nEnter breadth: ");
        b=input.nextDouble();
        double area=0.5*a*b;
        System.out.println("Area of Triangle: "+area);
    }
}
class Circle extends Shapes
{
    void printArea()
    {
        System.out.println("\t\tCalculating Area of Circle");
        Scanner input=new Scanner(System.in);
        System.out.print("Enter radius: ");
        a=input.nextDouble();
        double area=3.14*a*a;
        System.out.println("Area of Circle: "+area);
    }
}
class AbstractClassDemo
{
    public static void main(String[] args)
    {
        Shapes obj;
        obj=new Rectangle();
        obj.printArea();
        obj=new Triangle();
        obj.printArea();
        obj=new Circle();
        obj.printArea();
    }
}
```

Output

```
D:\>javac AbstractClassDemo.java
```

```
D:\>java AbstractClassDemo
    Calculating Area of Rectangle
```

Enter length: 10

Enter breadth: 20

Area of Rectangle: 200.0

Calculating Area of Triangle

Enter height: 30

Enter breadth: 25

Area of Triangle: 375.0

Calculating Area of Circle

Enter radius: 10

Area of Circle: 314.0

Experiment 7 : Write a Java program to implement user defined exception handling

Implementation : Refer Section 4.6.

Experiment 8 : Write a Java program that reads a file name from the user, displays information about whether the file exists, whether the file is readable or writable, the type of file and the length of the file in bytes.

FileDemo.java

```
import java.util.Scanner;
import java.io.File;
class FileDemo
{
    public static void main(String[] args)
    {
        System.out.println("Enter the name of the file");
        Scanner input=new Scanner(System.in);
        String s=input.nextLine();
        File f1=new File(s);
        System.out.println("-----");
        System.out.println("File Name:"+f1.getName());
        System.out.println("Path:"+f1.getPath());
        System.out.println("Abs Path:"+f1.getAbsolutePath());
        System.out.println("This file is:"+(f1.exists()?"Exists":"Does not exists"));
        System.out.println("Is file:"+f1.isFile());
        System.out.println("Is Directory:"+f1.isDirectory());
        System.out.println("Is Readable:"+f1.canRead());
        System.out.println("IS Writable:"+f1.canWrite());
        System.out.println("Is Absolute:"+f1.isAbsolute());
        System.out.println("File Size:"+f1.length()+"bytes");
        System.out.println("Is Hidden:"+f1.isHidden());
    }
}
```

```
}  
}
```

Output

Enter the name of the file

FileDemo.java

File Name:FileDemo.java

Path:FileDemo.java

Abs Path:D:\Prog_Anna\FileDemo.java

This file is:Exists

Is file:true

Is Directory:false

Is Readable:true

Is Writable:true

Is Absolute:false

File Size:1052bytes

Is Hidden:false

Experiment 9 : Write a java program that implements a multi-threaded application that has three threads. First thread generates a random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

Implementation:

```
import java.util.*;  
class NumberGenerate  
{  
    private int value;  
    private boolean flag;  
    // producer  
    public synchronized void put()  
    {  
        while (flag)  
        {  
            try {  
                wait();  
            } catch (InterruptedException e) { }  
        }  
        flag = true;  
        Random random = new Random();  
        this.value = random.nextInt(10);  
        System.out.println("The generated Number is: "+value);  
        notifyAll();  
    }  
}
```

```
// consumer
public synchronized void get1()
{
    while (!flag) {
        try {
            wait();
        }
        catch (InterruptedException e) { }
    }
    if(value%2==0)
    {
        System.out.println("Second is executing now...");
        int ans=value*value;
        System.out.println(value+" is Even Number and its square is: "+ans);
    }
    flag = false;
    notifyAll();
}

public synchronized void get2()
{
    while (!flag)
    {
        try
        {
            wait();
        } catch (InterruptedException e) { }
    }
    if(value%2!=0)
    {
        System.out.println("Third Thread is executing now...");
        int ans=value*value*value;
        System.out.println(value+" is Odd Number and its cube is: "+ans);
    }
    flag = false;
    notifyAll();
}
}

public class TestNumber
{
    public static void main(String[] args)
    {
        final NumberGenerate obj = new NumberGenerate();

        Thread producerThread = new Thread()
        {
```

```
public void run()
{
    for(int i=1;i<=6;i++)
    {
        System.out.println("Main thread started...");
        obj.put();
        try
        {
            Thread.sleep(1000);
        }catch(InterruptedException e){}
    }
}
};
```

```
Thread consumerThread1 = new Thread()
{
```

```
    public void run()
    {
        for(int i=1;i<=3;i++)
        {
            obj.get1();
            try
            {
                Thread.sleep(2000);
            }
            catch(InterruptedException e){}
        }
    }
};
```

```
Thread consumerThread2 = new Thread()
{
```

```
    public void run() {
        for(int i=1;i<=3;i++)
        {
            obj.get2();
            try
            {
                Thread.sleep(3000);
            }
            catch(InterruptedException e){}
        }
    }
};
```

```
    }  
};  
producerThread.start();  
consumerThread1.start();  
consumerThread2.start();  
}  
}
```

Output

```
Main thread started...  
The generated Number is: 6  
Second is executing now...  
6 is Even Number and its square is: 36  
Main thread started...  
The generated Number is: 4  
Main thread started...  
The generated Number is: 1  
Main thread started...  
The generated Number is: 1  
Third Thread is executing now...  
1 is Odd Number and its cube is: 1  
Main thread started...  
The generated Number is: 0  
Second is executing now...  
0 is Even Number and its square is: 0  
Main thread started...  
The generated Number is: 1  
Third Thread is executing now...  
1 is Odd Number and its cube is: 1
```

Experiment 10 : Write a java program to find the maximum value from the given type of elements using a generic function.

Implementation:

```
import java.util.*;  
public class Test  
{  
    public static <T extends Object & Comparable<? super T>> T max(Collection<?  
extends T> coll)  
    {
```

```
    Iterator<? extends T> MyList = coll.iterator();
    //This statement is used for comparison with the element.
    T element = MyList.next();
    while (MyList.hasNext())
    {
        T next_element = MyList.next();
        if (next_element.compareTo(element) > 0)
            element = next_element;
    }
    return element;
}
public static void main(String[] args)
{
    List<Integer> ints=new ArrayList<Integer>(Arrays.asList(1, 55, 4, 3, 23, 12, 25, 9));
    int max=Collections.max(ints);
    System.out.println(ints);
    System.out.println("maximum Value is "+max);
    List<Character> chars=new ArrayList<Character>(Arrays.asList('a','e','i','o','u'));
    char maxc=Collections.max(chars);
    System.out.println(chars);
    System.out.println("Maximum Value is "+maxc);
}
}
```

Output

```
[1, 55, 4, 3, 23, 12, 25, 9]
maximum Value is 55
[a, e, i, o, u]
Maximum Value is u
```

Experiment 11 : Design a calculator using event-driven programming paradigm of Java with the following options.

a) Decimal manipulations b) Scientific manipulations

Implementation:

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;

public class calci extends JFrame implements ActionListener
{
    Container contentpane;
    JPanel DisplayPanel,ControlPanel;
```

```
JTextField txt;
JButton one,two,three, four, five, six, seven, eight, nine, zero;
JButton plus, min,mul, div, log,CLR,exp;
JButton eq, addSub, dot, memread, memcancel, memplus;
JButton sqrt, sin, cos, tan,onebyx;

double tempnum, tempnextnum, result, ans;
static double ValueInMem;
int num1 = 0,num2 = 0;
int MemPlusFlag = 1, RepeatFlag = 0;
char ch;

calci()
{
    contentpane = getContentPane();
    contentpane.setLayout(new BorderLayout());
    JPanel DisplayPanel = new JPanel();
    txt = new JTextField(30);
    txt.setHorizontalAlignment(SwingConstants.RIGHT);//value is displayed on the
                                                    // right side of display
    /* An event taking care of the fact that user may enter the numbers through
    keyboard*/
    txt.addKeyListener(
        new KeyAdapter()
        {
            public void keyTyped(KeyEvent keyevent)
            {
                char ch = keyevent.getKeyChar();//user can enter the number using

                                                    // keyboard
                if (ch >= '0' && ch <= '9') //do not read other than numbers
                {
                }
                else
                {
                    keyevent.consume();//read the character typed using keyboard
                }
            }
        }
    );
    DisplayPanel.add(txt);
    ControlPanel = new JPanel();

    contentpane.add("Center", ControlPanel);//For Border Layout
    contentpane.add("North", DisplayPanel);
    ControlPanel.setLayout(new GridLayout(7,4,5,5));
    //setting grid layout for buttons
```

```
        memcancel = new JButton("MC");//memory cancel button
        ControlPanel.add(memcancel);
        memcancel.addActionListener(this);//when user clicks it event handler is called

        one = new JButton("1"); //number button
        ControlPanel.add(one);
        one.addActionListener(this);

        two = new JButton("2");
        ControlPanel.add(two);
        two.addActionListener(this);

        three = new JButton("3");
        ControlPanel.add(three);
        three.addActionListener(this);

        memread = new JButton("MR");
        ControlPanel.add(memread);
        memread.addActionListener(this);

        four = new JButton("4");
        ControlPanel.add(four);
        four.addActionListener(this);

        five = new JButton("5");
        ControlPanel.add(five);
        five.addActionListener(this);

        six = new JButton("6");
        ControlPanel.add(six);
        six.addActionListener(this);

        memplus = new JButton("M+");
        ControlPanel.add(memplus);
        memplus.addActionListener(this);

        seven = new JButton("7");
        ControlPanel.add(seven);
        seven.addActionListener(this);

        eight = new JButton("8");
        ControlPanel.add(eight);
        eight.addActionListener(this);
```

```
nine = new JButton("9");
ControlPanel.add(nine);
nine.addActionListener(this);

zero = new JButton("0");
ControlPanel.add(zero);
zero.addActionListener(this);

addSub = new JButton("/-");
ControlPanel.add(addSub);
addSub.addActionListener(this);

dot = new JButton(".");//represents decimal point
ControlPanel.add(dot);
dot.addActionListener(this);

eq = new JButton("=");//get an answer
ControlPanel.add(eq);
eq.addActionListener(this);

plus = new JButton("+");
ControlPanel.add(plus);
plus.addActionListener(this);

min = new JButton("-");
ControlPanel.add(min);
min.addActionListener(this);

mul = new JButton("*");
ControlPanel.add(mul);
mul.addActionListener(this);

div = new JButton("/");
div.addActionListener(this);
ControlPanel.add(div);

sqrt = new JButton("Sqrt");
ControlPanel.add(sqrt);
sqrt.addActionListener(this);

log = new JButton("LOG");
ControlPanel.add(log);
log.addActionListener(this);

sin = new JButton("SIN");
```

```
        ControlPanel.add(sin);
        sin.addActionListener(this);

        cos = new JButton("COS");
        ControlPanel.add(cos);
        cos.addActionListener(this);

        tan = new JButton("TAN");
        ControlPanel.add(tan);
        tan.addActionListener(this);

        exp = new JButton("Exp");
        exp.addActionListener(this);
        ControlPanel.add(exp);

        onebyx = new JButton("1/x");
        onebyx.addActionListener(this);
        ControlPanel.add(onebyx);

        CLR = new JButton("AC");
        ControlPanel.add(CLR);
        CLR.addActionListener(this);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e)
    {
        String s = e.getActionCommand();
        if (s.equals("1"))
        {
            if (RepeatFlag == 0) //Flag represents that this number is entered initially
            {
                txt.setText(txt.getText() + "1");//displaying the value 1 on the display
            }
            else //after performing some operation 1 is clicked for any other operation
            {
                txt.setText(""); //previous contents are cleared
                txt.setText(txt.getText() + "1");//then displaying 1
                RepeatFlag = 0;
            }
        }
        if (s.equals("2"))
        {
            if (RepeatFlag == 0)
            {
```

```
        txt.setText(txt.getText() + "2");
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "2");
        RepeatFlag = 0;
    }
}
if (s.equals("3"))
{
    if (RepeatFlag == 0)
    {
        txt.setText(txt.getText() + "3");
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "3");
        RepeatFlag = 0;
    }
}
if (s.equals("4"))
{
    if (RepeatFlag == 0)
    {
        txt.setText(txt.getText() + "4");
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "4");
        RepeatFlag = 0;
    }
}
if (s.equals("5"))
{
    if (RepeatFlag == 0)
    {
        txt.setText(txt.getText() + "5");
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "5");
        RepeatFlag = 0;
    }
}
```

```
    }  
}  
if (s.equals("6"))  
{  
    if (RepeatFlag == 0)  
    {  
        txt.setText(txt.getText() + "6");  
    }  
    else  
    {  
        txt.setText("");  
        txt.setText(txt.getText() + "6");  
        RepeatFlag = 0;  
    }  
}  
if (s.equals("7"))  
{  
    if (RepeatFlag == 0)  
    {  
        txt.setText(txt.getText() + "7");  
    }  
    else  
    {  
        txt.setText("");  
        txt.setText(txt.getText() + "7");  
        RepeatFlag = 0;  
    }  
}  
if (s.equals("8"))  
{  
    if (RepeatFlag == 0)  
    {  
        txt.setText(txt.getText() + "8");  
    }  
    else  
    {  
        txt.setText("");  
        txt.setText(txt.getText() + "8");  
        RepeatFlag = 0;  
    }  
}  
if (s.equals("9"))  
{  
    if (RepeatFlag == 0)  
    {  
        txt.setText(txt.getText() + "9");  
    }  
}
```

```
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "9");
        RepeatFlag = 0;
    }
}
if (s.equals("0"))
{
    if (RepeatFlag == 0)
    {
        txt.setText(txt.getText() + "0");
    }
    else
    {
        txt.setText("");
        txt.setText(txt.getText() + "0");
        RepeatFlag = 0;
    }
}
if (s.equals("AC")) //This means clear all the contents
{
    txt.setText("");
    num1 = 0;
    num2 = 0;
    RepeatFlag = 0;
}
if (s.equals("/+/-"))
{
    if (num1 == 0)
    {
        txt.setText("-" + txt.getText());
        num1 = 1;
    }
    else
    {
        txt.setText(txt.getText());
    }
}
if (s.equals("."))
{
    if (num2 == 0)
    {
        txt.setText(txt.getText() + ".");
        num2 = 1;
    }
}
```

```
    }
    else
    {
        txt.setText(txt.getText());
    }
}
if(s.equals("+"))
{
    if(txt.getText().equals(""))
    {
        txt.setText("");
        tempnum = 0;
        ch = '+';
    }
    else
    {
        tempnum = Double.parseDouble(txt.getText());
        txt.setText("");
        ch = '+';
        num2 = 0;
        num1 = 0;
    }
    txt.requestFocus();
}
if (s.equals("-"))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
        tempnum = 0;
        ch = '-';
    }
    else
    {
        num1 = 0;
        num2 = 0;
        tempnum = Double.parseDouble(txt.getText());
        txt.setText("");
        ch = '-';
    }
    txt.requestFocus();
}
if (s.equals("/"))
{
    if (txt.getText().equals(""))
    {
```

```
        txt.setText("");
        tempnum = 1;
        ch = '/';
    }
    else
    {
        num1 = 0;
        num2 = 0;
        tempnum = Double.parseDouble(txt.getText());
        ch = '/';
        txt.setText("");
    }
    txt.requestFocus();
}
if (s.equals("*"))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
        tempnum = 1;
        ch = '*';
    }
    else
    {
        num1 = 0;
        num2 = 0;
        tempnum = Double.parseDouble(txt.getText());
        ch = '*';
        txt.setText("");
    }
    txt.requestFocus();
}
if (s.equals("MC"))
{
    ValueInMem = 0;
    txt.setText("");
}
if (s.equals("MR"))
{
    txt.setText("");
    txt.setText(txt.getText() + ValueInMem);
}
if (s.equals("M+"))
{
    if (MemPlusFlag == 1)
    {
```

```
        ValueInMem = Double.parseDouble(txt.getText());
        MemPlusFlag++;
    }
    else
    {
        ValueInMem += Double.parseDouble(txt.getText());
        txt.setText("" + ValueInMem);
    }
}
if (s.equals("LOG"))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
    }
    else
    {
        ans = Math.log(Double.parseDouble(txt.getText()));
        txt.setText("");
        txt.setText(txt.getText() + ans);
    }
}
if (s.equals("1/x"))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
    }
    else
    {
        ans = 1 / Double.parseDouble(txt.getText());
        txt.setText("");
        txt.setText(txt.getText() + ans);
    }
}
if (s.equals("Exp"))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
    }
    else
    {
        ans = Math.exp(Double.parseDouble(txt.getText()));
        txt.setText("");
        txt.setText(txt.getText() + ans);
    }
}
```

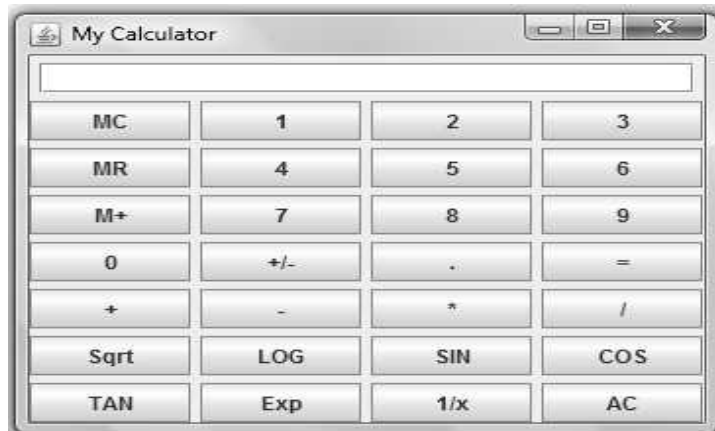
```
    }  
}  
  
if (s.equals("Sqrt"))  
{  
    if (txt.getText().equals(""))  
    {  
        txt.setText("");  
    }  
    else  
    {  
        ans = Math.sqrt(Double.parseDouble(txt.getText()));  
        txt.setText("");  
        txt.setText(txt.getText() + ans);  
    }  
}  
if (s.equals("SIN"))  
{  
    if (txt.getText().equals(""))  
    {  
        txt.setText("");  
    }  
    else  
    {  
        ans = Math.sin(Double.parseDouble(txt.getText()));  
        txt.setText("");  
        txt.setText(txt.getText() + ans);  
    }  
}  
if (s.equals("COS"))  
{  
    if (txt.getText().equals(""))  
    {  
        txt.setText("");  
    }  
    else  
    {  
        ans = Math.cos(Double.parseDouble(txt.getText()));  
        txt.setText("");  
        txt.setText(txt.getText() + ans);  
    }  
}  
if (s.equals("TAN"))  
{  
    if (txt.getText().equals(""))  
    {
```

```
        txt.setText("");
    }
    else
    {
        ans = Math.tan(Double.parseDouble(txt.getText()));
        txt.setText("");
        txt.setText(txt.getText() + ans);
    }
}
if (s.equals("="))
{
    if (txt.getText().equals(""))
    {
        txt.setText("");
    }
    else
    {
        tempnextnum = Double.parseDouble(txt.getText());
        switch (ch)
        {
            case '+':
                result = tempnum + tempnextnum;
                break;
            case '-':
                result = tempnum - tempnextnum;
                break;
            case '/':
                result = tempnum / tempnextnum;
                break;
            case '*':
                result = tempnum * tempnextnum;
                break;
        }
        txt.setText("");
        txt.setText(txt.getText() + result);
        RepeatFlag = 1;
    }
}
txt.requestFocus();
}

public static void main(String args[])
{
    calci c= new calci();
    c.setTitle("My Calculator");
    c.pack();
}
```

```
        c.setVisible(true);  
    }  
}
```

Output



□□□

December - 2018
Object Oriented Programming
Semester - III (CSE/IT) - Regulation 2017

AU
Solved Paper
25061

Time : Three Hours]

[Maximum Marks : 100

Answer ALL questions.

PART A - (10 × 2 = 20 Marks)

- Q.1** Define objects and classes in java. (Refer Two Marks Q.8 of Chapter-1)
- Q.2** Define access specifier. (Refer Two Marks Q.35 of Chapter-1)
- Q.3** What is object cloning ? (Refer Two Marks Q.5 of Chapter-3)
- Q.4** What is class hierarchy ? Give example. (Refer Two Marks Q.9 of Chapter-2)
- Q.5** Define runtime exceptions. (Refer Two Marks Q.17 of Chapter-4)
- Q.6** What is the use of assert keyword ? (Refer Two Marks Q.18,Chapter-4)
- Q.7** What is multithreading ? (Refer Two Marks Q.6 of Chapter-6)
- Q.8** What is the need for generic code ? (Refer Two Marks Q.1 Chapter-7)
- Q.9** What is meant by window adapter classes ?
(Refer Two Marks Q.6 of Chapter-9)
- Q.10** Enumerate the features of AWT in Java.
(Refer Two Marks Q.7 of Chapter-8)

PART B - (5 × 13 = 65 Marks)

- Q.11 a)** i) Explain the characteristics of OOPs. (Refer section 1.1) [6]
ii) Explain the features and the characteristics of JAVA. (Refer section 1.4) [7]

OR

- b)** i) What is method ? How method is defined ? Give example. (Refer section 1.10) [6]
ii) State the purpose of finalize () method in java ? With an example explain how finalize () method can be used in java program. (Refer section 2.12) [7]
- Q.12 a)** Define inheritance. With diagrammatic illustration and java programs illustrate the different types of inheritance with an example. (Refer section 2.5)

OR

- b) Write a Java program to create a student examination database system that prints the mark sheet of students. Input student name, marks in 6 subjects. This mark should be between 0 and 100.

If the average of marks is ≥ 80 then prints Grade 'A'. (Refer example 1.11.8)

If the average is < 80 and ≥ 60 then prints Grade 'B'.

If the average is < 60 and ≥ 40 then prints Grade 'C'.

else prints Grade 'D'. [13]

- Q.13 a)** Explain the different types of exceptions and the exception hierarchy in java with appropriate examples. (Refer section 4.2) [13]

OR

- b) What are input and output streams ? Explain them with illustrations. (Refer section 5.2) [13]

- Q.14 a)** Explain in detail the different states of a thread. (Refer section 6.3) [13]

OR

- b) Demonstrate inter thread communication and suspending, resuming and stopping threads. (Refer section 6.7) [13]

- Q.15 a)** State and explain the basic of AWT event handling in detail. (Refer section 9.2) [13]

OR

- b) Describe in detail about the different layout in Java GUI. Which layout is the default one ? (Refer section 10.2) [13]

PART C - (1 × 15 = 15 Marks)

- Q.16 a)** Create a Bank database application program to illustrate the use of multi-threads. (Refer example 6.5.2)

OR

- b) Code a java program to implement the following : Create four check boxes. The initial state of the first box should be in checked state. The status of each check box should be displayed. When we changes the state of a check box, the status should be display is updated. (Refer example 10.8.1)

□□□

<p style="text-align: center;">May - 2019</p> <p style="text-align: center;">Object Oriented Programming</p> <p style="text-align: center;">Semester - III (CSE/IT) - Regulation 2017</p>	<p style="text-align: center;">AU</p> <p style="text-align: center;">Solved Paper</p> <p style="text-align: center;">80096</p>
---	---

Time : Three Hours]

[Maximum Marks : 100

Answer ALL questions.

PART A - (10 × 2 = 20 Marks)

- Q.1** *Can a Java source file be saved using a name other than the class name ? Justify.*
(Refer Two Marks Q.37 of Chapter - 1)
- Q.2** *What are inline functions ? Give examples.*
(Refer Two Marks Q.38 of Chapter - 1)
- Q.3** *State the conditions for method overriding in JAVA.*
(Refer Two Marks Q.16 of Chapter - 2)
- Q.4** *Write the syntax for importing packages in a Java source file and give an example.*
(Refer Two Marks Q.34 of Chapter - 1)
- Q.5** *What happens when the statement `int value = 25/0` ; is executed ?*
(Refer Two Marks Q.20 of Chapter - 4)
- Q.6** *Give an example of reading data from files using file input stream.*
(Refer Two Marks Q.14 of Chapter - 5)
- Q.7** *Sketch the lifecycle of a thread. (Refer Two Marks Q.15 of Chapter - 6)*
- Q.8** *Give the syntax of a generic class with an example.*
(Refer Two Marks Q.3 of Chapter - 7)
- Q.9** *Write the class hierarchy for panel and frame.*
(Refer Two Marks Q.8 of Chapter - 8)
- Q.10** *State the purpose of `getRed()`, `getBlue()` and `getGreen()` methods.*
(Refer Two Marks Q.9 of Chapter - 8)

PART B - (5 × 13 = 65 Marks)

- Q.11 a)** *i) Discuss the three OOP principles in detail. (Refer section 1.4)* [7]
 ii) What are literals ? Explain the types of literals supported by java.
 (Refer section 1.7) [6]

OR

- b)** *i) Explain the selection statements in Java using suitable examples.*
 (Refer section 1.8) [7]

ii) Write a Java code using do - while loop that counts down to 1 from 10 printing exactly ten lines of "hello". **(Refer example 1.8.7)** [6]

Q.12 a) Explain hierarchical and multi-level inheritances supported by Java and demonstrate the execution order of constructors in these types. **(Refer section 2.6)** [13]

OR

b) i) Explain simple interfaces and nested interfaces with examples. **(Refer section 3.3)** [7]

ii) Present a detailed comparison between classes and interfaces. **(Refer section 3.1)** [6]

Q.13 a) i) Give an example for nested try statements in Java source file and explain. **(Refer section 4.4)** [7]

ii) Write a note on built - in exceptions. **(Refer section 4.5)** [6]

OR

b) Create an IN file in java to store the details of 100 students using a STUDENT class. Read the details from IN file, convert all the letters in IN file to lowercase letters and write it into OUT file. **(Refer example 5.4.7)** [13]

Q.14 a) Describe the creation of single thread and multiple threads using an example. **(Refer section 6.5)** [13]

OR

b) i) Using an example, explain inter-thread communication in Java. **(Refer section 6.7)** [7]

ii) Write a generic method for sorting an array of integer objects. **(Refer example 7.2.1)** [6]

Q.15 a) i) Use graphics objects to draw an arc and a semicircle inside a rectangular box. **(Refer example 8.4.1)** [4]

ii) Sketch the hierarchy of Java AWT classes and methods. Create a 'check box' using these classes and methods. **(Refer section 8.3)** [9]

OR

b) i) State the differences between AWT and swing. **(Refer section 10.1)** [4]

ii) Present the hierarchy of java swing classes and methods of component class. Create a simple 'combo box' in java swing using the classes and methods. **(Refer section 10.11)** [9]

PART C - (1 × 15 = 15 Marks)

- Q.16 a)** *The following is a system that can be used to synchronize threads. In some shops a machine issues numbered tickets to customers and customers are served numeric order.*

A ticket machine holds an integer, initially zero and has a single atomic operation : turn () - which increments the integer and returns its previous value.

A scheduler also holds an integer, initially zero and has two atomic operations :

next () - which increments the integer count

queue(value) - suspends the calling thread until the count is atleast as large as the value given as an argument.

Given a ticket machine, m and a scheduler s, a critical region could then be coded as follows :

number = m. turn();

s.queue(number);

. protected code

.

s.next();

Write Java classes ticket machine, with a turn method and scheduler, with next and queue methods to implement the system.

OR

- b)** *Define Java classes of your own without using any library classes to represent linked lists of integers. Provide it with methods that can be used to reverse a list and to append two lists. (Refer example 1.14.8)*



Q.15 a) Describe in detail about the different layouts in Java GUI. Which layout is the default one ? (Refer section 10.2)

OR

b) Discuss mouse listener and mouse motion listener. Give an example program. (Refer section 9.5)

PART C - (1 × 15 = 15 Marks)

Q.16 a) Develop a java program to find a smallest number in the given array by creating one dimensional array and two dimensional array using new operator. (Refer example 1.14.7)

OR

b) Create a simple real life application program in Java to illustrate the use of multithreads. (Refer section 6.5.2)

□□□

Notes