THIRD YEAR B. Sc.
COMPUTER SCIENCE

SEMESTER-V

NEW SYLLABUS
CBCS PATTERN

OBJECT ORIENTED PROGRAMMING USING JAVA–I

Dr. Ms. MANISHA BHARAMBE
Ms. MANISHA GADEKAR

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

OBJECT ORIENTED PROGRAMMING USING JAVA - I

For T.Y.B.Sc. Computer Science : Semester – V

[Course Code CS 355 : Credits - 2]

CBCS Pattern

As Per New Syllabus, Effective from June 2021

Dr. Ms. Manisha Bharambe

M.Sc. (Comp. Sci.), M.Phil. Ph.D. (Comp. Sci.)

Vice Principal, Associate Professor, Department of Computer Science

MES's Abasaheb Garware College

Pune

Ms. Manisha Gadekar

M.C.A., (Science), UGC-NET

Assistant Professor, Annasaheb Magar College

Pune

Price ₹ 600.00

NIRALI PRAKASHAN

ADVANCEMENT OF KNOWLEDGE

N5865

OBJECT ORIENTED PROGRAMMING USING JAVA - I **ISBN 978-93-5451-188-2**

Second Edition : September 2022

© : Authors

The text of this publication, or any part thereof, should not be reproduced or transmitted in any form or stored in any computer storage system or device for distribution including photocopy, recording, taping or information retrieval system or reproduced on any disc, tape, perforated media or other information storage device etc., without the written permission of Authors with whom the rights are reserved. Breach of this condition is liable for legal action.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, errors may have crept in. Any mistake, error or discrepancy so noted and shall be brought to our notice shall be taken care of in the next edition. It is notified that neither the publisher nor the authors or seller shall be responsible for any damage or loss of action to any one, of any kind, in any manner, therefrom. The reader must cross check all the facts and contents with original Government notification or publications.

Published By :
NIRALI PRAKASHAN
Abhyudaya Pragati, 1312, Shivaji Nagar,
Off J.M. Road, Pune – 411005
Tel - (020) 25512336/37/39
Email : niralipune@pragationline.com

Polyplate

Printed By :
YOGIRAJ PRINTERS AND BINDERS
Survey No. 10/1A, Ghule Industrial Estate
Nanded Gaon Road
Nanded, Pune - 411041

DISTRIBUTION CENTRES

PUNE

Nirali Prakashan
(For orders outside Pune)
S. No. 28/27, Dhayari Narhe Road, Near Asian College
Pune 411041, Maharashtra
Tel : (020) 24690204; Mobile : 9657703143
Email : bookorder@pragationline.com

Nirali Prakashan
(For orders within Pune)
119, Budhwar Peth, Jogeshwari Mandir Lane
Pune 411002, Maharashtra
Tel : (020) 2445 2044; Mobile : 9657703145
Email : niralilocal@pragationline.com

MUMBAI

Nirali Prakashan
Rasdhara Co-op. Hsg. Society Ltd., 'D' Wing Ground Floor, 385 S.V.P. Road
Girgaum, Mumbai 400004, Maharashtra
Mobile : 7045821020, Tel : (022) 2385 6339 / 2386 9976
Email : niralimumbai@pragationline.com

DISTRIBUTION BRANCHES

DELHI

Nirali Prakashan
Room No. 2 Ground Floor
4575/15 Omkar Tower, Agarwal Road
Darya Ganj, New Delhi 110002
Mobile : 9555778814/9818561840
Email : delhi@niralibooks.com

BENGALURU

Nirali Prakashan
Maitri Ground Floor, Jaya Apartments,
No. 99, 6th Cross, 6th Main,
Malleswaram, Bengaluru 560003
Karnataka; Mob : 9686821074
Email : bengaluru@niralibooks.com

NAGPUR

Nirali Prakashan
Above Maratha Mandir, Shop No. 3,
First Floor, Rani Jhanshi Square,
Sitabuldi Nagpur 440012 (MAH)
Tel : (0712) 254 7129
Email : nagpur@niralibooks.com

KOLHAPUR

Nirali Prakashan
New Mahadvar Road, Kedar Plaza,
1st Floor Opp. IDBI Bank
Kolhapur 416 012 Maharashtra
Mob : 9850046155
Email : kolhapur@niralibooks.com

JALGAON

Nirali Prakashan
34, V. V. Golani Market, Navi Peth,
Jalgaon 425001, Maharashtra
Tel : (0257) 222 0395
Mob : 94234 91860
Email : jalgaon@niralibooks.com

SOLAPUR

Nirali Prakashan
R-158/2, Avanti Nagar, Near Golden
Gate, Pune Naka Chowk
Solapur 413001, Maharashtra
Mobile 9890918687
Email : solapur@niralibooks.com

marketing@pragationline.com | www.pragationline.com

Also find us on www.facebook.com/niralibooks

Preface ...

We take an opportunity to present this Text Book on **"Object Oriented Programming Using Java - I"** to the students of Third Year B.Sc. (Computer Science) Semester-V as per the New Syllabus, June 2021.

The book has its own unique features. It brings out the subject in a very simple and lucid manner for easy and comprehensive understanding of the basic concepts. The book covers theory of An Introduction to Java, Objects and Classes, Inheritance and Interface, Exception and File Handling and User Interface with AWT and Swing.

A special word of thank to Shri. Dineshbhai Furia, and Mr. Jignesh Furia for showing full faith in us to write this text book. We also thank to Mr. Amar Salunkhe and Mr. Akbar Shaikh of M/s Nirali Prakashan for their excellent co-operation.

We also thank Ms. Chaitali Takle, Mr. Ravindra Walodare, Mr. Sachin Shinde, Mr. Ashok Bodke, Mr. Moshin Sayyed and Mr. Nitin Thorat.

Although every care has been taken to check mistakes and misprints, any errors, omission and suggestions from teachers and students for the improvement of this text book shall be most welcome.

Authors

■■■

Syllabus ...

1. An Introduction to Java (6 Lectures)

- Object Oriented Programming Concepts
- A Short History of Java
- Features or Buzzwords of Java
- Java Environment
- Simple Java Program
- Java Tools – jdb, javap, javadoc
- Types of Comments
- Data Types
- Final Variable
- Declaring 1D, 2D Array
- Accepting Input (Command Line Arguments, BufferedReader, Scanner)

2. Objects and Classes (7 Lectures)

- Defining Your Own Classes
- Access Specifiers - (public, protected, private, default)
- Array of Objects
- Constructors, Overloading Constructors and Use of 'this' Keyword, Static Block, Static Fields and Methods
- Predefined Classes
  - Object Class, Methods (equals(), toString(), hashcode(), getClass())
  - String Class and StringBuffer Class, Formatting String Data using format() Method
- Creating, Accessing and Using Packages
- Wrapper Classes

3. Inheritance and Interface (8 Lectures)

- Inheritance Basics (extends Keyword) and Types of Inheritance
- Superclass, Subclass and Use of Super Keyword
- Method Overriding and Runtime Polymorphism
- Use of Final Keyword related to Method and Class
- Use of Abstract Class and Abstract Methods
- Defining and Implementing Interfaces
- Runtime Polymorphism using Interface
- Concept of Marker and Functional Interfaces

4. Exception and File Handling (5 Lectures)

- Dealing with Errors, Exception Class, Checked and Unchecked Exception
- Catching Exceptions, Multiple Catch Block, Nested Try Block
- Creating User Defined Exception
- Introduction to Files and Streams
- Input-OutputStream: FileInput/OutputStream, BufferedInput/OutputStream, DataInput/OutputStream
- Reader-Writer: FileReader/Writer, BufferedReader/Writer, InputStreamReader, OutputStreamWriter

5. User Interface with AWT and Swing (10 Lectures)

- What is AWT? What is Swing? Difference between AWT and Swing
- The MVC Architecture and Swing
- Layouts and Layout Managers
- Containers and Components – JFrame, JButton, JLabel, JText, JTextArea, JCheckBox and JRadioButton, JList, JComboBox, JMenu and related Classes
- Dialogs (Message, Confirmation, Input), JFileChooser, JColorChooser
- Event Handling: Event Sources, Listeners
- Adapters and Anonymous Inner Class

■■■

Contents ...

1. **An Introduction to Java** **1.1 – 1.104**

2. **Objects and Classes** **2.1 – 2.92**

3. **Inheritance and Interface** **3.1 – 3.62**

4. **Exception and File Handling** **4.1 – 4.72**

5. **User Interface with AWT and Swing** **5.1 – 5.126**

■■■

CHAPTER 1

An Introduction to Java

Objectives...

- To understand Basic Concepts of Java Programming
- To learn Java Environment
- To study Java Tools
- To study Arrays in Java

1.0 INTRODUCTION

- Java is a general-purpose, high level, robust, Object-Oriented Programming (OOP) language developed by Sun Microsystems in 1991.
- OOP is a programming paradigm that uses objects and their interactions to design applications and computer programs.
- A programming paradigm describes the structure of a program. In other words, it determines how the instructions are placed in a program.
- Java programming language was initiated by James Gosling. He is called father of Java programming.
- The first version of Java 1.0 was released by Sun Microsystems in 1995. It made the promise of "Write Once, Run Anywhere (WORA)", with free (no cost) runtimes on popular platforms such as Enterprise applications.
- With the advancement of Java and its widespread popularity, use new enhanced versions of Java have been released. The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively.
- Java Platform, Standard Edition (Java SE) is a computing platform for development and deployment of portable code for desktop and server environments. Java SE was formerly known as Java 2 Platform, Standard Edition (J2SE).
- Java Platform, Micro Edition (Java ME) is a computing platform for development and deployment of portable code for embedded and mobile devices (micro-controllers, sensors, mobile phones, Personal Digital Assistants (PDAs etc.)). Java ME was formerly known as Java 2 Platform, Micro Edition (J2ME).
- Java Platform, Enterprise Edition (Java EE) is a computing platform, extending Java SE with specifications for enterprise features such as distributed computing and web services. Java EE was formerly known as Java 2 Platform, Enterprise Edition (J2EE).

- The latest release/version of the Java programming language is Java SE 16. Java 16 contains new features, enhancements and bug fixes to improve efficiency to develop and run Java programs.
- Java programming language is mainly used to create these following four types of applications:
  1. **Web Applications:** Web application is an application which creates a dynamic page and runs on the side of server. At present time, technologies that are used to create web application in Java programming language are Struts, Spring, Servlet, Hibernate, JSP, JSF, etc.
  2. **Mobile Applications:** Mobile application refers to application that is developed for mobile devices. At present time, platforms that are used to develop mobile applications in Java programming language are Java ME and Android.
  3. **Standalone Applications:** Desktop applications and Window based applications are the other name of standalone applications. We need to install this traditional software on every machine. Media player and antivirus are the examples of these applications. At current time, for the creation of standalone applications AWT and swing are used in Java programming language.
  4. **Enterprise Applications:** Enterprise applications are those applications which are distributed in nature. Banking application is the example of enterprise applications. It gives programmer the advantage of high level security, clustering and load balancing. At present time, EJB is used in Java programming language for developing enterprise applications.

1.1 OBJECT ORIENTED PROGRAMMING CONCEPTS [April 17]

- Java is an Object Oriented Programming (OOP) language. The fundamental idea behind an OOP is to combine both data and the functions that operate on the data into a single unit i.e., an object.
- An approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand is known as Object Oriented Programming (OOP).
- The popular object-oriented languages include Simula, Ada, Smalltalk, Java, C#, PHP, Python, C++ and so on.
- Java is a high-level programming language designed and developed to accommodate object-oriented concepts like polymorphism, class, object, abstraction, encapsulation, inheritance, message passing and so on.

Features of OOP:

1. OOP emphasis on data rather than procedure.
2. OOP follows bottom-up approach in program design.
3. Programs in OOP are divided into objects.
4. Objects in OOP may communicate with each other through functions.

5. Objects in OOP are characterized by functions or data structure that operates on data.
6. In OOP new data and functions can be easily added whenever necessary.
7. In OOP data is hidden and cannot be accessed by external functions.

- The fundamental concepts of object oriented programming are explained below:

1. Classes:

- A class is a blueprint/template from which individual objects are created. A class is a collection of objects of similar type of objects.
- Classes are user defined data type. Once, class has been defined, we can create any number of object belonging to that class.
- A class consists of data and methods. A method is a collection of statements that are group together to perform an operation.
- Following example define class Person with data and methods:

```
class Person
{
   String name;                    //data
   int age;
   void accept( String n, int a)  // methods
   {
      name=n;
      age=a;
   }
   void display()
   {
      System.out.println("Name="+name);
      System.out.println("Age="+age);
   }
}
```

2. Objects:

- Objects are basic run time entities in OOP. Objects may represent a real-world entity such as a book, a chair, a place, a bank account, a table etc.
- An object is an instance of a class. Objects are variables of class because it is created by the class data type.
- An object has two general characteristics namely, state and behavior. An **object's state** represents what the object knows. The state is maintained in one or more variables associated with the object.
- An **object's behavior** represents what the object can do. The behavior is implemented by defined methods that the object can access. Simply put, a method is a named block of code that does something.

- An object has identity which differentiates one object from the other. Object name is used to identify the object. Hence, name itself is an identity.
- Objects are generally used to access members (data and methods) of class. **For example:** `Person p = new Person();`. In this statement 'p' is an object of Person class.
- Fig. 1.1 shows example of class and objects. Fig. 1.1 shows class Car has objects like Audi, Nissan, Volvo.

Fig. 1.1: Example of Class and Object

Comparison between Object and Class:

Sr. No.	Object	Class
1.	Object is an instance of a class.	Class is a blueprint or template from which objects are created.
2.	Object is a real world entity such as a pen, a laptop, a keyboard, etc.	Class is a group of similar type objects.
3.	Object is a physical entity.	Class is a logical entity.
4.	Mainly Object is created through new keyword like `Student s1=new Student();`	Class is declared using class keyword like `class Student{ … }`
5.	Object is created many times as per requirement.	Class is declared or created once.
6.	Object allocates memory when it is created.	Class doesn't allocated memory when it is created.

3. Data Abstraction:

- Data abstraction refers to act of representing essential features without including background details or explanations.
- Classes use the concept of abstraction. They encapsulate all essential properties of objects that they are to be created.
- Since, the class used the concept of data abstraction they are known as Abstract Data Types (ADT).

4. Data Encapsulation:

- Data encapsulation means wrapping or binding of data and functions into a single unit.

- The data is not accessible to the outside world and only functions which are wrapped in the class can access it.
- Thus, insulation of data from direct access from program is called data hiding.

5. Inheritance:

- Inheritance is the process by which object of one class can acquire the properties of the objects of another class.
- Inheritance means one class of object inherits the data and behavior (methods) from another class.
- The old class is referred as a base class (parent class or super class) and new class is called as derived class (child class or sub class). The new class has combined features of both the classes.
- Fig. 1.2 shows a class called Vehicle, which has two subclasses called Car and Truck. The Vehicle class is the superclass of Car and Truck.

Fig. 1.2: Classes Car and Truck inherits from the class Vehicle

6. Polymorphism:

- Polymorphism is the ability to take more than one form. In short, polymorphism is the ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
- **For example:** An operation may demonstrate different behavior in different instance. The behavior depends upon the type of data used in the operation. As addition of two integer numbers give addition while addition of two strings give concatenation.

7. Dynamic Binding:

- Binding refers to the linking of a procedure call to the procedure definition (code) to be executed in response to the call.
- Binding is classified as static binding and dynamic binding as explained below:
  - **(i) Static Binding** means that the code associated with given procedure call is known at compile time.
  - **(ii) Dynamic Binding** means that the code associated with given procedure call is not known until the time of the call at run time. Dynamic binding associated with polymorphism and inheritance.

8. Message Passing:

- In OOP objects are communicate with each other through message passing. A message for an object is a request for execution of a method.

- Message passing involves specifying the name of object, the name of the method and the information to be sent.
- **For example:** `p.accept("abc",18);` here, p is an object of Person class, accept is message and abc, 18 are arguments is information.

Advantages of OOP:

1. Software is easily developed for complex problems.
2. Using OOP, software reuse is enhanced.
3. Using OOP, software maintenance cost can be reduced.
4. Modularity is achieved using OOP.
5. Software has improved performance.
6. Software quality is improved.
7. Data access is restricted providing better data security.

Disadvantages of OOP:

1. OOP requires intensive testing procedures.
2. Solving a problem using OOP approach consumes more time than the time taken by structured.

1.2 A SHORT HISTORY OF JAVA

- In 1990, Sun Micro Systems Inc. (US) was conceived a project to develop software for consumer electronic devices that could be controlled by a remote. This project was called Stealth Project but later its name was changed to Green Project.
- Java was developed at Sun Microsystems in 1991, by a team (also called as Green Team) comprising James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan as its members to initiated the project (Green Project) to develop a language for digital devices such as set-top boxes, televisions, etc. in the early 1990s.
- Java was originally designed for small and embedded systems in digital appliances such as television and set top boxes, etc.
- But later it suits internet programming and now it is used in mobile applications, enterprise applications, windows applications, etc.
- James Gosling is known as the father of Java. At first James Gosling called it "Greentalk" but later on the project was developed as green project's part and it was named "Oak".
- It was later termed as Java. Sun Microsystems released the first public implementation as Java 1.0 in 1996.
- The Java software was released as a development kit. The first two versions were named JDK 1.0 and JDK 1.1.
- In 1998, while releasing the next version, Sun Microsystems changed the nomenclature from Java Development Kit (JDK) to Software Development Kit (SDK). Also, it added "2" to the name. The released version of Java was called Java 2 SDK 1.2.

- With every version, Java became stronger and stronger. At the time of writing of the book, Java had come up with its version SE 8 in April 2014.
- Table 1.1 illustrates the various versions released of Java language, till date.

Table 1.1: History of Java

Version	Release Date	Major Additions
JDK 1.0	January 1996	Initial release
JDK 1.1	February 1997	Inner classes, JavaBeans JDBC, RMI
J2SE 1.2	December 1998	Swing and Collections Framework
J2SE 1.3	May 2000	HotSpot JVM, RMI, JavaSound
J2SE 1.4	February 2002	Regular expressions, Java Web Start
J2SE 5.0	September 2004	Generics, autoboxing, enumeration
J2SE SE 6	December 2006	Database manager and many new facilities.
JAVA SE 7	July 2011	New invoked dynamic bytecode under JSR-292
JAVA SE 8	March 2014	JDK Enhancement Proposals (JEPS).
JAVA SE 9	September, 2017	Fix some open issues and concerns, and to refine some critical technical questions.
JAVA SE 10	March, 2018	Security fixes and critical bug fixes.
JAVA SE 11	September, 2018	Open for bug fixes. It offers LTS(Long Term Support).
JAVA SE 12	March, 2019	Add new features, bug fixes.
JAVA SE 13	September, 2019	Text block, bug fixes and security.
JAVA SE 14	March, 2020	Pattern matching for instanceof record, Security and Bug fixes.
Java SE 15	September, 2020	Security and bug fixes.
Java SE 16	March, 2021	Timezone database update, bug fixes, and security updates.

1.3 FEATURES OR BUZZWORDS OF JAVA [April 16, 17, Oct. 17]

- There are various features of Java as given in Fig. 1.3. Each feature of Java will tell us about how and why Java is a powerful object-oriented programming language.

Fig. 1.3: Features/Buzzwords of Java

- Let us see Java features in detail:
  1. **Simple:** Java is a simple language. There are various concepts that make the Java as a simple language. Java is designed to be easy to learn. Programs in Java are easy to write and debug because Java does not use the pointers, preprocessor header files, operator overloading etc.
  2. **Object-oriented:** Java is object-oriented language like C++, Python etc. Almost everything in Java language is an object based. All program code and data in Java reside within objects and classes. Java language can be easily extended since it is based on the Object model.
  3. **Platform Independent:** Java Language is platform independent language. A platform is the hardware or software environment in which a program runs. Java code/program can be run on multiple platforms like Windows, Linux, Sun Solaris, and Mac Os etc. **[Oct. 18]**
  4. **Interpreted:** Java is an interpreted language, i.e. programs run directly from the source code. Java interpreter can execute Java bytecode directly on any computer machine. This machine should have the interpreter ported on it. Generally, we know that interpreters are very slow. But in Java, the interpreter uses the compiled code called bytecode, so it is faster than typical interpreter.

     There are compilers available for bytecode called JIT i.e. Just-In Time compilers. These compilers are used to compile the bytecode into native code. So they improve the speed.

5. **Distributed:** Java facilitates the building of distributed application by a collection of classes for use in networked applications. Java is designed for the distributed environment of the Internet.
6. **Dynamic:** Java language is capable of dynamically linking in new class libraries, methods and objects. Because Java is interpreted, Java is an extremely dynamic language, at runtime, the Java environment can extends itself by linking in classes that may be located on remote servers on a network, (for example; the Internet).
7. **Robust:** Java is a robust language because it provides many safeguards to ensure reliable code. Java does not allow pointer which avoids security problem. There is no concept of reference variable. This eliminates the possibility of overwriting memory and corrupting data. Automatic garbage collection eliminates memory leaks.
8. **Secure:** Java is aimed to be used in networked or distributed environments. Java does not allocate direct pointer to memory, this makes it impossible to accidentally reference memory that belongs to other program.
9. **Architecture Neutral:** Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors, with the presence of Java runtime system.
10. **Portable:** Being architectural-neutral and having no implementation dependent aspects of the specification makes Java portable. Java code is portable. It was an important design goal of Java that it be portable so that as new architectures (due to hardware, operating system, or both) are developed, the Java environment could be ported to them.

    Additionally, Java is portable because the compiler itself is written in Java. Compiler in Java is written in ANSI C with a clean portability boundary which is a POSIX subset.
11. **Multithreading:** A thread is like a separate program, executing concurrently. Multithreaded means handling multiple tasks simultaneously. With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.
12. **High Performance:** With the use of Just-In-Time (JIT) compilers, Java enables high performance.
13. **Interactive:** Java is called as interactive because code of Java supports CGI (Common Gateway Interface) and GUI (Graphical User Interface) programs.

1.4 JAVA ENVIRONMENT

- Java environment includes a large number of development tools [part of system known as JDK (Java Development Kit)] and hundreds of classes and methods [(part of the JSL (Java Standard Library)].

- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications.
- JSL also known as the Application Programming Interface (API). The JSL (or API) includes hundreds of classes and methods grouped into several functional packages.
- Fig. 1.4 depicts a typical Java environment.

Fig. 1.4: Java Environment

- The Java source code is stored as a '.java' file and is compiled by Java compiler (javac) to get an executable '.class' file. Java compiler translates source code into byte code instructions (.class file). The byte code is executed by JVM (Java Virtual Machine) which is a runtime environment for Java.
- A ClassLoader is use to download the executable file, the bytecode (instruction set for the Java Virtual Machine (JVM)) of which is then interpreted, a process which makes the file understood by the underlying processor which executes the file instructions.
- The interpretation process carried by interpreter which reads the instructions and then executes the instructions, step by step. After the bytecode of a class is loaded by the class loader, it has to be inspected by the Java bytecode verifier.
- Byte code verifier verifies the bytecode so that the code does not make any sort of disturbance for the interpreter. Byte code verifier checks the code fragments for illegal codes that can violate access right to the object.
- Java uses the Just-In-Time (JIT) compiler with the interpreter, enables all the frequently used class files to be compiled into native code just the once, rather than each time they are used.
- The API provided by the language is constantly available to the JVM, and the entire package, excepting the compiler that generates the .class file from the .java file, is referred to as the Java Runtime Environment (JRE).
- JRE is an acronym for Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment.

Java Virtual Machine (JVM):

- A JVM is a virtual machine that enables a computer to run Java programs as well as programs written in other languages that are also compiled to Java bytecode.
- A Java virtual machine's main job is to load class files and execute the byte codes they contain.
- Fig. 1.5 shows the Java virtual machine contains a class loader, which loads class files from both the program and the Java API.
- Only those class files from the Java API that are actually needed by a running program are loaded into the virtual machine. The byte codes are executed in an execution engine.
- The execution engine of JVM is responsible for executing the instructions contained in the methods of loaded classes.
- When a Java virtual machine runs a program, it needs memory to store many things, including byte codes and other information it extracts from loaded class files, objects the program instantiates, parameters to methods, return values, local variables, and intermediate results of computations.
- The Java virtual machine organizes the memory it needs to execute a program into several runtime data areas.

Fig. 1.5: Java Virtual Machine (JVM)

Java API:

- The Java API is set of runtime libraries that give us a standard way to access the system resources of a host computer.

- When we run a Java program, the virtual machine loads the Java API class files that are referred to by the program's class files.
- The combination of all loaded class files (from your program and from the Java API) and any loaded dynamic libraries (containing native methods) constitute the full program executed by the Java virtual machine.

Fig. 1.6: Java API

- The class files of the Java API are inherently specific to the host platform. To access the native resources of the host, the Java API calls native methods.
- As we can see in Fig. 1.6, the class files of the Java API invoke native methods so your Java program doesn't have to.
- In this manner, the Java API's class files provide a Java program with a standard, platform-independent interface to the underlying host.
- Precisely because the Java virtual machine and Java API are implemented specifically for each particular host platform, Java programs themselves can be platform independent.
- The Java Standard Library (JSL or API) includes hundreds of classes and methods grouped into several functional packages.
- Most commonly used packages are given in following Table 1.2.

Table 1.2: Common Packages of JSL

Package Name	Name of Package	Description
java.lang	Language Support Package	A collection of classes and methods required for implementing basic features of Java.
java.util	Utility Package	A collection of classes to provide utility functions such as date and time.

contd. …

java.io	Input/Output Package	A collection of classes for input and output manipulation.
java.net	Networking Package	A collection of classes for communicating with other computers.
java.awt	AWT Package	An Abstract Window Toolkit which contains classes and methods for implementing Graphical User Interface.
java.applet	Applet Package	Set of classes to create Java Applets.

1.5 SIMPLE JAVA PROGRAM

- Program is a set of instructions that tell a computer what to do. A Java program is a collection or set of objects that communicate via invoking each other's methods.
- It is necessary to know the exact structure of the Java program. Java program structure means - the way (or general format) to write a Java program.
- Fig. 1.7 shows basic structure of a Java program.

Fig. 1.7: Structure of Java Program

- Fig. 1.7 shows following sections of a Java Program.
  1. **Documentation Section:** This section is not compiled by java compiler. This section is a set of comment lines giving the name of the program, the author and the other details, which the programmer would like to, refer.
  2. **Package Statement:** Package statement is used by programmer whenever they want to create a package. Package is collection of classes and interface. In Java files, the first statement allowed is a package statement which declares a package name and informs the compiler that the classes defined here belong to this package. Package statement an optional statement. Package statement is written as, `package PackageName;` **For example:** `package employee;`

3. **Import Statement:** Import statement is used to include java packages or user defined packages in to the program. For example, to include java input output package user can use the statement, `import java.io.*;`.
4. **Interface Statement:** An interface is like class used to group of related methods. Interfaces are used when we want to implement or implement multiple inheritance.
5. **Class Definition:** Classes are an essential part of any Java program. To write a Java program, we first define classes. A Java program may contain many/multiple class definitions. As Java is pure object oriented so each and every code will comes under a class, which consist of main() method. Every program in Java will have at least one class with the main() method.
6. **main Method Class:** This is essential part of the Java program which is a starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. In simple words, the execution of a java program starts from main() method.

- The steps in writing a Java program are illustrated in Fig. 1.8.

Fig. 1.8: Steps for create a Java Program

- Fig. 1.8 shows following steps for creation of a Java program.

Step 1 : Write the source codes (.java) using a programming text editor or an IDE (Integrated Development Environment) like Eclipse, NetBeans etc.

Step 2 : Compile the source codes (.java) into Java portable bytecodes (.class) using the JDK compiler ("javac"). IDE compiles the source codes automatically while they are entered.

Step 3 : Run the compiled bytecodes (.class) with the input to produce the desired output, using the Java Runtime ("java").

- To write Java programs, we will need a text editor. A text editor is a type of program used for editing plain text files. The text editors for Java includes Notepad++, jEdit, Editplus and so on.
- There are even more sophisticated IDEs (Integrated Development Environment) available in the market like Eclipse, Netbeans, JCreator and so on.

First Java Program: **[Oct. 17, April 18]**

```
public class HelloJavaWorld
{
   public static void main(String[] args)
   {
      System.out.print("Hello");
      System.out.println("Java world!");
   }
}
```

Output:

```
Hello
Java World!
```

Explanation of above Program:

- The **class** keyword is used to declare a class in Java. In the above program, 'HelloWorld' defines the class name in Java. Every class definition in Java begins with an opening brace { and ends with closing brace }.
- The **public** is a keyword and a access specifier indicates that the class member can be accessed from anywhere in the program.
- The **static** keyword indicates that this method can be invoked simply by using the name of the class without creating any class-object.
- The **void** keyword specifies that this method will not return any type of data. The main() method is the main entry point of the program, to start execution. First of all JVM calls the main method of a class and start execution. JVM (Java Virtual Machine) is responsible for running Java programs.
- **args** is a string array that takes values from Java command line. It's index starts from '0'. We can access values by writing args[0], args[1] etc.
- The **print** word sends the information into a buffer. This buffer is not flushed till a newline or end-of-line character is send. So it prints output on one line and cursor is blinking on the same line.
- The **println**() function prints the output to the standard output stream i.e. monitor.
- The **out** word represents the standard output stream (monitor).

Compiling the Program: **[Oct. 17]**

- To compile the HelloWorld program, execute the compiler, javac specifying the name of the source file on the command line. This is shown below:

```
$ javac HelloJavaWorld.java
```

- Javac compiler creates a file named 'HelloJavaWorld.class' which contains the byte code version of the program. It is an intermediate representation of the program.

Executing the Program:

- To actually run the program we must use the Java application launcher called 'java'. This is shown below:

```
$ java HelloJavaWorld
```

1.6 JAVA TOOLS

- Java platform includes a tool set for development of Java programs. These tools run in a command-line window and provide basic functionality.
- The Java Development Kit (JDK) comes with a set of tools that are used for developing and running Java program. JDK has a collection of tools for Java program development as shown in Table 1.3.

Table 1.3: Java Development Tools

Tool	Description
javac	The java compiler that translates javacode to bytecode file which java interpreter can understand.
java	java is a Java application launcher. It launches a java application by starting Java Runtime Environment, loading a specified class and invoking that class's main method to run the program.
appletviewer	Enables us to run java applets. Normally applets are the programs those can be embedded in a web page and run in web browser. This is very useful tool which gives the same effect but on console.
javadoc	Creates HTML formatted documentation from Java source code files.
jdb	Java debugger that helps us in finding errors in Java programs.
jar	Java achieve file, creates a zip/jar file.
javah	Produces header files for use with native methods.
javap	Java dissembler which enables us to convert bytecode files into a program description.

1.6.1 javac

- javac is the primary java compiler included in the JDK. javac is used to compile Java source into bytecode.
- The java source code is written in files with extension .java. In order to execute the program, these files need to be compiled in .class files. javac translates or converts the ".java" file to a ".class" file.

Syntax: `$ javac filename.java`

Example: `$ javac HelloJavaWorld.java`

- Fig. 1.9 illustrates the operation of the Java compiler.

Fig. 1.9: The operation of the Java compiler (javac)

1.6.2 java

- In Java language, java is launcher for Java program. java is the interpreter which launches a java program.
- The java interpreter executes the bytecode files produced by the Java compiler. It translates the Java bytecode into the code that can be understood by the Operating System (OS).

  Syntax: `$ java filename`
  Example: `$ java HelloJavaWorld`
- Fig. 1.10 shows java interpreter process.

Fig. 1.10: Java interpreter process

1.6.3 AppletViewer [April 16]

- An Applet is a Java program that runs in a Web browser. Java AppletViewer is the utility which runs Applets outside the web browser.
- AppletViewer is a standard JDK tool which has following facilities:
  1. To create dynamic applet, and
  2. To view java applet.
- A Java applet is a Java program designed to be included in a HyperText Markup Language (HTML) document and run inside a web browser.

- Fig. 1.11 shows steps in Applet.

Fig. 1.11: Applet

1.6.4 jdb [April 18]

- The Java debugger (jdb) is a tool for Java classes to debug a program in command line. jdb helps us to find and fix bugs in Java language programs.
- Debugging is a technical procedure to find and remove bugs or defects in a program and get expected results.
- A debugger allows us to step through every aspect of a code, inspect all the elements, and remove errors, if any.
- jdb helps in detecting and fixing bugs in a Java program using Java Debug Interface (JDI). Fig. 1.12 shows the role of jdb in JDK.
- jdb contains three units i.e., Java Virtual Machine Tool Interface (JVM TI), Java Debug Wiring Pool (JDWP) and Java Debugger Interface (JDI) as shown in Fig. 1.12.

Fig. 1.12: Architecture of jdb

1.6.5 javadoc [April 16, 17, 19]

- The javadoc tool is a document generator tool in Java for generating standard documentation in HTML format.
- javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.
- The javadoc is a documentation generator created by Sun Microsystems for the Java language generating API documentation in HTML format from Java source code.
- The documentation comments (also called doc comments) starts with a slash and two asterisks (/**) and it is terminated with one asterisk and a slash (*/). That is why they appear as:

```
/**
    ----------
    ----------
*/
```

- Tags are keywords recognized by javadoc which define the type of information that follows. We can use javadoc tags in the doc comments. The **syntax** of a tag is, `@ tag`.
  - All javadoc tags start with @ character.
  - Each tag must start on a new line.
  - Various tags are used to provide the heading.
  - All tags should follow the order.
- Some common pre-defined tags for javadoc are given below:
  1. **@author [author name]:** Identifies author(s) of a class or interface.
  2. **@version [version]:** Version info of a class or interface.
  3. **@param [argument name] [argument description]:** Describes an argument of method or constructor.
  4. **@return [description of return]:** Describes data returned by method (unnecessary for constructors and void methods).
  5. **@exception [exception thrown] [exception description]:** Describes exception thrown by method.

1.6.6 javap [Oct. 16, 18]

- The javap tool of Java is used to get the information of any class. In Java the javap command disassembles a class file.
- The javap command (also known as the Java Disassembler) disassembles one or more class files.
- The javap tool disassemblies compiled Java files and prints out a representation of the Java program. This may be helpful when the original source code is no longer available on a system.

- The javap command displays information about the fields, constructors and methods present in a class file.

  Syntax: `javap class_name`

  Example:

```
class Simple
{
    public static void main(String args[])
    {
        System.out.println("hello java");
    }
}
```

- Now let use the javap tool to disassemble the class file.

```
javap Simple
```

Output: Compiled from ".java"

```
class Simple
{
    Simple();
    public static void main(java.lang.String[]);
}
```

- It's output depends on the options used. The important options of javap tools are as follows:

Option	Description
-help	Print the help message.
-l	Prints line number and local variable.
-c	Disassembles the code.
-s	Prints internal type signature.
-sysinfo	Shows system info (path, size, date, MDS hash).
-constants	Shows static final constants.
-version	Shows version information.

1.6.7 jar [April 19]

- jar stands for Java archive. The jar tool is a java application that combines multiple files into a single JAR archive file.
- jar is a general-purpose archiving and compression tool based on ZIP and the ZLIB compression format.
- However, jar was designed mainly to facilitate the packaging of java applets or applications into a single archive.

  Syntax: `jar option destinationfile_list`

Where, destination file_list can have more than one file separated by space, ?, & and * are the wildcard characters used with filenames.

Options	Description
c	Creates new or empty ·jar file.
f	Created jar file named on command line.
t	Lists the table of contents.
v	Generates additional output.
x	Extract the file on command line.
0 (digit zero)	No compression and store the file.

Example:
```
jar cf first.jar  second.class
jar tf  newjar.jar
```

1.7 BASIC CONCEPTS IN JAVA

- In this section we will study basic concepts in Java like variables, data types, constants, comments, identifiers and so on.

1.7.1 Comments

- Comments are non-executable statements and are ignored by the Java compiler. The comment increases the readability of the programs.
- Comments help the person/user reading the code better understand the intent and functionality of the program.
- All characters available inside any comment are ignored by Java compiler. Java language provides three styles or types of comments as given in following table:

Sr. No.	Type of Comment	Description
1.	/* Multiline comment */	Multi line comments start with /* and ends with */. Any text between /* and */ will be ignored by Java compiler. **Example:** `/* Welcome to Nirali Prakashan. A text book publishing firm */`
2.	// Single line comment	Single-line comments start with two forward slashes (//). Any text between // and the end of the line is ignored by Java (will not be executed). **Example:** `// Welcome to Nirali Prakashan.`

contd. ...

3.	/** Documentation comment */	A doc comment is an ordinary multiline comment that begins with /** and ends with */. A doc comment appears immediately before a class, interface, method, or field definition and contains documentation for that class, interface, method, or field. The documentation can include simple HTML formatting tags and other special keywords that provide additional information.
		Example:
		`/** Welcome to Nirali Prakashan.`
		`A text book publishing firm */`

Program 1.1: Program for Java comment.

```
/** This is a Documentation Comment. */
public class CommentsDemoExample
{
   public static void main(String args[])
   {
      /* This is a Multi line Comment */
      System.out.println("Hello World"); // This is a Single Live Comment.
   }
}
```

Output:

```
Hello World
```

1.7.2 Data Types [Oct. 16, April 17]

- Data types specify the different values that can be stored in the variable. Variables are nothing but reserved memory locations to store values.
- The type of value that a variable can hold is called as data type. The main purpose of data types in Java is to determine what kind of value we can store into the variable.
- The data type of a variable determines the value which a variable can contain and the operations that can be performed on it. Every variable must have a data type.
- Java has a rich set of data types. The data types in Java are shown in Fig. 1.13.

Fig. 1.13: Data Types in Java

- Java supports two kinds of data types namely, primitive data types or predefined data types and non-primitive data types or reference data types.

1. Primitive Data Types: **[Oct. 16]**

- Primitive data types also known as built-in data types, are the fundamental data types provided by a programming language.
- In Java language primitive data types include integer, floating point, character and Boolean.
- The eight primitive data types supported by the java programming language are:
  - **(i)** **byte**: The byte data type is an 8-bit signed two's complement integer. It has a maximum value of −128 and a maximum value of 127 (inclusive).
  - **(ii)** **short**: The short data type is 16-bit singed two's complement integer. It has a maximum value of −32,768 and a maximum value of 32,767 (inclusive).
  - **(iii)** **int**: The int data type is 32-bit singed two's complement integer. It has a maximum value of −2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).
  - **(iv)** **long**: The long data type is 64-bit singed two's complement integer. It has a maximum value of −9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,808 (inclusive). Use this data type when you need a range of values wider than those provided by int.
  - **(v)** **float**: The float data type is a single-precision 32-byte IEEE 754 Floating point.
  - **(vi)** **double**: The float data type is a double-precision 64-byte IEEE 754 Floating point.
  - **(vii)** **boolean**: The Boolean data type has only two possible values namely, True and False. Use this data type for simple flags that track true/false conditions.
  - **(viii)** **char**: The char data type is single 16-bit Unicode character. It has a maximum value of '\u0000'(or 0) and a maximum value of '\uffff'(or 65,535 inclusive).

- Table 1.4 shows data types with their size in bytes.

Table 1.4: Data types with their size

Sr. No.	Data Type	Size in Bytes
1.	byte	1
2.	short	2
3.	int	4
4.	long	8
5.	float	4
6.	double	8
7.	char	2

2. Non-Primitive Data Types:

- Non-primitive data types also known as reference data types are derived from the primitive data types.
- In Java, derived data types include classes, interface, arrays etc.

1.7.3 Tokens

- The smallest individual units in a Java program are called tokens. Java tokens are smallest elements of a Java program which are identified by the compiler.
- Tokens in Java include variables, constants, identifiers, keywords, literals, operators, special characters, separators and so on.

Java Identifiers:

- Identifiers are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.
- In Java, there are several rules for identifiers. They are as follows:
  1. All identifiers should begin with a letter (A to Z or a to z), currency character ($) or an underscore (_).
  2. After the first character, identifiers can have any combination of characters.
  3. A keyword cannot be used as an identifier.
  4. Identifiers are case sensitive so Amar and amar has different meaning.
- Examples of legal identifiers: `age, $salary, _phone, _ID.`
- Examples of illegal identifiers: `123abc, -salary.`

Constants in Java:

- In java constants, refer to fixed values that do not change during the execution of a program.

- Constants are also called as literals which has fixed values in a program. Java supports several types of constants below:

1. **Integer Constants:**

- An integer constant means the sequence of digits. There are three types of integers:
  - **(i) Decimal Integer:** Decimal integers consists of a set of digits, 0 through 9 preceded by an optional minus sign.
    For example: `325, –978, 45673`.
    Embedded spaces, commas and non-digits characters are not allowed between digits.
    For example: `15 78, $657–, #47` is illegal number.
  - **(ii) Octal Integer:** An octal integer constant consists of any combination of digits from the set 0 through 7, with al leading 0.
    For example: `027, 0, 0231, 0341`.
  - **(iii) Hexadecimal Integer:** A sequence of digits preceded by 0X or 0X is considered a hexadecimal integer (hex integer).They may also include alphabet A through F or a through f. A letter A through F represents the number 1 through 15.
    For example: `0X4, 0X15, 0xabed`.

2. **Real Constant:**

- Integers numbers are insufficient to represents quantities that vary continuously, such as distance heights, temperature, prices, and so on.
- Numbers having fractional parts like 17.548 represents these quantities. Such numbers are called real (or floating point) constants.
  For example: `1.9783, 0 0047`
- The real number constant can comprise of four parts namely, whole number, decimal point, fractional part and exponent part.
- These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part, which is an integer.
  For example: `0.035, 111.98`.
- A real number may also be expressed as in exponential (or scientific) notation. For example, the value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10^2. .
  For example: `0.65e4, 15e–5, 2.6e+6 56.E4`.

3. **Character Constant**:

- A single character constant or character constant contains a single character enclosed within a pair of single quote marks.
- Examples are `‘7’, ‘A’, ‘*’, ‘ ’`.
- The character constant ‘7’ is not the same as the number 7. The last constant is blank space.

4. **String Constant**:

- A string constant is a sequence of character enclosed between double quotes in which the character may be alphabets, digits, special characters and blank spaces.
- **For example:** `“WELCOME”, “@june”, “abhishek”, “!NIDHI!”`

Backslash Characters:

- The backslash character constant are used with output statements for getting more effects like tab, new line, single quote etc. Java support some special backslash character constants.
- A backslah (\) precedes that character to represent it as an escape sequence. For example, the character n represents the alphabet n, where \n represents a new line character. Here, \n is an escape sequence that uses an available printable character n.
- A list of such backslash character constant is given following table. Each one of them represents one character, although they consist of two character. These characters combinations are known as escape sequences.

Backslash Character Constant	Meaning
\b	Back space
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\’	Single quote
\”	Double quote
\\	Backslash

Keywords:

- There are certain words that have a standard, predefined, special meaning in Java. They are called as keywords or reversed words and they can be used only for their intended purposes.
- These reserved words may not be used as constant or variable or any other identifier names. The keywords in Java are formed using lower case letters only.
- Keywords in Java are given below:

```
abstract        assert          boolean         break
byte            case            catch           char
class           const           continue        default
do              double          else            enum
extends         final           finally         float
for             goto            if              implements
import          instanceof      int             interface
long            native          new             package

private         protected       public          return
short           static          strictfp        super
switch          synchronized    this            throw
throws          transient       try             void
volatile        while
```

Separator:

- In java separators are symbols used to show the separated code, they describe function of our code.
- Table 1.5 shows various type of separators used in Java.

Table 1.5: Type of Separators

Name	Use
()	Parameter in method definition, containing statements for conditions, etc.
{ }	Used for define a code for method and classes.
[ ]	Used for declaration of array.
;	Used to show that separate statement.
,	Used to show the separation in identifier in variable declaration.
.	Used to show the separate package name from sub-packages and classes, separate variable and method from reference variable.

Operators in Java:

- Operator is a symbol that represents some operation that can be performed on data. Operators perform some specific operations using the operands and operator.
- For example, answer can be given using expression 4 + 5 is equal to 9. Here, 4 and 5 are called operands and + is called operator.
- The operator that uses a single operand is called a unary operator and that which uses two operands is known as a binary operator. A ternary operator uses three operands.
- Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the various groups Arithmetic Operators, Relational Operators, Bitwise Operators, Logical Operators, and Assignment Operators and so on.

1. Arithmetic Operators:

- Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.
- Table 1.6 lists the arithmetic operators. Assume integer variable A holds 10 and variable B holds 20.

Table 1.6: Arithmetic Operators

Sr. No.	Operator	Description	Example
1.	+ (Addition Operator)	Adds values on either side of the operator.	A + B will give 30
2.	− (Subtraction Operator)	Subtracts right hand operand from left hand operand.	A − B will give − 10

contd. …

3.	* (Multiplication Operator)	Multiplies values on either side of the operator.	A * B will give 200
4.	/ (Division Operator)	Divides left hand operand by right hand operand.	B / A will give 2
5.	% (Modulus Operator)	Divides left hand operand by right hand operand and returns remainder.	B % A will give 0
6.	++ (Increment Operator)	Increases the value of operand by 1.	B++ gives 21
7.	-- (Decrement Operator)	Decreases the value of operand by 1.	B-- gives 19

Program 1.2: Program for arithmetic operators.

```
public class Test
{
   public static void main(String args[])
   {
      int a = 10;
      int b = 20;
      int c = 25;
      int d = 25;
      System.out.println("a + b = " + (a + b) );
      System.out.println("a - b = " + (a - b) );
      System.out.println("a * b = " + (a * b) );
      System.out.println("b / a = " + (b / a) );
      System.out.println("b % a = " + (b % a) );
      System.out.println("c % a = " + (c % a) );
      System.out.println("a++   = " +  (a++) );
      System.out.println("b--   = " +  (a--) );
      // Check the difference in d++ and ++d
      System.out.println("d++   = " +  (d++) );
      System.out.println("++d   = " +  (++d) );
     }
}
```

Output:

```
a + b = 30
a - b = -10
a * b = 200
b / a = 2
b % a = 0
c % a = 5
a++   = 10
b--   = 11
d++   = 25
++d   = 27
```

2. Relational Operators:

- Relational operators are also called as comparison operators used for comparing values of two variables.
- Table 1.7 shows relational operators supported by Java language. Assume variable A holds 10 and variable B holds 20.

Table 1.7: Relational Operators

Sr. No.	Operator	Description	Example
1.	= = (Equal to Operator)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A = = B) is not true.
2.	!= (Not Equal to Operator)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
3.	> (Greater than Operator)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
4.	< (Less than Operator)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
5.	>= (Greater than or Equal to Operator)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
6.	<= (Less than or Equal to Operator)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Program 1.3: Program for relational operators.

```
public class Sample
{
   public static void main(String args[])
    {
      int a = 10;
      int b = 20;
      int c = 25;
      int d = 25;
      System.out.println("a < b = " + (a < b) );
      System.out.println("c > b = " + (c > b) );
```

```
        System.out.println("c <=  d = " + (c <= d) );
        System.out.println("c >=  d = " + (c >= d) );
        System.out.println("c ==  d = " + (c == d) );
    }
}
```

Output:

```
a < b = true
c > b = true
c <=  d = true
c >=  d = true
c ==  d = true
```

3. Logical Operators:

- When we want to form compound conditions by combining two or more relations, then we can use logical operators.
- Logical operators are used to check whether an expression is True or False. Table 1.8 lists the logical operators. Assume Boolean variables A holds True and variable B holds False.

Table 1.8: Logical Operators

Sr. No.	Operator	Description	Example
1.	&& (Logical AND Operator)	If both the operands are non-zero, then the condition becomes true.	(A && B) is False.
2.	|| (Logical OR Operator)	If any of the two operands are non-zero, then the condition becomes true.	(A || B) is True.
3.	! (Logical NOT Operator)	Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	! (A && B) is True.

Program 1.4: Program for logical operators.

```
public class Sample
{
    public static void main(String args[])
    {
        int a = 60;
        int b = 20;
        int c = 25;
        int d = 25;
        System.out.println("(a > b && c == d) = " + (a > b && c == d) );
        System.out.println("(a < b && c == d) = " + (a < b && c == d));
        System.out.println("(a < b || c == d) " + (a < b || c == d) );
        System.out.println(" !(a < b) = " + !(a < b ) );
    }
}
```

Output:

```
(a > b && c == d) = true
(a < b && c == d) = false
(a < b || c == d) true
!(a < b) = true
```

4. Assignments Operators:

- Assignment operators are used to assign values to variables.

Table 1.9: Assignments Operators

Sr. No.	Operator	Description	Example
1.	= (Simple Assignment Operator)	Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
2.	+= (Add and Assignment Operator)	It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
3.	−= (Subtract and Assignment Operator)	It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C − A
4.	*= (Multiply and Assignment Operator)	It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
5.	/= (Divide and Assignment Operator)	It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
6.	% = (Modulus and Assignment Operator)	It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
7.	<<=	Left Shift AND Assignment Operator.	C <<= 2 is same as C = C << 2
8.	>>=	Right shift AND Assignment Operator.	C >>= 2 is same as C = C >> 2
9.	&=	Bitwise AND Assignment Operator.	C &= 2 is same as C = C & 2
10.	^=	Bitwise Exclusive OR AND Assignment Operator.	C ^= 2 is same as C = C ^ 2
11.	|=	Bitwise Inclusive OR AND Assignment Operator.	C |= 2 is same as C = C | 2

5. Increment and Decrement operator:

- Java provides two increment and decrement operators which are unary increment (++) and decrement (−−) operators.
- Increment and decrement operators are used to increase or decrease the value of an operand by one, the operand must be a variable, an element of an array, or a field of an object.
- Suppose, a = 5 then,

```
++a;          //a becomes 6
a++;          //a becomes 7
--a;          //a becomes 6
a--;          //a becomes 5
```

Program 1.5: Program for increment and decrement operators.

```
class Operator
{
   public static void main(String[] args)
   {
      int var=5;
      System.out.println("var++:"+ var++);
      System.out.println("\n++var:"+ ++var);
   }
}
```

Output:

```
var++:5
++var:7
```

6. Conditional Operator (?:):

- Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions.
- The goal of the ternary operator is to decide which value should be assigned to the variable.
- The **syntax** for ternary operator is,

```
variable x = (expression) ? value if true: value if false
```

Program 1.6: Program for ternary operator.

```
public class Test
{
   public static void main(String args[])
   {
      int a , b;
      a=10;
      b=(a ==1)?20:30;
```

```
        System.out.println("Value of b is: "+  b );
        b=(a ==10)?20:30;
        System.out.println("Value of b is: "+ b );
    }
}
```

Output:

```
Value of b is:30
Value of b is:20
```

7. Bitwise Operators:

- Bitwise operators in Java are used to perform operations on individual bits.
- Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte. Bitwise operator works on bits and performs bit-by-bit operation.
- Assume if a = 60; and b = 13; now in binary format they will be as follows:

  a = 0011 1100
  b = 0000 1101
  ----
  a&b = 0000 1100
  a|b = 0011 1101
  a^b = 0011 0001
  ~a = 1100 0011

- Table 1.10 lists the bitwise operators: Assume integer variable A holds 60 and variable B holds 13.

Table 1.10: Bitwise Operators

Sr. No.	Operator	Description	Example
1.	& (Binary AND Operator)	Copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
2.	| (Binary OR Operator)	Copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
3.	^ (Binary XOR Operator)	Copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
4.	~ (Binary Ones (1's) Complement Operator)	Unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.

contd. …

5.	<< (Binary Left Shift Operator)	The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
6.	>> (Binary Right Shift Operator)	The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
7.	>>> (Shift right zero fill operator)	The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Program 1.7: Program for bitwise operators.

```
public class Sample
{
   public static void main(String args[])
   {
      int a = 60;
      int b = 20;
      System.out.println("a & b = " + (a & b) );
      System.out.println("a | b = " + (a | b) );
      System.out.println("a <<  2 = " + (a << 2) );
      System.out.println("a >>  2 = " + (a >> 2) );
      System.out.println("a >>> 2 = " + (a >>> 2) );
  }
}
```

Output:

```
a & b = 20
a | b = 60
a <<  2 = 240
a >>  2 = 15
a >>> 2 = 15
```

8. Special Operators:

- There are few other operators supported by Java language as explained below:

(i) instanceof Operator:

- This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type).

- The instanceof operator is written as,

  `(Object reference variable) instanceof (class/interface type)`
- If the object on the left side of the operator is an object of right side, then the result will be true.

Program 1.8: Program for instanceof operator.

```
public class Test
{
   public static void main(String args[])
   {
      String name ="James";
      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

Output:

```
true
```

(ii) Dot Operator:

- The dot operator (.) is used to access class members i.e. methods and data.

Program 1.9: Program for dot operator.

```
class Person
{
   String name;
   int age;
   void display()
   {
      System.out.println("Name="+name);
      System.out.println("Age="+age);
   }
   public static void main(String args[])
   {
      Person p=new Person()
      p.name="abc";
      p.age="10";
      p.display();
   }
}
```

Output:

```
Name=abc
Age=10
```

Operator Precedence in Java:

- Java has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators.
- For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence Order:

- When two operators share an operand the operator with the higher precedence goes first.
- For example, 1 + 2 * 3 is treated as 1 + (2 * 3), whereas 1 * 2 + 3 is treated as (1 * 2) + 3 since multiplication has a higher precedence than addition.

Operator Associativity in Java:

- When two operators with the same precedence the expression is evaluated according to its associativity.
- For example, x = y = z = 17 is treated as x = (y = (z = 17)), leaving all three variables with the value 17, since the = operator has right-to-left associativity (and an assignment statement evaluates to the value on the right hand side).
- On the other hand, 72 / 2 / 3 is treated as (72 / 2) / 3 since the / operator has left-to-right associativity.
- Table 1.11 shows the associativity and precedence order of java operators.

Table 1.11: Operator Precedence

Operators	Associativity	Precedence Rank
(), []	Left to right	1
−, ++, --, !	Right to left	2
*, /, %	Left to right	3
+, −	Left to right	4
<< , >>	Left to right	5
<, <=, >, >=	Left to right	6
==, !=	Left to right	7
&	Left to right	8
^	Left to right	9
|	Left to right	10
&&	Left to right	11
||	Left to right	12
?:	Right to left	13

1.7.4 Variables

- Variables are containers for storing data values. Variables are the basic units used to store any information in Java.
- A variable is an identifier that denotes a storage location used to store a data value. The value in the variable can be changed during the program's execution.
- Variables are nothing but reserved memory locations to store values. This means that when we create a variable we reserve some space in memory.
- Following are the some rules used for variable naming:
  1. Variable name must begin with either a letter or the dollar sign "$" or the underscore character "_".
  2. Variable names are case-sensitive. Upper case and lower case must be distinct this means that the variable Abc is not the same as abc or ABC.
  3. Variable must not begin with a digit.
  4. White space is not allowed in variable naming.
  5. Variable name should not be a keyword.
  6. Variable names can be of any length.

Variable Declaration:

- In Java, all the variables must be declared before use. A variable declaration, in its simplest form, includes the name and the data type of variables.
- Declaration does following three things:
  1. It tells variable name to the compiler.
  2. It specifies the data type of the data hold by variable.
  3. The place of declaration in the program decided the scope of the variable.
- Following is the basic **form/syntax** of a variable declaration:

```
data_type variable_Name;
```

  Here, data_type is one of Java's data types and variable_Name is the name of the variable.
- To declare more than one variable of the specified type, you can use a comma-separated list as given below:

```
data_type variable_Name1, variable_Name2, …… variable_NameN;
```

Example:

```
int age;      // Declares age
int a, b, c;  // Declares three ints, a, b, and c.
```

Variable Initialization:

- Variable initialization means assigning a value to variables. Initializing a variable means specifying an initial value to assign to it (i.e., before it is used at all).
- In Java, we can assign a value to variables in two ways:
  1. **Static:** This means that the memory is determined for variables when the program starts. Variables can be assigned initial values at the time of declaration. Assignment operator (=) assigns the value of an expression to a variable.
     Syntax: `variable_name=value;` **OR** `data_type variable_name=value;`

Examples:
```
IntitalValue = 0;
a = b = c = 0;
int a = 10, b = 10;  // Variable initialization
byte B = 22;         // initializes a byte type variable B.
double pi = 3.14159; // declares and assigns a value of PI.
```

2. **Dynamic:** Dynamic means that in Java, we can declare variables anywhere in the program, because when the statement is executed the memory is assigned to them. Java allows its programmers to initialize a variable at run time also. Initializing a variable at run time is called dynamic initialization.

Program 1.10: Program for dynamic initialization of variables.

```
class DynamicInit
{
   public static void main(String args[])
   {
      double a = 3.0, b = 4.0;
      // c is dynamically initialized
      double c = Math.sqrt(a * a + b * b);
      System.out.println("C=" + c);
   }
}
```

Output:

```
C=5.0
```

Scope of Variables:

- The area of the program where the variable is accessible is called its scope. Scope refers to the lifetime and accessibility of a variable.
- The scope of a variable defines the section of the code in which the variable is accessible or visible.
- Scope is the lifetime of a variable and refers to how long the variable exists before it is destroyed.
- Scope itself specifies that the system allocates and de-allocates the memory for the variable.
- Java variables are actually classified into three categories and their scope is defined according their category as given below:
  1. **Instance Variables:** Instances are declared inside the class. Instance variables are created when the objects are instantiated (created) and those are associated with the objects. They take different values for each object. Instance variables are declared in a class, but outside a method, constructor or any block. The instance variables are visible for all methods, constructors and block in the class.
  2. **Class/Static Variables:** Class variables are declared inside the class and global to the class and belong to the entire set of object and that class creates. Only one memory location is created for each class variable. Visibility is similar to instance variables.

3. **Local variables:** Local variables declared and used inside the methods. They are not available outside of the method definition. Local variables can be declared inside the program blocks that are defined between the opening { and closing braces }. These variables are visible to the program only. When the program control leaves block, all the variables in the block will cease to exit.

Sr. No.	Characteristic	Local Variable	Instance Variable	Class Variable
1.	Where declared	In a method, constructor, or block.	In a class, but outside a method.	In a class, but outside a method. Must be declared static.
2.	Use	Local variables hold values used in computations in a method.	Instance variables hold values that must be referenced by more than one method.	Class variables are mostly used for constants, variables that never change from their initial value.
3.	Lifetime	Created when method or constructor is entered. Destroyed on exit.	Created when instance of class is created with new. Destroyed when there are no more references to enclosing object (made available for garbage collection).	Created when the program starts. Destroyed when the program stops.
4.	Scope/Visibility	Local variables (including formal parameters) are visible only in the method, constructor, or block in which they are declared.	Instance (field) variables can been seen by all methods in the class.	Same as instance variable, but are often declared public to make constants available to users of the class.

Program 1.11: Program for scope of variables.

```
public class Sample
{
   string name;                    //instance variable
   int age;
   static int count;               //class variable
   void accept(string n,int a)     // n, a are local variables
   {
      name=n;
      age=a;
   }
   void show()
   {
      count++;
      System.out.println("count="+count);
   }
   void display()
   {
      System.out.println("Name="+name);
      System.out.println("Age="+age);
   }
   public static void main(String args[])
   {
      Sample s1=new Sample();
      Sample s2=new Sample();
      s1.accept("abc",10);
      s1.display();
      s1.show();
      s2.accept("xyz",20);
      s2.display();
      s2.show();
   }
   }
```

Output:

```
Name=abc
Age=10
count=1
Name=xyz
Age=20
count=2
```

1.7.5 Final Variable [April 16, 17, Oct. 18]

- Variables are useful when we need to store information that can be changed as program runs.
- However, there may be certain situations in the program in which the value of variable should not be allowed to modify or change. This is accomplished using a special type of variable known as final variable.
- The final variable also called constant variable. Final variable is a variable with a value that cannot be modified during the execution of the program.
- To declare a final variable, use the final keyword before the variable declaration and initialize it with a value. Its syntax is,

```
final data_type varName = value;
```

For example:

```
final double PI = 3.141592;
```

- Above statement declares a variable PI of type double which is initialized to 3.141592, that cannot be modified. If an attempt is made to modify a final variable after it is initialized, the compiler issues an error message "cannot assign a value to final variable PI".

Program 1.12: Program to find area of circle using final variables.

```
public class area
{
    public static void main(String args[])
    {
        int r=10;           //initializing radius
        final double PI=3.141592; // final variable
        double area;
        area = PI * r *r;
        System.out.println("Area of Circle is: " +area);
    }
}
```

Output:

```
314.1592
```

1.7.6 Expressions and Statements

- A computer program is a set of instructions to be executed by computer and these programs instructions are called as statements.
- Java statements are separated by semicolon (;). An expression is a combination of operators and operands which reduces to a single value.
- Any valid grouping of constants, variables and operators constitutes an expression. For example a + b is an expression and x is also an expression.

- Different statements are supported by Java as shown in Table 1.12.

Table 1.12: Statement in java

Sr. No.	Statement	Description
1.	Empty Statement	These statements do nothing and are used during program development as a place holder.
2.	Labelled Statement	Any statement may begin with a label. Such labels may noy be keywords, already declared as local variables or previously used labels in this module. Labels in java are used as arguments of jump statements.
3.	Expression Statement	Most statements are expression statements. Java has seven types of expression statements namely, assignment, pre-increment, pre-decrement, post-increment, post-decrement, method call and allocation expression.
4.	Selection Statement	These statements select one of several control flows. There are three types of selection statements in Java namely if, if-else and switch.
5.	Iteration Statement	These statements specify how and when looping will take place. There are three types of iteration statements namely, while, do while and for.
6.	Jump Statement	These statements pass control to the beginning or end of the current block, or at a labelled statement such labels must be in the same block, and continue labels must be in the iteration statement. The four types of jump statements are break, continue, return and throw.
7.	Synchronization Statement	These statements are used for handling issues with multi-threading.
8.	Guarding Statement	These statements are used for safe handling of code that may cause exceptions (such as division by zero) these statements use keywords try, catch and finally.

1.7.7 Control Flow Statements

- Statements that control the flow of the program are called control flow statements.
- The control statements are used to control the flow of execution of the program. This execution order depends on the supplied data values and the conditional logic.
- There are three basic flow control constructs namely, sequential, conditional (or decision), and loop (or iteration) as shown in Fig. 1.14.

Fig. 1.14: Basic Flow Control Constructs

- Statements in a program are generally executed in a sequential manner, which is called sequential execution or sequential control flow.
- Conditional selection is a situation where we may have different types of data values, and may need to perform different processing steps.
- Iterative control is a situation where we need to perform repetitive process steps on similar data.
- Fig. 1.15 shows various Java control statements.

Fig. 1.15: Java control statements

1.7.8 Selection / Decision Making Statements

- A selection statement is a control statement that allows choosing between two or more execution paths in a program.
- The decision making statements are used to change the flow of the program based on conditions.
- Various selection statements are discussed below:

1. if Statement:

- The if statement is a powerful and simplest decision making statement and is useful to control the flow of execution of program statement.
- In 'if' statement if the condition is true then, it executes statement or statements which are present under if statement. Otherwise (if the condition is false) it goes out of loop or control transfer out of loop.

- Fig. 1.16 shows syntax and flow diagram of if statement.

```
if(condition)
{
    // Statements;
}
```

(a) Syntax **(b) Flow Diagram**

Fig. 1.16: if Statement

Program 1.13: Program for if statement.

```
public class ExampleIf
{
   public static void main(String[] args)
   {
      int number = 25;  //defining an variable
      if(number >0)
      {
         System.out.print("Number is greater than zero (0) ");
      }
   }
}
```

Output:

```
Number is greater than zero (0)
```

2. if else Statement:

- It is basically extension of if statement. If the Boolean-expression is True if code gets executed. If Boolean expression is False else code gets executed.
- In this decision making statements generally either of the block gets executed whether it's if code or else code, after either execution rest code gets executed.
- Fig. 1.17 shows syntax and flow diagram of if-else statement.

```
if(Boolean expression)
  {
     //if   code   goes   here…
     Executes when the Boolean
     expression is True…;
  }
  else
  {
     //else   code   goes   here
     Executes when the Boolean
     expression is False…;
  }
   rest-of-code;
```

(a) Syntax **(b) Flow Diagram**

Fig. 1.17: if else Statement

Program 1.14.: Program to find even or odd number using if else statement.

```
public class IfElseExample
{
   public static void main(String[] args) {
    //defining a variable
    int number=13;      //defining a variable
    if(number%2==0)    //Check if the number is divisible by 2 or not
   {
       System.out.println("Even number");
    }
   else
   {
       System.out.println("Odd number");
    }
}
```

Output:

```
Odd number
```

Nested 'if' Statement:

- The nested if statement represents the if statement within another if statement. Here, the inner if statement condition executes only when outer if statement condition is True.

Syntax:

```
if(Boolean expression 1)
{
   // Executes when the Boolean expression 1 is true
   if(Boolean expression 2)
   {
      // Executes when the Boolean expression 2 is true
   }
}
```

Program 1.15: Program for nested if statement.

```
public class JavaNestedIfExample
{
   public static void main(String[] args)
   {
       int age=20;  //Creating two variables for age and weight
       int weight=80;
       //applying condition on age and weight
```

```
        if(age>=18)
        {
           if(weight>50)
           {
             System.out.println("Eligible to donate blood");
           }
        }
     }
  }
```

Output:

```
Eligible to donate blood
```

Nested if else Statement:

- The nested if-else statements are used where we want to make series of decision making. Thus one if-else is nested within another if-else. Therefore nested if-else give us to make a decision within a decision already taken.
- The **syntax** for nested if-else statement given below:

```
if(Boolean expression 1)
{
   if(Boolean expression 2)
   {
      // Statement 1;
   }
   else
   {
      // Statement 2;
   }
   }
   else
   {
      // Statement 3;
   }
```

Program 1.16: Program for nested if else statement.

```
public class JavaNestedIfElseExample
{
   public static void main(String[] args)
   {
      int age=25;  //Creating two variables for age and weight
      int weight=48;
```

```
        //applying condition on age and weight
        if(age>=18)
        {
            if(weight>50)
            {
                System.out.println("Eligible to donate blood");
            }
            else
            {
                System.out.println("Not eligible to donate blood");
            }
        }
        else
        {
            System.out.println("Age must be greater than 18");
        }
    }
}
```

Output:

```
Not eligible to donate blood
```

3. else if Statement:

- Use the else if statement to specify a new condition if the first condition is false.

Syntax:

```
if(condition1)
{
    //block of code to be executed if condition1 is true
}
else if (condition2)
{
    //block of code to be executed if the condition1 is false and
    condition2 is true
}
else
{
    //block of code to be executed if the condition1 is false and
    condition2 is false
}
```

Program 1.17: Program for else if statement.

```
public class Main
{
   public static void main(String[] args)
   {
   int time = 22;
   if(time < 10)
   {
      System.out.println("Good morning.");
   }
   else if(time < 20)
   {
      System.out.println("Good day.");
   }
   else
   {
      System.out.println("Good evening.");
    }
  }
}
```

Output:

```
Good evening
```

4. switch Statement:

- Sometimes, it becomes cumbersome to write lengthy programs using if and if-else statements. To avoid this we can use switch statements in Java.
- The switch statement is the multi-way selection statement. It tests the value of the variable (or expression) against the list of case values and when match is found, a block of statements associated with that case are executed.
- Fig. 1.18 shows syntax and flow diagram of switch statement.

```
switch(variable)
{
   case value-1:
   statements-1;
   break;
   case value-2:
   statements-2;
   break;
   ---------
   ---------
   default:
   default block;
}
Rest-of-code;
```

(a) Syntax **(b) Flow Diagram**

Fig. 1.18: switch Statement

Program 1.18: Program for switch statement.

```
public class SwitchMonthExample
{
   public static void main(String[] args)
   {
      //Specifying month number
      int month=7;
      String monthString="";
      //Switch statement
      switch(month){
      //case statements within the switch block
      case 1: monthString="1 - January";
      break;
      case 2: monthString="2 - February";
      break;
      case 3: monthString="3 - March";
      break;
      case 4: monthString="4 - April";
      break;
      case 5: monthString="5 - May";
      break;
      case 6: monthString="6 - June";
      break;
      case 7: monthString="7 - July";
      break;
      case 8: monthString="8 - August";
      break;
      case 9: monthString="9 - September";
      break;
      case 10: monthString="10 - October";
      break;
      case 11: monthString="11 - November";
      break;
      case 12: monthString="12 - December";
      break;
      default:System.out.println("Invalid Month!");
      }
      //Printing month of the given number
      System.out.println(monthString);
   }
}
```

Output:

```
7 - July
```

- We can use switch statement inside other switch statement in Java. It is known as nested switch statement.

Program 1.19: Program for nested switch statement.

```
public class NestedSwitchExample
{
   public static void main(String args[])
   {
      //C - CSE, E - ECE, M - Mechanical
      char branch = 'C';
      int collegeYear = 4;
      switch(collegeYear)
      {
         case 1:
            System.out.println("English, Maths, Science");
            break;
         case 2:
            switch( branch )
            {
               case 'C':
               System.out.println("Operating System, Java, Data Structure");
               break;
               case 'E':
               System.out.println
                              ("Micro processors, Logic switching theory");
               break;
               case 'M':
               System.out.println("Drawing, Manufacturing Machines");
               break;
            }
            break;
         case 3:
         switch(branch)
         {
            case 'C':
            System.out.println("Computer Organization, MultiMedia");
            break;
            case 'E':
            System.out.println
                        ("Fundamentals of Logic Design, Microelectronics");
            break;
```

```
                case 'M':
                System.out.println
                        ("Internal Combustion Engines, Mechanical Vibration");
                break;
            }
            break;
            case 4:
            switch(branch)
            {
                case 'C':
                System.out.println
                              ("Data Communication and Networks, MultiMedia");
                break;
                case 'E':
                System.out.println("Embedded System, Image Processing");
                break;
                case 'M':
                   System.out.println
                               ("Production Technology, Thermal Engineering");
                break;
            }
            break;
        }
    }
}
```

Output:

```
Data Communication and Networks, MultiMedia
```

1.7.9 Loops

- There may be a situation when we need to execute a block of code several number of times.
- A loop statement allows us to execute a statement or group of statements multiple times until conditions become true.
- The process of repeatedly performing tasks is known as looping. There is either entry controlled loop or exit controlled loop.
- If a loop first checks condition for execution then it is called as entry controlled loop and if a loop checks condition after the execution of statement then they are called as exit controlled loops.

- The for and while loops are known as entry controlled loop. The do-while loop is exit controlled loop.
- Java is having three different loop statements, i.e., while loop, do-while loop and for loop.

1. while Loop:

- In Java while loop is used when as single or a group of statements is to be executed repeatedly as long as a given condition is satisfied.
- The while loop is entry controlled loop statement. The while statement continually executes a block of statements while a particular condition is true.
- Fig. 1.19 shows syntax and control flow of while loop.

```
while(condition)
    {
        //body of the loop;
    }
```

(a) Syntax

(b) Flow Diagram

Fig. 1.19: while Loop

Program 1.20: Program for while loop.

```
public class Examplewhile
{
   public static void main(String[] args)
   {
      int i =1;       // initalization of variable
      while(i<=10)    // check the condition
       {
          System.out.println(i);
         i++;     //increment variable by 1
       }
     }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

2. do-while Loop:

- Sometimes, in the program it is desirable to execute the loop at least once. The do-while loop can be applied in this case.
- The loop is executed first and then the loop condition is checked. The condition is written at the end of the loop.
- This is also called as exit-controlled loop.

```
do
{
   //body of the loop;
}while(condition);
```

(a) Syntax

(b) Flow Diagram

Fig. 1.20: do-while Loop

- In Fig. 1.20 the condition is evaluated after the end of each pass and the body of the loop will always execute at least once, even if the condition is false the first time.

Program 1.21: Program for do while loop.

```
public class DoWhileDemo
{
   public static void main(String[] args)
   {
   int count = 1;
      do
      {
         System.out.println("Count is: " + count);
         count++;
      } while (count < 11);
   }
}
```

Output:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

3. for Loop:

- The 'for' loop is entry-controlled loop. The for loop provide a more concise loop control structure.
- That is, initialization, condition and increment/decrement can be done in the single loop statement as shown in Fig. 1.21 (b).

```
for(initialization; condition;
increment/decrement)
{
   //Body of the loop;
}
```

(a) Syntax

(b) Flow Diagram

Fig. 1.21: for Loop

Program 1.22: Program for 'for' loop.

```
public class ForDemo
{
   public static void main(String[] args)
   {
      for(int i=1; i<=10; i++)
      {
         System.out.println("Count is: " + i +"  ");
      }
   }
}
```

Output:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

5. for each Loop:

- The basic for loop was extended in Java 5 to make iteration over arrays and other. This newer for statement is called the 'enhanced' for or 'for each'.
- The for each and equivalent for statements have following forms. The two basic equivalent forms are given, depending one whether it is an array or an Iterable that is being traversed.
- In both cases an extra variable is required, an index for the array and an iterator for the collection.

  Syntax:

  ```
  for(typevar: arr)
  {
     // Body-of-loop…
  }
  ```

  Here, var represent the element of array. During first iteration var will contain first element of array, during second iteration it will hold second element of array and so on.
- The loop will be executed for all elements of array. If array elements are finished loop will terminate automatically.

Program 1.23: Program of 'for each' loop.

```
public class Person
{
   public static void main(String [] args)
   {
      double[] ar = {1.2, 3.0, 0.8};
      int sum = 0;
      for (double d: ar)
      {
         System.out.println("Element:"+ d);
      }
   }
}
```

Output:

```
Element: 1.2
Element: 3.0
Element: 0.8
```

Nested Loops:

- The process of placing of one loop inside the body of another loop is called nesting. Like all other programming languages, Java allows loops to be nested. That is, one loop may be inside another.
- A nested loop is a loop that lies inside another loop. When we "nest" two loops, the outer loop takes control of the number of complete repetitions of the inner loop.

1. Nested for Loop:

- We can use for loop inside another for loop is called nested for loop.

Syntax:

```
for(initialization; test condition; increment/decrement)
{
   //Statements;
   for(initialization; test condition; increment/decrement)
   {
      //Statements;
      :
      :
      :
      for(initialization; test condition; increment/decrement)
      {
         //Statements;
      }
   }
}
```

Program 1.24: Program to display triangle of * using nested for loop.

```
public class NestedForLoopDemo
{
   public static void main(String args[])
   {
   for(int i = 1; i <=3 ; i++)
   {
      for(int j = 1; j <= i; j++)
      {
         System.out.print("* ");
      }
      System.out.println("");
      }
   }
}
```

Output:

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

Program 1.25: Program for nested for loop.

```
public class ForLoop
{
   public static void main(String[] args)
   {
      for(int i = 1;i <= 3;i++)
      {
         for(int j = 1;j <= i;j++)
         {
            System.out.print(i);
         }
            System.out.println();
      }
   }
}
```

Output:

```
1
2  2
3  3  3
```

2. Nested while Loop:

- One while loop can be nested with other while loop. We can use while loop inside another while loop is called nested while loop.

Syntax:

```
while(expression)
{
    //Statements;
    :
    :
    :
    while(expression)
    {
        //Statements;
    }
}
```

Program 1.26: Program to print tables.

```
public class NestedWhileLoop
{
    public static void main(String args[])
    {
        int i=1, j=1;
        System.out.println("Tables");
        while(i <= 2) // change to 2 to 10 or 20 as many tables user want
        {
            while(j <= 10)
            {
              System.out.println(i + " * " + j + " = " + (i*j));
              j++;
            }
            i++;
              System.out.println("");
              System.out.println("");
            }
        }
    }
```

Output:

```
Tables
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
```

3. Nested do while Loop:

- We can use do while loop inside another do while loop is called nested do while loop.
- The syntax for a nested do...while loop as follows:

```
do
{
   //Statement(s);
   do
   {
      //Statement(s);
   }while(condition);
}while(condition);
```

Program 1.27: Program to print triangle of numbers.

```
public class NestedDOWhileLoop
{
   public static void main(String args[])
   {
      int i=1, j=1;
   do
   {
   int k = 5;
      do
      {
         System.out.print(" ");
         k--;
         }while(k>=i);
            j = 1;
```

```
                do
                {
                   System.out.print(i + " ");
                   j++;
                }while(j<=i);
                   System.out.println("");
                i++;
                }while(i<=5);
            }
        }
```

Output:

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

1.7.10 Loop Control Statements

- Loop control statements (or jumping/branching statements) change execution of program from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.
- Using jumping statements like break and continue it is easier to jump out of loops to control other areas of program flow.
- Java supports three jump statements i.e., break, continue, and return.

1. **break Statement:**

- When the break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
- The break statement can be also used to terminate a case in the switch statement. The break statement has two forms namely labeled and unlabeled.

  Syntax: `break` and `break label;`

Program 1.28: Program to check a number is prime or not.

```
public class Sample
{
   public static void main(String args[])
   {
   int num=7;
   int flag=0;
      for(int i=2; i<num-1; i++)
      {
         if(num% i==0)
```

```
            {
               flag=1;
               break;
            }
            else
            {
               flag=0;
            }
         }
         if(flag==0)
         System.out.println(num +"is prime");
         else
         System.out.println(num +" is not prime");
      }
   }
```

Output:

```
7 is prime
```

Program 1.29: Program to break statement with label.

```
Public class BreakWithLabelDemo
{
   public static void main(String[] args)
   {
      int a[][]  = {{ 32, 87, 3, 589 }, { 12, 1076, 2000, 8 },
                                                 { 622, 127, 77, 955 }};
      int num = 12;
      int i;
      int j = 0;
      boolean flag = false;
      search: for (i = 0; i <a.length; i++)
      {
         for (j = 0; j < a[i].length; j++)
         {
            if (a[i][j] == num)
            {
            flag = true;
            break search;
         }
      }
   }
   if (flag)
   System.out.println("Found " + num + " at " + i + ", " + j);
   else
   System.out.println(num + " not in the array");
   }
}
```

Output:

```
Found 12 at 1, 0
```

2. continue Statement:

- The continue statement causes the loop to be continued with the next iteration after skipping any statements in between.
- The continue statement is used when it is necessary to skip a part of the body of the loop under certain conditions.
- The continue statement tells the compiler, "skip or omit the statements mentioned and continue with the next iteration."
- In while and do loops, using the continue statement, the control to go directly to the test condition and then continue the iteration process.
- In the case of for loop, the increment section of the loop is executed before the test condition is evaluated.
- The continue statement skips the current iteration of a for, while or do-while loop.
- The unlabeled form skips to the end of the innermost loop's body and evaluatesthe Boolean expression that controls the loop.

Syntax: `continue;`

Program 1.30: Program for continue statement.

```
public class Sample
{
   public static void main(String args[])
    {
      for(int i=1;i<=10;i++)
      {
         if(i==4)
         continue;
         System.out.println("i=" + i);
      }
   }
}
```

Output:

```
i=1
i=2
i=3
i=5
i=6
i=7
i=8
i=9
i=10
```

3. return Statement:

- The return statement exits from the current method and control flow returns to where the method was invoked.
- A return statement causes the program control to transfer back to the caller of a method.

  Syntax: `return;`

Program 1.31: Program for return statement.

```
public class ReturnTest
{
   public int add()
   { // without arguments
      int x = 30;
      int y = 70;
      int z = x+y;
      return z;
   }
   public static void main(String args[])
   {
      ReturnTypeTest1 test = new ReturnTypeTest1();
      int add = test.add();
      System.out.println("The sum of x and y is: " + add);
   }
}
```

Output:

```
The sum of x and y is: 100
```

1.8 ARRAYS IN JAVA [Oct. 16, April 18, 19]

- An array is a group of contiguous memory locations to store the data of common type. An array is a collection of similar type of elements that have contiguous memory location.
- Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.
- Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type.
- An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Commonly used terms or element in array are explained below:
  1. **Element:** Element of an array is the items it contains. An array can contain one or more elements.

2. **Value:** Each element contains one value.
3. **Key or index:** Key or index of an array is a unique number or a string that is associated with each value of an element.
4. **Length:** The length of an array is the number of elements it contains.

- The index of an array numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8 (See Fig. 1.22).

Fig. 1.22: Element of Array

- The types of arrays include **Single or One Dimensional (1D) array** is used to represent and store data in a linear form so it is also called as linear array. Array having only one subscript variable is called one dimensional array. A **Two Dimensional (2D) array**, also known as a table or a matrix, where each element associates with a pair of indexes.
- We conceptualize a two-dimensional array as a rectangular grid of elements divided into rows and columns, and we use the (row, column) notation to identify a specific element.

1.8.1 One Dimensional (1D) Array

- A Java array is a collection of variables of the same type. An array is a container object that holds a fixed number of values of a single type.
- One Dimensional (1D) array in java is always used with only one subscript ([ ]). Each item in an array is called an element, and each element is accessed by its numerical index.
- As shown in Fig. 1.23, numbering begins with 0. The 4th element, for example, would therefore be accessed at index 3.

Fig. 1.23: One Dimensional (1D) Array

- Creating on 1D array involves following three steps:

Step 1 : **Declaration of Arrays:** Before using the array, we must declare it. 1D array declared in following two forms:

Syntax/Form 1: `type array_name[];`

Syntax/Form 2: `type[] array_name;`

Examples:
```
int num[];
float[] avg;
```

We do not enter the size of the arrays in the declaration.

Step 2 : **Creation of Arrays OR Creating Memory Locations for Array Elements:**

After declaring an array memory is created with the help of "new" operator.

Syntax: `array_name = new type[size];`

Examples:
```
num = new int[5];
avg = new float [5];
```

Above lines create necessary memory locations for the arrays 'num' and 'avg'.

It is possible to combine the two steps declaration and creation into one step.

```
int num[] = new int[5];
type array_name[] = new type[size];
```

Fig. 1.24: Creation of an Array and Array Elements in Memory

Step 3 : **Initialization of an Array OR Putting Values into the Memory Locations of Array Elements:**

The final step is to put values into the array created. This process is known as "initialization". Java creates arrays starting with the subscript of "0" and end with value "1" less than size specified.

The array initialization can be done in following two ways:

1. **By using array subscripts:** The syntax is given below:

```
array_name[subscript] = value;
```

Example:

```
num[0] = 10;
num[1] = 20;
num[2] = 30;
num[3] = 40;
num[4] = 50;
```

2. **Array can be created at the time of declaration:** The syntax is given below:

```
type array_name[] = {list of value};
```

Here, the array initialize is a list of values separated by commas and surrounded by curly braces. The size is not given the compiler allocates enough space for all the elements specified in the list.

Example: `int num[ ] = {10, 20, 30, 40, 50};`

It is also possible to assign an array object to another.

```
int x[ ] = {1, 2, 3};
int b[ ];
    b = x;
```

Program 1.32: Program for One Dimensional 1D array.

```
public class arraydemo
{
   public static void main(String args[])
   {
      int a[ ]={2,4,6,3,1};
      System.out.println("Number of elements in array A: "+a.length);
      System.out.println("Elements in array A");
      for(int i=0;i <a.length;i++)
      System.out.print (a[i] + "\t");
   }
}
```

Output:

```
Number of elements in array A: 5
Elements in array A
2       4      6       3       1
```

1.8.2 Two Dimensional (2D) Array

- The Two Dimensional (2D) Array in Java programming language is nothing but an array of arrays.
- The 2D array is organized as matrices which can be represented as the collection of rows and columns. Two dimensional array is declared as follows:

```
data_type[][] array_name = new int[row_size][column_size];
```

Where,

data_type will decide the type of elements it will accept.
array_name is the name you want to give it to this Java two dimensional array.
row_size is number of row elements an array can store.
column_size is number of column elements an array can store.

- There are different ways we can declaring and creating a two dimensional array in Java:

```
int[][] Student_Marks = new int[3][3];
OR
int[][]Student_Marks = { {10, 20, 30}, {15, 25, 35}, {22, 44, 66}};
```

Program 1.33: Program for 2D array.

```
public class twodim
{
   final static int r = 3;
   final static int c = 3;
   public static void main(String [] args)
   {
    int mult[] [] = new int [r][c];
    int row, column,
    int i, j;
    for (i=1; i<=r; i++)
    {
       for (j=1; j<=c; j++)
       {
          mult [i][j] = i * j;
          System.out.println(" " + mult [i][j]);
       }
    }
    System.out.println("  ");
   }
}
```

Output:

```
The given matrix:
1 2 3
2 4 6
3 6 9
```

Program 1.34: Program addition of two matrices.

```
public class Testarray
{
   public static void main(String args[])
   {
   //creating two matrices
   int a[][]={{1,3,4},{3,4,5}};
   int b[][]={{1,3,4},{3,4,5}};
   //creating another matrix to store the sum of two matrices
   int c[][]=new int[2][3];
   //adding and printing addition of Two matrices
   for(int i=0;i<2;i++)
   {
      for(int j=0;j<3;j++)
      {
         c[i][j]=a[i][j]+b[i][j];
         System.out.print(c[i][j]+" ");
      }
         System.out.println();//new line
      }
   }
}
```

Output:

```
2   6   8
6   8   10
```

Program 1.35: Program for sort in array.

```
import java.util.Arrays;
public class SortIntArrayExample
{
   public static void main(String[] args)
   {
      //create an int array
      int[] i1 = new int[]{3,2,5,4,1};
      //print original int array
      System.out.print("Original Array: ");
      for(int index=0; index < i1.length ; index++)
      System.out.print(" " + i1[index]);
      Arrays.sort(i1);
```

```
        //print sorted int array
        System.out.print("Sorted int array: ");
        for(int index=0; index < i1.length ; index++)
        System.out.print(" " + i1[index]);
        int[] i2 = new int[]{5,2,3,1,4};
        Arrays.sort(i2,1,4);
        //print sorted int array
        System.out.print("Partially Sorted int array: ");
        for(int index=0; index < i2.length ; index++)
        System.out.print(" " + i2[index]);
    }
}
```

Output:

```
Original Array: 3 2 5 4 1
Sorted int array: 1 2 3 4 5
Partially Sorted int array: 5 1 2 3 4
```

Program 1.36: Program to Reverse elements in array.

```
import java.util.*;
public class ReverseArray
{
    public static void main(String args[])
    {
        // Create java.util.Scanner object for taking input
        Scanner s=new Scanner(System.in);
        // Take no.of elements and store it in n
        System.out.println("Enter the no.of elements");
        int n=s.nextInt();
        // Create array of size n
        int a[]=new int[n];
        // Read elements into the array
        System.out.println("Enter the elements into the array");
        for(int i=0;i<n;i++)
        {
            a[i]=s.nextInt();
        }
        // Reverse elements in the array
        reverse(a);
        // Print the array
```

```
        for(int i=0;i<n;i++)
        {
            System.out.printf("a[%d]=%d\n",i,a[i]);
        }
    }
    public static void reverse(int[] a)
    {
        // Loop for length/2 times only else re-swapping takes place
            for(int i=0;i<a.length/2;i++)
            {
                int temp=a[i];
                a[i]=a[(a.length-1)-i];
                a[(a.length-1)-i]=temp;
            }
        }
    }
```

Output:

```
Enter the no. of elements
4
Enter the elements into the array
1
2
3
4
a[0]=4
a[1]=3
a[2]=2
a[3]=1
```

Program 1.37: Program for transpose matrix.

```
public class TransposeaMatrix
{
    public static void main(String args[])
    {
        int m, n, c, d;
        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 matrix[][] = new int[m][n];
        System.out.println("Enter the elements of matrix:");
```

```
        for ( c = 0 ; c < m ; c++ )
            for ( d = 0 ; d < n ; d++ )
                matrix[c][d] = in.nextInt();
        int transpose[][] = new int[n][m];
        for ( c = 0 ; c < m ; c++ )
        {
            for ( d = 0 ; d < n ; d++ )
                transpose[d][c] = matrix[c][d];
        }
        System.out.println("Transpose of entered matrix:-");
        for ( c = 0 ; c < n ; c++ )
        {
            for ( d = 0 ; d < m ; d++ )
                System.out.print(transpose[c][d]+"\t");
            System.out.print("\n");
        }
    }
}
```

Output:

```
Enter the number of rows and columns of matrix:
2  3
Enter the elements of matrix:
1  2  3
4  5  5
Transpose of entered matrix:
1  4
2  5
3  5
```

1.9 ACCEPTING INPUT [April 16, Oct. 17]

- Java provides different ways to get or accept input from the user. In Java, there are different ways for reading input from the user in the command line environment (console).
- The three common techniques for accepting input in Java are using BufferedReader class, Scanner class and Command line arguments.

Using BuferedReader Class:

- The Java.io.BufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.

- Following are the important points about BufferedReader:
  1. The buffer size may be specified, or the default size may be used.
  2. Each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream.
  3. By wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

Class Declaration of BufferReader:

```
public class BufferedReader
extends Reader
```

Class Constructors:

1. `BufferedReader(Reader in)`: This constructor creates a buffering character-input stream that uses a default-sized input buffer.
2. `BufferedReader(Reader in, int sz)`: This constructor creates a buffering character-input stream that uses an input buffer of the specified size

Class Methods of BufferedReader:

1. `int read()`: This method reads a single character.
2. `void close()`: This method closes the stream and releases any system resources associated with it.
3. `void reset()`: This method resets the stream.
4. `void mark(int readAheadLimit)`: This method marks the present position in the stream.
5. `boolean ready()`: This method tells whether this stream is ready to be read.
6. `boolean markSupported()`: This method tells whether this stream supports the mark() operation, which it does.
7. `long skip(long n)`: This method skips characters.
8. `int read(char[] cbuf, int off, int len)`: This method reads characters into a portion of an array.
9. `String readLine()`: This method reads a line of text.

Program 1.38: Program for BufferReader class.

```
import java.io.*;
public class ExampleBufferedReader
{
    public static void main(String args[])throws Exception
    {
     BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
     String name="";
     System.out.println("Enter data: ");
     name=br.readLine();
     System.out.println("data is: "+name);
     }
}
```

Output:

```
Enter data:
Hello World
data is: Hello World
```

Using Command Line Arguments:

- The Java command line argument is an argument i.e. passed at the time of running the Java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- An argument which is supplied/passed to the main() from the command line at the time of calling the main() by the java interpreters is called command line argument.
- The arguments supplied from the command line are of string type arguments. Therefore these arguments are stored in the parameter of main() which is an array of string objects i.e. args[].

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

- Each element in the array is one command-line argument.

Program 1.39: Program to display all command line arguments.

```
public class commdLineArg
{
   public static void main(String args[])
   {
      System.out.println ("Total arguments passed are" + args.length);
      System.out.println ("Each of them are");
      for(int k = 0; k < args.length; k++)
      System.out.println ("arg[" + k + "]:" + args [k]);
   } //end main
} //end class
```

Output:

```
java commdLineArg I am a girl
Total argument's passed are 4
Each of them are:
   arg[o] = I
   arg[1]: am
   arg[2]: a
   arg[3]: girl
```

Using Scanner Class:

- This is probably the most preferred method to take input. The Scanner class is used to read Java user input.

- The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.
- Java Scanner class allows the user to take input from the console. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.
- In order to use the object of Scanner, we need to import java.util.Scanner package. Java Scanner class provides the following methods to read different primitives types:

Method	Description
`int nextInt()`	It is used to scan the next token of the input as an integer.
`float nextFloat()`	It is used to scan the next token of the input as a float.
`double nextDouble()`	It is used to scan the next token of the input as a double.
`byte nextByte()`	It is used to scan the next token of the input as a byte.
`String nextLine()`	Advances this scanner past the current line.
`boolean nextBoolean()`	It is used to scan the next token of the input into a Boolean value.
`long nextLong()`	It is used to scan the next token of the input as a long.
`short nextShort()`	It is used to scan the next token of the input as a Short.

Program 1.40: Program for java Scanner nextInt().

```
import java.util.Scanner;
public class Sample1
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);  // creates a Scanner object
      System.out.println("Enter an integer: ");
      int number = input.nextInt(); // reads an int value
      System.out.println("Number using nextInt(): " + number);
      input.close();
   }
}
```

Output:

```
Enter an integer: 22
Number using nextInt(): 22
```

PROGRAMS

Program 1: Program to display message.

```
import java.io.*;
class MsgExample
{
   public static void main(String s[])
   {
      System.out.println("Welcome to Java World ");
   }
}
```

Output:

```
Welcome to Java World
```

Program 2: Program to display string.

```
import java.io.*;
class FirstExample
{
   public static void main(String s[])
   {
      String str = "Object Oriented Programming – I ";
      System.out.println("String = "+str);
   }
} //end of class
```

Output:

```
String = Object Oriented Programming – I
```

Program 3: Program to constant integer number.

```
import java.io.*;
class JConstant
{
   public static void main(String s[])
   {
      int a,b;
      a = 400;
      b = 100;
      System.out.println("a = "+a);
      System.out.println("b = "+b);
   }
}//end of class
```

Output:

```
a = 400
b = 100
```

Program 4: Program to float data type.

```
import java.io.*;
class JFloat
{
   public static void main(String s[])
   {
      int a=100;
      float f=a;
      System.out.println("a = "+a);
      System.out.println("f = "+f);
   }
}
```

Output:

```
a = 100
f = 100.0
```

Program 5: Program of addition of numbers.

```
import java.io.*;
class Jadd
{
   public static void main(String s[])
   {
      int a,b,sum;
      a = 88;
      b = 77;
      sum = a + b;
      System.out.println("a = "+a);
      System.out.println("b = "+b);
      System.out.println("Sum= "+sum);
   }
}end of class
```

Output:

```
a = 88
b = 77
Sum = 165
```

Program 6: Program for command line argument.

```
class ExampleCommandLine
{
   public static void main(String args[])
   {
      System.out.println("Enter the Data: "+args[0]);
   }
} // end of class
```

Output:

```
java ExampleCommandLine 22
Enter the Data: 22
java ExampleCommandLine JAVA
Enter the Data: JAVA
```

Program 7: Program to get the number from command line arguments.

```
class Sample
{
   public static void main(String args[])
   {
      int x = Integer.parseInt(args[1]);  // convert to integer and it will
                                          store in x
      int y = Integer.parseInt(args[0]);  //convert to integerand it will
                                          store in y
      System.out.println("x = "+x);
      System.out.println("y = "+y);
   }
}//end of class
```

Output:

```
x = 2
y = 4
```

Program 8: Program to read a line using Scanner.

```
import java.util.Scanner;
class Examplescanner
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in);  // creates an object of
                                                  Scanner
      System.out.print("Enter your name: ");
```

```
        String strname = input.nextLine();       // takes   input   from   the
                                                    keyboard
        System.out.println("Name is " + strname);  // prints the name
        input.close(); // closes the scanner
    }
}
```

Output:

```
Enter your name: rose
Name is rose
```

Program 9: Program for Scanner nextInt().

```
import java.util.Scanner;
class Sample1
{
    public static void main(String[] args)
    {
        Scanner input = new Scanner(System.in);  // creates a Scanner object
        System.out.println("Enter an integer: ");
        int number = input.nextInt(); // reads an int value
        System.out.println("Number using nextInt(): " + number);
        input.close();
    }
}
```

Output:

```
Enter an integer: 22
Number using nextInt(): 22
```

Program 10: Program to reading data from console.

```
import java.io.*;
public class ExampleBufferedReader
{
    public static void main(String args[])throws Exception
    {
        BufferedReader br=new BufferedReader
                                        (new InputStreamReader(System.in));
        String name="";
        System.out.println("Enter data: ");
        name=br.readLine();
        System.out.println("data is: "+name);
    }
}
```

Output:

```
Enter data:
Hello World
data is: Hello World
```

Program 11: Program to check a leap year on not.

```
import java.util.Scanner;
class EvenOdd
{
   public static void main(String[] args)
   {
      Scanner reader = new Scanner(System.in);
      System.out.print("Enter a number: ");
      int year = reader.nextInt();
      if (((year % 4 == 0) && (year % 100!= 0)) || (year%400 == 0))
      System.out.println(year + " is  Leap Year");
      else
      System.out.println(year  + " is NOT Leap Year ");
   }
}
```

Output:

```
Enter a number: 2020
2020 is Leap Year
```

Program 12: Program to display maximum number.

```
import java.util.Scanner;
class Testmax
{
   public static void main(String s[])
   {
      Scanner reader = new Scanner(System.in);
      int a,b,c;
      System.out.println("Enter the values");
         a = reader.nextInt();   // Enter value  of a
         b = reader.nextInt();   // Enter value  of b
         c = reader.nextInt();   // Enter value  of c
      if(a>b && a>c)
      System.out.println("Maximum Number ="+a);
      if(b>a && b>c)
      System.out.println("Maximum Number ="+b);
      else
      System.out.println("Maximum Number ="+c);
   }
}
```

Output:

```
Enter the values
3
5
2
Maximum Number =5
```

Program 13: Program to display even numbers.

```
public class Examplewhile
{
   public static void main(String[] args)
   {
      int i =2;        // initalization of variable
      while(i<=10)     // check the condition
      {
         System.out.println(i);
         i+=2;         //increment variable by 2
      }
   }
}
```

Output:

```
2
4
6
8
10
```

Program 14: Program for reverse a number.

```
import java.util.Scanner;
public class Examplereverseno
{
   public static void main(String[] args)
   {
      int rev = 0,sum = 0;
      System.out.print("Enter the number: ");
      Scanner sc = new Scanner(System.in);
      int num = sc.nextInt();
```

```
        while(num>0)
        {
            rev = (rev *10)+ ( num%10);
            num = num /10;
        }
        System.out.println("The reverse of the given number is: " + rev);
    }
}
```

Output:

```
Enter the number: 345
The reverse of the given number is: 543
```

Program 15: Program for pattern.

```
import java.util.Scanner;
public class Pattern2
{
    public static void main(String[] args)
    {
        int lim;
        Scanner sc = new Scanner(System.in); // creates an object of Scanner
        System.out.print("Enter your name: ");
        lim = sc.nextInt();   // takes input from the keyboard
        for(int i=lim;i>0;i--)
        {
            for(int j = 1; j<=i ;j++)
            {
                System.out.print(" "+j);
            }//j loop
            System.out.println();//new line
        }//i loop
    }
} //end of class
```

Output:

```
Enter your name: 5
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Program 16: Program for pattern.

```
import java.util.Scanner;
public class Pattern3
{
   public static void main(String[] args)
   {
      int lim;
      Scanner sc = new Scanner(System.in);  // creates an object of Scanner
      System.out.print("Enter your name: ");
      lim = sc.nextInt();   // takes input from the keyboard
      char ch = 'A';
      for(int i=1 ;i<=lim;i++)
      {
         for(int j = 1; j<=i ;j++)
         {
            System.out.print(" "+ch);
         }//j loop
         ch++;  //increment by 1
         System.out.println();//new line
      }
   }  //i loop
} //end of class
```

Output:

```
Enter your name: 5
A
B B
C C C
D D D D
E E E E E
```

Program 17: Program to display elements of the array.

```
 public class Readarr
 {
   public static void main(String[] args)
   {
      int arr[]={11,22,44,55,77};  //Declaring an array
      for(int i=0;i<arr.length;i++)
      {
         System.out.println("arr [" + i + "]  = " + arr[i]);
                                                  //display with indexing
      }
      System.out.println("Array Elements are: ");
```

```
        for(int i=0;i<arr.length;i++)
        {
           System.out.print(" " + arr[i]);   //display elements
        }
     }
  }
```

Output:

```
arr [0]  = 11
arr [1]  = 22
arr [2]  = 44
arr [3]  = 55
arr [4]  = 77
Array Elements are:
11 22 44 55 77
```

Program 18: Program to print odd and even numbers from an array.

```
public class Exoddevenarray
{
   public static void main(String args[])
   {
      int arr[]={10,23,54,61,32,22};  //Declaration of Array
      System.out.println("Even Array Elements:");
      for(int i=0;i<arr.length;i++)
      {
         if(arr[i]%2==0)
         {
            System.out.println(arr[i]);
         }
      }
      System.out.println("Odd Array Elements:");
      for(int i=0;i<arr.length;i++)
      {
         if(arr[i]%2!=0)
         {
            System.out.println(arr[i]);
         }
      }
   }
}  //end of class
```

Output:

```
Even Array Elements:
10
54
32
22
Odd Array Elements:
23
61
```

Program 19: Program to display the array element in descending order.

```
import java.util.Scanner;
public class Exdessort
{
   public static void main(String[] args)
   {
      int n,temp;
      Scanner sc=new Scanner(System.in);
         System.out.print("Enter the Limit: ");
         n=sc.nextInt();  //enter the number of elements
         int[] array = new int[10];  //declare the array element
         System.out.println("Enter the elements of the array: ");
      for(int i=0; i<n; i++)
      {
         array[i]=sc.nextInt();  //reading array elements from the user
      }
      System.out.println("Array elements are: ");
      for (int i=0; i<n; i++)
      {
         System.out.print(" "+array[i]);
                              // accessing array elements using the for loop
      }
      for (int i=0; i<n; i++)
      {
         for (int j=i+1; j<n; j++)
         {
            if(array[i]<array[j])
            {
```

```
                    temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
            }
        }
        System.out.println(" \nSorted Array elements are: ");
        for (int i=0; i<n; i++)
        {
            System.out.print(" "+array[i]);
        }
    }
} //end of class
```

Output:

```
Enter the Limit: 5
Enter the elements of the array:
78
5
91
2
15
Array elements are:
 78 5 91 2 15
Sorted Array elements are:
 91 78 15 5 2
```

Program 20: Program to read and display 2D array.

```
class Example2d
{
    public static void main(String[] args)
    {
        int[][] arr = { { 1, 2, 3 }, { 4, 5, 6 },{7, 8, 9 } };
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                System.out.println("arr[" + i + "][" + j + "] = " + arr[i][j]);
            }//j
        }//i
```

```
        System.out.println("Array Elements: ");
        for (int i = 0; i < 3; i++)
        {
           for (int j = 0; j < 3; j++)
           {
              System.out.print(" " + arr[i][j]);
           }//j
           System.out.println();
        }//i
     }
  }
```

Output:

```
arr[0][0] = 1
arr[0][1] = 2
arr[0][2] = 3
arr[1][0] = 4
arr[1][1] = 5
arr[1][2] = 6
arr[2][0] = 7
arr[2][1] = 8
arr[2][2] = 9
Array Elements:
1 2 3
4 5 6
7 8 9
```

Program 21: Program to read and display 2D array using Scanner.

```
import java.util.Scanner;
public class Example2d1
{
   public static void main(String args[])
   {
      int m, n, i, j;
      Scanner sc=new Scanner(System.in);
      System.out.print("Enter the number of rows: ");
      m = sc.nextInt();             //enter row
      System.out.print("Enter the number of columns: ");
      n = sc.nextInt();             //enter column
```

```
        int array[][] = new int[m][n]; // Declaring the two-dimensional array
        System.out.println("Enter the elements of the array: ");
                                       // enter the array values
        for (i = 0; i < m; i++)        //loop for row
        {
           for (j = 0; j < n; j++)     //loop for column
           {
              array[i][j] = sc.nextInt();
           }//j
        }
        System.out.println("Elements of the array are: ");
                                       //accessing array elements
        for (i = 0; i < m; i++)
        {
           for (j = 0; j < n; j++)
           {
              System.out.print(array[i][j] + " "); //prints the array elements
           }
           System.out.println();   // new line
        }
     }
  } //end of class
```

Output:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the array:
11
22
33
44
55
66
77
88
99
Elements of the array are:
11 22 33
44 55 66
77 88 99
```

Program 22: Program to addition of 2 dimensional array.

```
import java.util.Scanner;
class Example2dadd
{
   public static void main(String args[])
   {
      int row, col,i,j;
      Scanner in = new Scanner(System.in);
      System.out.println("Enter the number of rows");
      row = in.nextInt();
      System.out.println("Enter the number columns");
      col = in.nextInt();
         int a[][] = new int[row][col];
         int b[][] = new int[row][col];
         int c[][] = new int[row][col];
      System.out.println("Enter the elements of 'a' matrix");
      for ( i= 0 ; i < row ; i++ )
      {
         for ( j= 0 ; j < col ;j++ )
         {
            a[i][j] = in.nextInt();
         }
         System.out.println();
      }
      System.out.println("Enter the elements of 'b' matrix");
      for ( i= 0 ; i < row ; i++ )
      {
         for ( j= 0 ; j < col ;j++ )
         {
            b[i][j] = in.nextInt();
         }
         System.out.println();
      }
      for ( i= 0 ; i < row ; i++ )
      {
         for ( j= 0 ; j < col ;j++ )
         {
            c[i][j] = a[i][j] + b[i][j] ;
         }
      }
```

```
        System.out.println("Matrix 'a': ");
        for ( i= 0 ; i < row ; i++ )
        {
           for ( j= 0 ; j < col ;j++ )
           {
              System.out.print(a[i][j]+" ");
           }
           System.out.println();
        }
        System.out.println("Matrix 'b': ");
        for ( i= 0 ; i < row ; i++ )
        {
           for ( j= 0 ; j < col ;j++ )
           {
              System.out.print(b[i][j]+" ");
           }
           System.out.println();
        }
        System.out.println("Addition of matrices: ");
        for ( i= 0 ; i < row ; i++ )
        {
           for ( j= 0 ; j < col ;j++ )
           {
              System.out.print(c[i][j]+"\t");
           }
           System.out.println();
        }
     }
  }//end of end
```

Output:

```
Enter the number of rows
2
Enter the number columns
2
Enter the elements of 'a' matrix
1
2
3
4
```

```
Enter the elements of 'b' matrix
5
6
7
8
Matrix 'a':
1 2
3 4
Matrix 'b':
5 6
7 8
Addition of matrices:
6       8
10      12
```

Program 23: Program to accept number from command line and print square root of the number.

```
import java.lang.*;
class squareroot
{
   public static void main(String args[])
   {
      int num;
      num = Integer.parseInt(args[0]);
      doublesq = Math.sqrt(num);
      System.out.println("Square root of "+ num +" is: "+sq);
   }
}
```

Output:

```
Square root of 16 is: 4.0
```

Program 24: Program to find sum of digit of number entered by user. OR Program to compute the sum of the digits of a given integer numbers.

```
import java.io.*;
class SumOfDigits
{
   public static void main(String args[])throws IOException
   {
      BufferedReader br = new BufferedReader(new
                                          InputStreamReader(System.in));
      System.out.println("Enter number: ");
      int num=Integer.parseInt(br.readLine());
      int remainder, result=0;
```

```
        while(num>0)
        {
            remainder = num%10;
            result = result + remainder;
            num = num/10;
        }
        System.out.println("Sum of digits of given number is: "+result);
    }
}
```

Output:

```
Enter number: 5 5 5
Sum of digits of given number is: 15
```

Program 25: Program to accept two numbers as command line arguments and print the addition of those numbers.

```
class addition
{
    public static void main(String args[])
    {
        int num1=Integer.parseInt(args[0]);
        int num2=Integer.parseInt(args[1]);
        int result=0;
        result=num1+num2;
        System.out.println("Addition of two numbers is: " +result);
    }
}
```

Output:

```
10 20
Addition of two numbers is: 30
```

Program 26: Program to find whether number is palindrome or not.

```
class palindrome
{
    public static void main(String args[])
    {
        int num=Integer.parseInt(args[0]);
        int n=num;
        int reverse=0,remainder;
        while(num>0)
        {
            remainder=num%10;
            reverse=reverse*10+remainder;
            num=num/10;
        }
```

```
        if(reverse==n)
           System.out.println(n+"is a palindrome number");
        else
           System.out.println(n+" is not palindrome number");
     }
  }
```

Output:

```
121
121 is a palindrome number
```

Program 27: Program to generate Fibonacci series 1 1 2 3 5 8 13 21 34 55 89.

```
class Fibonacci
{
   public static void main(String args[])
   {
      int num=Integer.parseInt(args[0]);
      System.out.println("Fibonacci series are: ");
      int f1,f2=0,f3=1;
      for(int i=1;i<=num;i++)
      {
         System.out.print(f3+" ");
         f1=f2;
         f2=f3;
         f3=f1+f2;
      }
   }
}
```

Output:

```
11
Fibonacci series are:
1 1 2 3 5 8 13 21 34 55 89
```

Program 28: Program to check whether an entered number is prime or not.

```
import java.io.*;
class PrimeNo
{
   public static void main(String args[]) throws IOException
   {
      BufferedReader br=new BufferedReader(new
      InputStreamReader(System.in));
      System.out.println("Enter number: ");
      int num = Integer.parseInt(br.readLine());
      int flag=0;
```

```
        for(int i=2;i<num;i++)
        {
            if(num%i==0)
            {
                System.out.println(num + " is not a prime number");
                flag=1;
                break;
            }
        }
        if(flag==0)
        System.out.println(num + " is a prime number");
    }
}
```

Output:

```
Enter number:5
5 is a prime number
```

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. An expression involving byte, int, and literal numbers is promoted to which of these?
   (a) long (b) int
   (c) byte (d) float
2. Which of following literals can be contained in float data type variable?
   (a) − 1.7e + 308 (b) − 3.4e + 038
   (c) + 1.7e + 308 (d) − 3.4e + 050
3. With x = 0, which of the following are legal lines of Java code for changing the value of x to 1?
   (i) x++; (ii) x = x + 1; (iii) x + = 1; (iv) x = + 1;
   (a) (i), (ii) and (iii) (b) (i) and (iv)
   (c) (ii) and (iii) (d) (i), (ii), (iii) and (iv)
4. What is true about a break?
   (a) break stops the execution of entire program
   (b) break forces the control out of the loop and starts the execution of next iteration
   (c) break halts the execution and forces the control out of the loop
   (d) break halts the execution of the loop for certain time frame
5. Which of the following is used with the switch statement?
   (a) break (b) continue
   (c) exit (d) do

6. Which component is used to compile, debug and execute java program?
   (a) JVM (b) JRE
   (c) JIT (d) JDK
7. Which of the below is invalid identifier with the main method?
   (a) public (b) static
   (c) private (d) final
8. What is the extension of compiled java classes?
   (a) .class (b) .java
   (c) .txt (d) .exe
9. Which of the following is a valid declaration of an object of class Box?
   (a) Box obj = new Box(); (b) Box obj = new Box;
   (c) obj = new Box(); (d) new Box obj;
10. Which of the following is not a valid flow control statement?
   (a) continue (b) return
   (c) exit() (d) break
11. What software compiles a Java program?
   (a) JRE (Java Runtime Environment) (b) JDK (Java Developer Kit)
   (c) JVM (Java Virtual Machine) (d) Command Prompt
12. Byte code is the result of?
   (a) Compiling a Java file (b) Compiling a Class file
   (c) Interpreting a Java File (d) Interpreting a Class file
13. What is portability offered by Java language?
   (a) Small code size easy to carry occupying less disk space
   (b) Generating suitable Byte Code for each machine by the Compiler
   (c) Ability to run the Byte on different machines producing the same behavior and output
   (d) Java does not actually provide portability
14. A function in C language is similar to what in Java language?
   (a) Method (b) Member
   (c) Variable (d) None of the above
15. What is the need to mention "static" before main method?
   (a) To call main method without creating an object of class
   (b) To make main method as class method common to all instances
   (c) Both (a) and (b)
   (d) None of the above
16. What is the output of the below Java code snippet?

```
int a = 2 - - 7;
System.out.println(a);
```

   (a) – 5 (b) 10
   (c) 9 (d) Compiler Error
17. Increment and Decrement arithmetic operators in Java has which Associativity?
   (a) Left to Right (b) Right to Left
   (c) Left to Left (d) Right to Right

18. What is the output of the Java code snippet?

```
int a = 10, b = 20;
int c = a++*2;
int d = --b*2;
System.out.println(c +"," + d);
```

(a) 20, 40 (b) 22, 40
(c) 20, 38 (d) 22, 38

19. What is meant by javap?
(a) java compiler (b) java Interpreter
(c) java debugger (d) java Disassemble

20. How can we identify whether a compilation unit is class or interface from a .class file?
(a) Java source file header
(b) Extension of compilation unit
(c) We cannot differentiate between class and interface
(d) The class or interface name should be postfixed with unit type

21. What is use of interpreter?
(a) They convert bytecode to machine language code
(b) They read high level code and execute them
(c) They are intermediated between JIT and JVM
(d) It is a synonym for JIT

22. Which is a high level, object-oriented, platform independent programming language?
(a) C (b) C++
(c) Java (d) SmallTalk

23. Which is a software or hardware environment in which we can run a program?
(a) Computing platform (b) Computer framework
(c) Computer IDE (d) Computer editor

24. Which is an engine or a specification which provides a runtime environment to execute the Java bytecodes or applications?
(a) Java Runtime Environment (b) Java Virtual Machine (JVM)
(c) Java Standard Library (JSL) (d) SmallTalk

25. Which is an instance of class?
(a) Method (b) Polymorphism
(c) Inheritance (d) Object

26. The single-line comment in Java is denoted by,
(a) // (b) /*...*/
(c) /** ... */ (d) None of mentioned

27. Which is a collection of objects of similar type of objects?
(a) Object (b) Class
(c) Inheritance (d) Polymorphism

28. Binding of data and functions into a single unit known as,
    (a) Object (b) Class
    (c) Encapsulation (d) Polymorphism
29. Which a request for execution of a method?
    (a) Object (b) Class
    (c) Message (d) Polymorphism
30. Java environment includes,
    (a) JSL (b) JDK
    (c) Both (a) and (b) (d) None of mentioned
31. Which is a Java application launcher?
    (a) java (b) javac
    (c) javadoc (d) jdb
32. Primitive data types of Java includes,
    (a) int (b) char
    (c) float (d) All of mentioned
33. Which operator is used only for object reference variables?
    (a) instanceof (b) dot
    (c) ternary (d) None of mentioned
34. Which is an identifier who's values can changed during execution of program?
    (a) constant (b) variable
    (c) operator (d) literal
35. To declare a final variable, following which keyword is used.
    (a) var final (b) final
    (c) const (d) finalize
36. Which is an expression is a combination of operators and operands which reduces to a single value.
    (a) Statement (b) Operator
    (c) Expression (d) None of mentioned
37. Following which is a multi-way selection statement.
    (a) if (b) if else
    (c) switch (d) None of mentioned
38. Which statements allows to execute a statement or group of statements multiple times until conditions become true?
    (a) sequential (b) looping
    (c) selection (d) decision
39. Following which statement causes the program control to transfer back to the caller of a method.
    (a) return (b) break
    (c) continue (d) exit
40. Which is a collection of similar type of elements that have contiguous memory location?
    (a) Object (b) Array
    (c) Class (d) Method

41. The array having only one subscript variable is called as,
    (a) One Dimensional (1D) array (b) Two Dimensional (2D) array
    (c) Multi-dimensional array (d) None of mentioned
42. The three common techniques for accepting input in Java includes,
    (a) BufferedReader class (b) Scanner class
    (c) Command line arguments (d) All of mentioned
43. Which Java class allows the user to take input from the console?
    (a) BufferReader (b) Scanner
    (c) Command line parameters (d) All of mentioned

Answers

1. (b)	2. (b)	3. (d)	4. (c)	5. (a)	6. (d)	7. (c)	8. (a)	9. (a)	10. (c)
11. (b)	12. (a)	13. (c)	14. (a)	15. (c)	16. (c)	17. (b)	18. (c)	19. (d)	20. (a)
21. (b)	22. (c)	23. (a)	24. (b)	25. (d)	26. (a)	27. (b)	28. (c)	29. (c)	30. (c)
31. (a)	32. (d)	33. (a)	34. (a)	35. (b)	36. (c)	37. (c)	38. (b)	39. (a)	40. (b)
41. (a)	42. (d)	43. (b)							

Q.II Fill in the Blanks:

1. _____ and do while loops can be used where the number of iteration is not known before hand.
2. _____ is a concept in OOP which hides unimportant implementation details from other.
3. _____ method can be called without the instance of a class.
4. Operator _____ is used to crate an array.
5. _____ variables are nothing but constants.
6. A break statement inside a loop like while, for, do while and enhanced-for causes the program execution _____ loop.
7. In Java language, break or continue statements can be implemented inside a loop only with the help of _____ statements to avoid never-ending loops.
8. An enhanced for loop in Java misses _____ and Increment/ Decrement compared to the old-style for loop.
9. A break-with-label or continue-with-label are used in particular in Java to select _____ loop either to break or continue.
10. _____ is used to find and fix bugs in the Java programs.
11. Platform independent code file created from Source file is understandable by _____ .
12. Java programming was designed by _____.
13. Class file must have extension _____.
14. Java has _____ primitive types of data.
15. Integer Data type does not include following primitive data type _____.
16. _____ Gosling initiated Java language project in June 1991.
17. To write Java programs, we will need a _____ editor like Notepad++ while even more sophisticated IDEs like Eclipse, Netbeans available in the market.

18. Names used for classes, variables, and methods are called _____.
19. Arrays are objects that store multiple variables of the _____ type.
20. The _____ may not be used as constant or variable or any other identifier names.
21. The inheritance concept allows to _____ the fields and methods of the existing class without having to rewrite the code in a new class.
22. A _____ is a blueprint from which individual objects are created.
23. The _____keyword is used to create the object.
24. _____are nothing but reserved memory locations to store values.
25. _____ data types are predefined by the language and named by a keyword like int. float, char and so on.
26. A variable provides us with _____ storage that the programs can manipulate.
27. Java provides a number of _____ modifiers to set access levels for classes, variables, methods and constructors.
28. In _____ statement, if the Boolean expression evaluates to true then the block of code inside the if statement will be executed.
29. _____ is the ability of an object to take on many forms.
30. The _____ is the interpreter which launches a java program.
31. A _____ statement allows us to execute a statement or group of statements multiple times.
32. In Java the _____ command disassembles a class file.
33. The type of value that a variable can hold is called as _____ type.
34. _____ refer to fixed values that do not change during the execution of a program.
35. OOP emphasis on _____ rather than procedure.
36. An object is an _____ of a class and has two general characteristics namely, state and behavior.
37. We access an array element by referring to the _____ number.
38. The _____ keyword causes the loop to immediately jump to the next iteration of the loop.
39. The while loop loops through a block of code as long as a specified condition is _____.
40. _____ refers to act of representing essential features without including background details or explanations.
41. _____ translates or converts the ".java" file to a ".class" file.
42. _____ perform some specific operations using the operands and operator.
43. The _____ stores a fixed-size sequential collection of elements of the same type.
44. Java is an _____ language, i.e. programs run directly from the source code.
45. The _____ is a software development environment which is used to develop Java applications.
46. When the _____ statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
47. _____ is Java debugger that helps us in finding errors in Java programs.

48. _____ line arguments is the way to pass argument to the main method in Java.
49. The _____ class is used to get user input, and it is found in the java.util package.

Answers

1. while	2. Encapsulation	3. static	4. []	5. Final
6. exit	7. if else	8. Initialization	9. Outer loop	10. jdb
11. JVM	12. Sun	13. .class	14. eight (8)	15. double
16. James	17. text	18. identifiers	19. same	20. keywords
21. reuse	22. class	23. new	24. Variables	25. Primitive
26. named	27. access	28. if	29. Polymorphism	30. java
31. loop	32. javap	33. data	34. Constants	35. data
36. instance	37. index	38. continue	39. true	40. Abstraction
41. javac	42. Operators	43. array	44. interpreted	45. JDK
46. break	47. Jdb	48. Command	49. Scanner	

Q.III State True or False:

1. All bitwise operations are carried out with the same level of precedence in Java.
2. Consider the statement "x = (a > b) ? a: b"; then the value of x is 27, if a = 18 and b = 27.
3. An array in the Java programming language has the ability to store many different types of values.
4. Variables declared inside a *for* loop are limited in scope to the loop.
5. The operations y >> 3 and y >>> 3 produce the same result when y > 0.
6. In Java, an array can be declared without initialization without mentioning the size.
7. We should not specify the array size if declaration and initialization are done at the same time.
8. An array in Java can be declared only of some predefined types.
9. A WHILE loop in Java executes the statements at least once even the condition is not satisfied.
10. Allocating memory with the keyword "new" causes the array elements to carry default values.
11. A loop in Java generally contains a Loop-Counter variable.
12. The keyword "goto" can be used in Java programs with labels.
13. Java Programming promises programmer - "Write Once, Run Anywhere".
14. The name of the class file is created automatically when you compile the program with the javac command.
15. An Increment operator "++" and/or a Decrement operator "--" are used along with a Loop- Counter variable in Java.
16. Loops are in Java save time, reduce errors, and they make code more readable.
17. Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

18. An array is a blueprint or prototype from which objects are created.
19. Object-Oriented Programming (OOPs) is a programming paradigm based on the concept of “objects” that contain data and methods.
20. Abstraction is a process where we show only “relevant” data and “hide” unnecessary details of an object from the user.
21. The process by which one class acquires the properties and functionalities of another class is called inheritance.
22. In Java arrays and strings are primitive data types.
23. Java platform is a collection of programs that help programmers to develop and run Java programming applications efficiently.
24. Java is a general-purpose, object-oriented programming language.
25. Java Virtual Machine (JVM) provides a runtime environment to execute Java applications.
26. An array is a collection of similar types of data.
27. An class can be defined as an instance of a object.
28. Variable in Java is a data container that stores the data values during Java program execution.
29. Data types in Java are defined as specifiers that allocate different sizes and types of values that can be stored in the variable or an identifier.
30. Java is a platform-independent language because it has runtime environment i.e. JRE and API.
31. In computer programming, comments are a portion of the program that are completely ignored by Java compilers. They are mainly used to help programmers to understand the code.
32. Every variable is assigned data type which designates the type and quantity of value it can hold. Variable is a memory location name of the data.
33. For mobile applications, Java uses J2ME framework which is cross platform that runs applications across phones and smartphones.
34. Java Standard Edition is also known as Java 2 Platform, Standard Edition (J2SE).
35. The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applications written in the Java programming language.
36. The java creates HTML formatted documentation from Java source code files.
37. The AppletViewer tool enables to run java applets.
38. Byte code verifier checks the code fragments for illegal codes that can violate access right to the object.
39. Interpreter reads the byte code stream and then executes the instructions, step by step.
40. Scope of a variable decides its accessibility throughout the program.
41. Java keywords can be used as an identifier.
42. The if statement is a single conditional based statement that executes only if the provided condition is true.

43. Loop is designed to execute particular code block till the specified condition is false.
44. In Java, the nested if statement is a if inside another if. In this, one if block is created inside another if block when the outer block is true then only the inner block is executed.
45. The if else statement consists of conditional based cases and a default case.
46. The if-else statement is used for testing condition. If the condition is true, if block executes otherwise else block executes. The else block execute only when condition is false.
47. Break statement is used to break the current execution of the program.
48. Multi-Dimensional Array represent data into tabular form in which data is stored into row and columns.
49. The BufferedReader class of Java is used to read the stream of characters from the specified source (character-input stream).
50. The break statement is used to skip the current iteration of the loop. It jumps to the next iteration of the loop immediately.
51. Bitwise operators are used to perform operations bit by bit.
52. The command line argument is the argument that passed to a program during runtime. It is the way to pass argument to the main method in Java.
53. Initialization is a process of allocating memory to an array.
54. Logical operators are used to perform arithmetic operations like: addition, subtraction etc and helpful to solve mathematical expressions.
55. An array is a collection of similar data types.
56. Relational operators are used to test comparison between operands or values in two variables.
57. Operator is a symbol which tells to the compiler to perform some operation.
58. Assignment operators are used to assign a value to a variable.
59. Conditional operator is also known as ternary operator because it works with three operands.
60. Single dimensional array use multi index to store elements.

Answers

1. (F)	2. (T)	3. (F)	4. (F)	5. (T)	6. (T)	7. (F)	8. (F)	9. (F)	10. (T)
11. (T)	12. (F)	13. (T)	14. (T)	15. (T)	16. (T)	17. (T)	18. (F)	19. (T)	20. (T)
21. (T)	22. (F)	23. (T)	24. (T)	25. (T)	26. (T)	27. (F)	28. (T)	29. (T)	30. (T)
31. (T)	32. (T)	33. (T)	34. (T)	35. (T)	36. (F)	37. (T)	38. (T)	39. (T)	40. (T)
41. (F)	42. (T)	43. (T)	44. (F)	45. (F)	46. (T)	47. (T)	48. (T)	49. (T)	50. (F)
51. (T)	52. (T)	53. (T)	54. (F)	55. (T)	56. (T)	57. (T)	58. (T)	59. (F)	60. (T)

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. What is OOP?
2. What is Java?

3. Why Java is platform-neutral language?
4. What is object?
5. Define class.
6. What is encapsulation?
7. Define inheritance.
8. Give any two features of OOP.
9. Define constant.
10. What is variable?
11. What is data type?
12. List types of data types in Java.
13. What is an operator?
14. What is the use of Scanner class?
15. What is the purpose of loop statements?
16. List loop control statements with its uses.
17. What is switch statement?
18. What is javac?
19. What is the purpose of jdb?
20. What is JRE?
21. Define OOP.
22. How to run Java program?
23. Why Java is called portable?
24. Give the use of BufferedReader class.
25. What is the function of class loader?
26. What is a byte code?
27. What is comment?

(B) Long Answer Questions:

1. Give short history of Java.
2. Explain features of Java.
3. Explain Java environment in detail.
4. Write short note on:
   (i) javac (ii) jar
   (iii) appletviewer (iv) jdb
   (v) javadoc (vi) javap
5. What is object oriented programming approach?
6. Why Java is called distributed and dynamic?
7. What is the use of Byte code verifier and class loader?
8. Which are the tasks performed by Java environment?
9. What is the syntax of doc comment?
10. What is object? How to create it? Explain with example.
11. What is class? Why class is called as blueprints for an object?
12. Differentiate between class and object.

13. What is meant by the data abstraction?
14. Write short note on: Encapsulation.
15. Describe inheritance concept with example.
16. Enlist various features of OOP.
17. What are the advantages and disadvantages of OOP?
18. What are keywords used by Java?
19. Define the following terms:
    (i) OOP, (ii) Class, (iii) Object, (iv) Operator, (v) Variable, (vi) Constant, (vii) Loop, (viii) Statement.
20. Enlist various operators used by Java with example.
21. What is data type? How to use data types in Java? Explain with example.
22. What are the types of constant used by Java?
23. Describe the statements with example: (i) if, (ii) if else, (iii) for, (iv) while.
24. Which jump statements used by java programming language? Explain them with example.
25. Write short note on: Scanner class.
26. What is variable? How to declare it? Explain with example.
27. With the help of example describe conditional operator.
28. Explain scope of variables in detail.
29. Describe command line arguments with example.
30. Write program to calculate average of three numbers.
31. Write program for fibbonacci series.
32. Write program to find give number is Armstrong or not.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. Define final variable with an example. **[1 M]**
Ans. Refer to Section 1.7.5.
2. Name any four tags of Javadoc. **[1 M]**
Ans. Refer to Section 1.6.5.
3. What is AppletViewer? **[1 M]**
Ans. Refer to Section 1.6.3.
4. Explain features of Java in detail. **[5 M]**
Ans. Refer to Section 1.3.
5. Explain various ways to read input string from console or keyboard. **[5 M]**
Ans. Refer to Section 1.9.

October 2016

1. What is Javap? **[1 M]**
Ans. Refer to Section 1.6.6.
2. What is Array? How to create an array of elements of double type? **[1 M]**
Ans. Refer to Section 1.8.

April 2017

1. What is Javadoc? **[1 M]**

Ans. Refer to Section 1.6.5.

2. List primitive data types in Java with their width in bits. **[1 M]**

Ans. Refer to Section 1.7.2 Point (2).

3. Why is Java called purely object oriented programming language? Explain features of Java. **[5 M]**

Ans. Refer to Section 1.1.

October 2017

1. Justify true/false 'Java is not fully object oriented'. **[1 M]**

Ans. Refer to Section 1.1.

2. Explain the process of compilation of a Java program. **[5 M]**

Ans. Refer to Page 1.15.

3. Explain how the input is accepted from command line with the help of program. **[5 M]**

Ans. Refer to Section 1.9.

April 2018

1. What do you mean by jdb? **[1 M]**

Ans. Refer to Section 1.6.4.

2. Write a java program to find second smallest element in an array. **[5 M]**

Ans. Refer to Section 1.8.

October 2018

1. What is the use of javap tool? **[1 M]**

Ans. Refer to Section 1.6.6.

2. What is blank final variable? **[1 M]**

Ans. Refer to Section 1.7.5.

3. "Java is platform dependant language." True/False ? Justify. **[1 M]**

Ans. Refer to Section 1.3.3.

April 2019

1. What is JAR file? **[1 M]**

Ans. Refer to Section 1.6.7.

2. What is Javadoc comments? **[1 M]**

Ans. Refer to Section 1.6.5.

3. Write a Java program to find second smallest element in an array. **[5 M]**

Ans. Refer to Section 1.8.

■■■

CHAPTER 2

Objects and Classes

Objectives...

- To understand Basic Concepts of Objects and Classes
- To study Constructors and Access Specifiers
- To learn Predefined Classes in Java
- To study Concepts of Packages
- To learn Wrapper Classes

2.0 INTRODUCTION

- Java is an object-oriented programming language. Objects and classes are the basic building blocks of OOP. Everything in Java is associated with classes and objects.
- Anything we wish to represent in Java must be encapsulated in a class that defines the "state" and "behavior" of the basic program components known as objects.
- An object is a basic unit of OOP and represents the real life entities like a house, a tree etc. A class is a template for defining the behavior and states of the objects.
- Classes create objects and objects use methods to communicate between them i.e. all about object-oriented programming.
- A Java package is a mechanism for organizing Java classes. Packages are used in Java, in-order to avoid name conflicts and to control access of class, interface and so on.

2.1 OVERVIEW OF CLASS AND OBJECT

- In Object-Oriented Programming (OOP) technique, we design a program using objects and classes. An A class is a blueprint for creating objects. In Java, an object is created from a class.

Object in Java:

- A real-world entity that has state and behavior is known as an object. These real-world objects share two characteristics i.e. state and behavior. **Example:** A cat has states color, name, etc. as well as behaviors like walking and eating.
- In simple words, object is an instance of a class.

Class in Java:

- A class can be defined as "a template/blueprint that describes the behaviors/states that object of its type support".
- A class is a user-defined type which groups data members and its associated functions together.
- In Java language the data members are called fields and the functions are called methods. Calling a specific method in an object is described as sending the object of message.
- Example of class and objects: class Student has objects like Yogi, Atharva, Rudra, Amita.
- Take another example of class and objects: class Car has objects like Ford, Toyota, Volkswagaon.

Fig. 2.1: Class and Object

Defining Classes:

- A class is the basic building block of Java language. A class is a template (or blueprint) that describes the state and behavior associated with instances (objects) of that class.
- The class is composed of fields and methods. A field is a variable defined within a class definition that is associated with an instance of that class or with all instances of the class. A Java method is a set of Java statements which can be included inside a Java class.
- A class is declared by the keyword 'class' followed by class name. The declaration **syntax** is given below:

```
<access specifier> class ClassName [extends SuperClassName]
{
   [fields declaration]
   [methods declaration]
}
```

- In the above syntax the access specifier can be public, private, protected and so on. The class is a keyword and the ClassName is a user defined name. A field is a variable defined within a class definition that is associated with an instance of that class or with all instances of the class. A Java method included inside a Java class.

- The data in a class is in the form of instance variables. We can declare the instance variables exactly the same way as we declare the local variables.

  For example: Here, class name is Employee which contents two instance variables E_id and Salary.

```
class Employee
{
   int E_id;
   float Salary;
}
```

Program 2.1: Creating a simple class in Java.

```
public class Student
{
   int id;//declaration of data member or instance variable
   String name;
   public static void main(String args[]) //creating main
   {
      Student s1=new Student();//creating an object of Student
      System.out.println("Student id  = "+s1.id);
      //accessing member through reference variable
      System.out.println("Student name  = "+s1.name);
   }
}
```

Output:

```
Student id  = 0
Student name  = null
```

Methods in Java:

- A Java method is a collection of statements that are grouped together to perform a certain task.
- Method describes behavior of an object. A method is a collection of statements that are group together to perform an operation.

  Syntax:

```
return_type methodName(parameter_list)
{
   //body of method …
}
```

Example of a Method:

```
public String getName(String st)
{
   String name="Nirali Prakashan";
   name=name+st;
   return name;
}
```

- For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).
- The process of method calling is simple, when a program invokes a method, the program control gets transferred to the called method. This called method then returns control to the caller in two conditions, when:
  1. The return statement is executed.
  2. It reaches the method ending closing brace.

Program 2.2: Program to show how to define a method and how to call it.

```
public class ExampleMinNumber
{
   public static void main(String[] args)
   {
      int a = 11;
      int b = 6;
      int c = minFunction(a, b);
      System.out.println("Minimum Value = " + c);
   }
   /* returns the minimum of two numbers */
   public static int minFunction(int n1, int n2)
   {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;
      return min;
   }
}
```

Output:

```
Minimum value = 6
```

Creating Objects:

- Objects are programming units of a particular class. Creating an object of the class is also called as instantiating an object.
- There are three steps when creating an object from a class:

1. **Declaration of an Object:**

   We have to specify what type (i.e. class) the object will be. A variable declaration with a variable name with an object type.

   Syntax: `<class name> object name;`

   Where, class name is the name of the already defined class and the object name is a valid identifier.

2. **Instantiation (Creating Objects):**

   Objects are created using the 'new' keyword. This 'new' keyword creates an object of the specified class and returns the reference of that object.

   The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

   Syntax: `<objectname>=new classname([arguments]);`

   Let us create an object for the already defined Rectangle class in the previous chapter

```
Rectangle r1;       //Rectangle name of the class, r1 name of the object
r1=new Rectangle(); //Allocating memory
```

   Fig. 2.2 illustrates the above process.

Fig. 2.2: Creating object

3. **Initialization:**

- After the objects are created, there is a separate copy of the instance variables is created for each of them. So, separate memory is allocated to them.
- The new operator dynamically allocates the memory for the object i.e., it allocates the memory at run time.
- It has the following general **form/syntax**:

```
class-var = new classname();
```

  Here, the 'class-var' is a variable of the class type being created. The 'classname' is name of the class that is being instantiated.

- Here, the first statement declares a variable to hold the object reference. The second statement assigns the object reference to the variable.

Accessing Class Members:

- The dot operator (.) or member selection operator is used to access the instance variables of the class.
- We cannot access the instance variables outside of the class without referring to the object.
- Member access follows the following **syntax**:

  ```
  objectname.variablename
  ```

  Here, the 'objectname' is the name of the object and 'variablename' is name of the instance variable.
- So, the instance variables of the class Employee can be accessed as,

  ```
  Employye.E_id=1234;
  Employee.salary=30000;
  ```

Program 2.3: Program for accessing class members.

```
public class Student
{
   String name ="abc";
   int age = 20;
   void display()
   {
      System.out.println("My name is "+name);
      System.out.println("I am "+ age+ " years old");
   }
   public static void main(String args[])
   {
      Student s=new Student();
      s.display();    // accessing methods using object
   }
}
```

Output:

```
My name is abc
I am 20 years old
```

Mutator and Accessor Methods:

- These are the two basic terminologies used with Java classes.
  1. **Mutator** method allows the change in value of class variables. It means that from outside class we can manipulate the data of a class.
  2. **Accessor** method just allows to access the data. It does not allow to manipulate the data.
- In the same program, if we are using both i.e. mutator and accessor methods together, then it is suggested that it is not a good programming. For writing a good programs, we should use any one method.

For example:

```
class a
{
   int p;
   int get()            // accessor method
   { return p; }
   void setp(int q)     //mutator method
   { p = q; }
}
```

Program 2.4: Program shows the usage of mutator method.

```
class area
{
   float length;
   float breadth;
   void setd (float x, float y)
   {
      length = x;
      breadth = y;
   }
}
class mainarea
{
   public static void main (String args[])
   {
      float a;
      area ob1=new area();
      ob1.setd(8, 10);
      a = ob1.length * ob1.breadth;
      System.out.println ("Area = " +a);
   }
}
```

Output:

```
Area = 80
```

2.2 DEFINING YOUR OWN CLASSES

- To use Java effectively, you want to create and use your own classes. This is one of the great powers of Object-Oriented Languages, the ability to construct programs out of independent building-blocks that cut large problems down into small, easily solvable ones.
- We are going to create a very simple object class that will print a "Hello" message to the screen when called.
- This class will have two different methods that will do the same thing, though we will vary the messages they print a bit so that we can see which one is doing the work.

```
public class HelloClass
{
    /** sayHello() prints "Hello!" to the Java console output
    */
    public void sayHello()
    {
        System.out.println("Hello!\n");
    }
    /**
    * doHello() prints "Hello, hello!" to the Java console output
    * It's static, so you don't need to instatiate a HelloClass
    * object to use it
    */
    public static void doHello()
    {
        System.out.println("Hello, hello!\n");
    }
} // End of HelloClass
```

- The two methods we have to print messages are sayHello() and doHello(). To use sayHello(), we need to have a HelloClass object created, then call that object's sayHello()method.
- doHello(), however, is a static method. This means is belongs to the class, not to any object of the class. So we can use it without any HelloClass objects being created first.

2.3 ACCESS SPECIFIERS

- In Java, it may be necessary in some situations to restrict the access to certain variables and methods from outside the class.
- For example, we may not like the objects of a class directly alter the value of a variable or access a method. We can achieve this in Java by applying visibility modifiers.

- The visibility modifiers are also known as access modifiers. Java provides a number of access modifiers to set access (scope) levels for classes, variables, methods and constructors. The four access levels are:
  1. Visible to the package (default/friendly),
  2. Visible to the class only (private),
  3. Visible to the world (public), and
  4. Visible to the package and all subclasses (protected).
- Let us see above access specifiers in detail.

1. Default / Friendly Access Modifier (No Keyword):

- If we give no specifier at all, the access is referred as friendly. It means that all the other classes in the current package have access to the friendly member but to all the classes outside the package the members appear to be private.
- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
- A variable or method declared without any access control modifier is available to any other class in the same package.
- The fields in an interface are implicitly public static final and the methods in an interface are by default public.
- Variables and methods can be declared without any modifiers, as in the following examples:

```
String version = "1.5.1";
Boolean processOrder()
{
    return true;
}
```

2. Private Access Modifier (private):

- Methods, variables and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the highest degree restrictive access level. Class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
- The following class uses private access control:

```
public class PLog
{
    private String format;
    public String getFormat()
    {
        return this.format;
    }
```

```
        public void setFormat(String format)
        {
            this.format = format;
        }
    }
```

- Here, the format variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.
- So to make this variable available to the outside world, we defined two public methods i.e., getFormat(), which returns the value of format, and setFormat(String), which sets its value.

Private Protected Access Modifier:

- A field can be declared with two keywords private and protected together such as,
  `private protected int RollNumber;`
- This modifier gives a visibility level in between the "protected" access and "private" access.
- This modifier makes the fields visible in the subclasses regardless of what package they are in. Remember that these fields are not accessible by other classes in the same package.

3. Public Access Modifier (public):

- A class, method, constructor, interface etc. declared public has the widest possible visibility and accessible everywhere.
- Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
- However, if the public class we are trying to access is in a different package, then the public class still needs to be imported.
- Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
- The following function uses public access control:

```
public static void main(String[] arguments)
{
    // ...
}
```

- The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

4. Protected Access Modifier (protected):

- In Java the visibility level of a "protected" field lies in between the public access and friendly access.
- Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
- The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
- The following parent class uses protected access control, to allow its child class override openSpeaker()method:

```
class AudioPlayer
{
    protected boolean openSpeaker(Speaker sp)
    {
        // implementation details …
    }
}
class StreamingAudioPlayer
{
    boolean openSpeaker(Speaker sp)
    {
        // implementation details
    }
}
```

- In above program code, if we define openSpeaker() method as private, then it would not be accessible from any other class other than AudioPlayer. If we define it as public, then it would become accessible to all the outside world. But our intension is to expose this method to its subclass only, that's why we used protected modifier.
- Following table summarizes visibility of field in a class of Java:

Access Location	Public Modifier	Protected Modifier	Friendly (Default) Modifier	Private Protected Modifier	Private Modifier
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	**Yes**	No
Non-subclasses in other packages	Yes	No	No	No	No

Program 2.5: Program for access modifier.

```
class Rectangle
{
   public int length;          //public Access
   public int breadth;         //public Access
   static int rectCount = 0;   //default Access
   //Constructor to initialize length and breadth
   Rectangle()
   {
      rectCount++;
   }                           //method to calculate area of rectangle
      public int area()
      {
         int rectArea;
         rectArea = length * breadth;
         return rectArea;
      }
   }
   class AccessSpecifer
   {
      public static void main(String[] args)
      {
         //create first rectangle object
         Rectangle firstRect = new Rectangle();
         //accessing public members outside that Rectangle class
         firstRect.length = 10;
         firstRect.breadth = -30;
         System.out.println("Area of Rectangle is: "+firstRect.area());
         //Rectangle.rectCount = 5;
         //accessing member with default access
         System.out.println("Number of Object Created:
                                                  "+Rectangle.rectCount);
      }
   }
```

Output:

```
Area of Rectangle is: – 300
Number of Object Created: 1
```

2.4 ARRAY OF OBJECTS

- In java, rather than creating object of a particular or class, we can also create array of objects.
- In Java, an array of objects can also be created to store and process the objects. The process to declare and create an array of objects is similar to the process of declaring and creating an array of simple data types.
- It is possible to create array of objects of user created/defined class. When we create an array of objects then the data type gets replaced with the class name.
- The following is the syntax to declare and create an array of objects:

```
class_name: arr_name [ ] = new class_name [size];
```

  Where,

  class_name is the name of the class whose objects are used to create an array of objects.

  arr_name is the name of the array, which is to be created.

  new is the Java operator, which is used to allocate memory to an array.

  size is the number of objects that an array can hold.
- Let us now suppose that we have a class, ArrayDemo. We have to create an array of objects of this class. To create an array of objects to ArrayDemo class see the following example:

```
ArrayDemo a_obj[ ] = new ArrayDemo [2]:
```

- The a_obj array can hold 2 objects of ArrayDemo class. To use the a_obj array, we have to create objects of every array index.
- See the following code snippet to create objects of every index of a_obj array:

```
a_obj [0] = new ArrayDemo();
a_obj [1] = new ArrayDemo();
```

Fig. 2.3: Array of Objects

Program 2.6: Program for array of objects.

```
class Employee
{
   int id;
   String name;
   public void setData(int id, String name)
   {
      this.id = id;
      this.name = name;
   }
   public void displayData()
   {
      System.out.println("Employee ID: "+this.id);
      System.out.println("Employee Name: "+this.name);
   }
}
class Main
{
   public static void main(String args[])
   {
      Employee[] emp = new Employee[2];
      emp[0].setData(1,"Amar");
      emp[1].setData(2,"Akbar");
      emp[0].displayData();
      emp[1].displayData();
   }
}
```

Output:

```
Employee ID: 1
Employee Name: Amar
Employee ID: 2
Employee Name: Akbar
```

2.5 CONSTRUCTORS

- A constructor is a special method of a class in Java programming that initializes an object of that type.
- Constructors have the same name as the class itself. A constructor is automatically called when an object is created.

- Constructor constructs the values i.e. provides data for the object that is why it is known as constructor.

  Syntax:

```
class_Name
{
   constructor_Name (same as class_Name); // Constructor
   {
      //constructor body…..
   }// end of Constructor
}// end of Class
```

- There are basically following rules defined for the constructor:
  1. Constructor name must be same as its class name.
  2. Constructor must have no explicit return type.
  3. Constructors may be private, protected or public.
  4. Multiple constructors may exist, but they must have different signatures, i.e., different numbers and/or types of input parameters.

Program 2.7: Program for constructors.

```
class Rectangle
   {
   int length;
   int breadth;
   //constructor to initialize length and breadth of rectangle
   Rectangle()
   {
      length = 5;
      breadth= 6;
   }
   //method to calculate area of rectangle
   int area()
   {
      int rectArea = length * breadth;
      return rectArea ;
   }
   }
   //class to create rectangle objects and calculates area
   class ConstructorExample
   {
      public static void main(String[] args)
      {
         Rectangle firstRect = new Rectangle();
         System.out.println("Area of Rectantangle: "+ firstRect.area());
      }
   }
```

Output:

```
Area of rectangle: 30
```

- Constructors can be classified into two types, default constructors and parametarized constructors.

1. Default Constructor:

- The constructor which does not accept any argument is called default constructor. The argument list is void.
- In other word, when the object is created Java creates a no-argument constructor automatically known as default constructor.
- It does not contain any parameters nor does it contain any statements in its body. Its only purpose is to enable you to create object of class type.
- Default constructor provides the default values to the object like 0, null etc. depending on the type.

Syntax:
```
<class_name>()
{
// Body of the constructor…
}
```

Program 2.8: Program for default constructor.

```
class StudentInfo
{
   int rollno;
   int marks1;
   int marks2;
   int total;
      StudentInfo()   //no-arg default constructor
      {
         rollno=2;
         marks1=30;
         marks2=40;
      }
      //method to calculate total
      int Total()
      {
         total = marks1 + marks2;
         return total;
      }
      //method to display the result
      void displayResult()
      {
         System.out.println("Roll no of student is" + rollno );
         System.out.println("marks1 are" + marks1 );
         System.out.println("marks2 are" + marks2);
         System.out.println("total is" + total);
      }
   }
```

```
class StudentResult
{
public static void main(String args[])
{
   int total1,total2,grandtotal;
   StudentInfo student1 =new StudentInfo();  //creates first object
   StudentInfo student2 =new StudentInfo();  //creates second object
   total1=student1.Total();
   total2=student2.Total();
   grandtotal=total1+total2;
   student1.displayResult();
   student2.displayResult();
   System.out.println("Grand Total is" + grandtotal);
   }
}
```

Output:

```
Roll no of student is 2
marks1 are 30
marks2 are 40
total is 70
Roll no of student is 2
marks1 are 30
marks2 are 40
total is 70
Grand Total is 140
```

2. Parameterized Constructor:

- A constructor that has parameters is known as parameterized constructor. Parameterized constructor is used to provide different values to the distinct objects.
- Using parameterized constructor, it is possible to initialize objects with different set of values at the time of their creation. These different set of values initialized to objects must be passed as arguments when constructor is invoked.
- The parameter list can be specified in the parentheses in the same way as parameter list is specified in the method.
- The **syntax** for constructor is as follows:

```
ConstructorName([parameterList])
{
// Constructor body…
}
```

- Here, the ConstructorName is same as the class name it belongs to. The parameterList is the list of optional zero or more parameter(s) that is specified after the classname in parentheses. Each parameter specification, if any, consists of a type and a name and is separated from each other by commas.

Program 2.9: Program for parameterized constructor.

```
class StudentInfo
{
   int rollno;
   int marks1;
   int marks2;
   int total;
StudentInfo(int roll_no,int m1,int m2)  //Parameterized constructor
{
   rollno=roll_no;
   marks1=m1;
   marks2=m2;
}
   //method to calculate total
   int Total()  //Method declaration
   {
    total = marks1 + marks2;
    return total;
   }
   //method to display the result
   void displayResult()
   {
      System.out.println("Roll no of student is" + rollno );
      System.out.println("marks1 are" + marks1 );
      System.out.println("marks2 are" + marks2);
      System.out.println("total is" + total);
   }
}
class StudentResult
{
 public static void main(String args[])
 {    int total1,total2,grandtotal;
      StudentInfo student1 = new StudentInfo(1,50,80);
      StudentInfo student2 = new StudentInfo(2,40,60);
      total1=student1.Total();
      total2=student2.Total();
      grandtotal=total1+total2;
      student1.displayResult();
      student2.displayResult();
      System.out.println("Grand Total is" + grandtotal);
   }
}
```

Output:

```
Roll no of student is 1
marks1 are 50
marks2 are 80
total is 130
Roll no of student is 2
marks1 are 40
marks2 are 60
total is 100
Grand Total is 230
```

2.6 CONSTRUCTOR OVERLOADING [Oct. 16]

- Constructors can be overloaded. Constructor having the same name with different parameter list is called as constructor overloading.

Program 2.10: Program for constructor overloading.

```
class Student
{
   int id;
   String name;
   int age;
   Student(int i,String n)
   {
      id = i;
      name = n;
   }
   Student(int i,String n,int a)
   {
      id = i;
      name = n;
      age=a;
   }
   void display(){System.out.println(id+" "+name+" "+age);
}
public static void main(String args[])
   {
      Student s1 = new Student(1,"Amar", 22);
      Student s2 = new Student(2,"Amol",20);
      s1.display();
      s2.display();
   }
}
```

Output:

```
1 Amar 22
2 Amol 20
```

2.7 USE OF 'this' KEYWORD [April 16, 17, 18]

- Many times it is necessary to refer to its own object in a method or a constructor. To allow this Java defines the 'this' keyword.
- As we know in C++, sometimes a method will need to refer to the object that invoked it. To do so, Java also has a 'this' keyword.
- The 'this' can be used inside any method to refer to the current object. It means that 'this' is always a reference to the object on which the method was invoked.
- The 'this' is used inside the method or constructor to refer its own object i.e., 'this' is always a reference to the object of the current class' type.

Usage of this Keyword:

1. 'this' keyword can be used to refer current class instance variable.
2. 'this' keyword can be used to invoke current class constructor.
3. 'this' keyword can be used to invoke current class method (implicitly).
4. 'this' can be passed as an argument in the method call.

Syntax:

```
this.field
```

- Fig. 2.4 shows use of 'this' keyword.

Fig. 2.4: this Keyword

Example:

```
class BoxDim
{
   int height;
   int depth;
   int length;
   BoxDim(int height, int depth, int length)
   {
      this.height = height;
      this.depth = depth;
      this.length = length;
   }
}
```

- The class 'BoxDim' contains three instance variables i.e. height, depth and length. The constructor of the class also contains three different local variables with the same names of instance variables.

- Compiler will not show any error here. Then how differentiate among these variables? The 'this' keyword does this job. 'this' will refer to the variables of its own class. It is acting as the object of the current class.
- In method also the 'this' keyword is used to differentiate between instance and local variables. This concept is also referred as 'Instance variable hiding'. This resolves the name-space collisions.

Program 2.11: Program for 'this' keyword.

```
class Rectangle
{
   int length,breadth;
   void show(int length,int breadth)
   //Formal and instance variable name are same
   {
      this.length=length;    //Use of this keyword
      this.breadth=breadth;
   }
   int calculate()
   {
   return(length*breadth);
   }
}
/* Main class */
public class UseOfThisOperator
{
   public static void main(String[] args)
   {
      Rectangle rectangle=new Rectangle();
      rectangle.show(8,6);
      int area = rectangle.calculate();
      System.out.println("The area of a Rectangle is: " + area);
   }
}
```

Output:

```
The area of a Rectangle is: 48
```

2.8 STATIC BLOCK, STATIC FIELDS AND METHODS [Oct. 16]

- In Java basically, class contains variables called instance variables and methods called instance method.
- When we create an object the memory gets allocated for the variables and methods also. If we create one more object, same thing happens again.

2.8.1 Static Block

- A class can contain code in a static block that does not exist within a method body. Static code block executes only once when the class is loaded.
- A static block is used to initialize static attributes. Static blocks are also called static initialization blocks.
- A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword.
- Here, is an example:

```
static
{
    //whatever code is needed for initialization goes here, …
}
```

- A class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code. And don't forget, this code will be executed when JVM loads the class.
- JVM combines all these blocks into one single static block and then executes. Here, are a couple of points to mention:
  1. If you have executable statements in the static block, JVM will automatically execute these statements when the class is loaded into JVM.
  2. If we are referring some static variables/methods from the static blocks, these statements will be executed after the class is loaded into JVM same as above i.e., now the static variables/methods referred and the static block both will be executed.

Program 2.12: Program for static block.

```
public class StaticExample
{
static
    {
        System.out.println("This is first static block");
     }
     public StaticExample()
    {
        System.out.println("This is constructor");
     }
     public static String staticString = "Static Variable";
     static {
         System.out.println("This is second static block and" + staticString);
     }
```

```
    public static void main(String[] args){
        StaticExample statEx = new StaticExample();
        StaticExample.staticMethod2();
    }
    static {
        staticMethod();
        System.out.println("This is third static block");
    }
    public static void staticMethod() {
        System.out.println("This is static method");
    }
    public static void staticMethod2() {
        System.out.println("This is static method2");
    }
}
```

Output:

```
This is first static block
This is second static block and Static Variable
This is static method
This is third static block
This is constructor
This is static method2
```

- Now lets the output of program. First all static blocks are positioned in the code and they are executed when the class is loaded into JVM.
- Since, the static method staticMethod() is called inside third static block, its executed before calling the main method.
- But the staticMethod2() static method is executed after the class is instantiated because it is being called after the instantiation of the class.

2.8.2 Static Field

- In certain situations, it may be required to share a common copy of fields among all the objects of the same class. This is accomplished by declaring the field(s) to be static and such fields are known as static field(s).
- If a field is declared static then there is one field for the entire class instead of one per object.

- A static field of a class is often referred to as a class variable because static field is associated with a class and not with individual instances of the class.
- A static field gets memory only once for the whole class no matter how many objects of a class are created.
- To declare a static field, prefix the field declaration in the class with the static modifier. It's **syntax** is,

```
static datatype fieldName;
```

Properties/Characteristics of Static Field:

1. There is only one copy of static field in a class which is shared among all the objects of the class.
2. A static field can be accessed before any object of a class is created, without reference to any object.
3. To make a static field constant in Java, make a variable as both static and final.

- The static field is similar to static variable. The main difference is that a static variable is used to retain information between calls whereas static field is used to share information among multiple objects of a class.

Program 2.13: Program shows use of static fields.

```
class Rectangle
{
   int length; //length of rectangle
   int breadth; //breadth of rectangle
   static int rectCount =0; //count rectangle objects
   void setData(int l,int b)
   {
      length=l;
      breadth=b;
      rectCount++;
   }
 //method to calculate area of rectangle
   int area()
   {
      int rectArea;
      rectArea = length * breadth;
      return rectArea;
   }
}
```

```
//class to create Rectangle objects and access static field
class StaticField
{
   public static void main(String[] args)
{
  //create first rectangle object
  Rectangle firstRect =new Rectangle();
  firstRect.setData(5,8);
  System.out.println("Area of Rectangle 1: "+firstRect.area());
  //create second rectangle object
  Rectangle secondRect =new Rectangle();
  secondRect.setData(10,15);
  System.out.println("Area of Rectangle 2: "+secondRect.area());
  // access static field of rectangle class
  System.out.println("Total Number of Objects: "+Rectangle.rectCount);
  }
}
```

Output:

```
Area of Rectangle 1: 40
Area of Rectangle 2: 150
Total Number of Objects: 2
```

- In the Program 2.13, the class Rectangle has two instance variables length and breadth and one class variable rectCount.
- When we create firstRect and secondRect objects of type Rectangle, each has its own copy of length and breadth instance variables but both share a single copy of class variable rectCount.
- The class variable rectCount is initialized to 0 only when the class is first loaded, not each time a new object is made. When the object firstRect is created and the setData() method is invoked, the static variable rectCount is incremented by 1 and it is set to 1 (= 0+ l).
- Similarly, when the object secondRect is created and the setData () method is invoked the value of rectCount variable is set to 2 (= 1+l). Finally, the statement,

  `System.out.println("Total Number of Objects: "+ Rectangle.rectCount);`

  prints the total number of objects.

2.8.3 Static Methods [Oct. 16]

- It is possible to have static methods in a class in the same way as we have static fields. If you apply static keyword with any method, it is known as static method.
- A static method belongs to the class rather than object of a class. A static method can be invoked without the need for creating an instance of a class.
- Static method can access static data member and can change the value of it. The static method is similar to instance method of a class but the only difference is that the static method can be called through its class name without creating any object of that class.
- A static method is also called class method as it is associated with a class and not with individual instance of the class.
- A static method cannot access non-static method either because non-static methods are usually associated with instance variable state.
- To call a particular static method, the **syntax** is given below:

```
classname.staticmethodname (arg-list);
```

Program 2.14: Program using static method which maintains bank account information about various customers.

```
class Account
{
   int accountNo; //account ID)
   double balance;
   static double rate = 0.05;
   void setData(int n,double bai)
   {
      accountNo = n;
      balance = bai;
   }
   void quarterRatecal()
   {
      double interest = balance * rate * 0.25;
      balance += interest;
   }
   static void modifyRate(double incr)
   {
      rate += incr;
      System.out.println("Modified Rate of Interest: " + rate);
   }
```

```
    void show()
    {
        System.out.println("Account Number: " + accountNo);
        System.out.println("Rate of Interest: " +rate)
        System.out.println("Balance: "+ balance);
    }
}
public class StaticMethod
{
    public static void main(String[] args)
    {
        Account acc1 = new Account();
        Account acc2 = new Account();
        Account.modifyRate(0.01);
        System.out.println("Customerl Information......");
        acc1.setData(201,1000);
        acc1.quarterRatecal(); //Calculate interest
        acc1.show(); //display account interest
        System.out.println("Customer2 Information......");
        acc1.setData(202,1000);
        acc2.quarterRatecal(); //Calcuate interest
        acc2.show(); //display account information
    }
}
```

Output:

```
Modified Rate of Interest: 0.060000000000000005
Customer1 Information …
Account Number: 201
Rate of Interest: 0.06
Balance: 1000.0
Customer2 Information …
Account Number: 202
Rate of Interest: 0.06
Balance of 1000.0
```

- The Account class in Program 2.14 contains a static method modifyRate ().It modifies the rate of interest by the value specified as argument (0.01).
- The method modifyRate() is made static in the class as we are only using static field rate in it. Only static method can access a static field.

2.9 PREDEFINED CLASSES

- There are many predefined classes in Java. The object class is a root of all classes in the Java technology programming language.
- If a class is declared with no extends clause, then the compiler implicitly adds "extends object" to the declaration.

2.9.1 Object Class [April 18]

- The object class is the parent class of all the classes in Java by default. In other words, it is the topmost class of java.
- The java.lang.Object class is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.
- Following is the declaration for java.lang.Object class:

```
public class Object
```

- The Object class provides some common behaviours to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Fig. 2.5: Hierarchy of class

- We already known object class is the superclass of all classes. Some methods of the object class are: [April 18]
  - `Boolean equals(Object obj)`: Indicates whether some other object is "equal to" this one.
  - `protected Object clone()`: Creates and returns a copy of this object.
  - `Boolean equals(Object obj)`: Indicates whether some other object is "equal to" this one.
  - `protected void finalize()`: This method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
  - `int hashCode()`: Returns a hash code value for the object.
  - `void notify()`: Wakes up a single thread that is waiting on this object's monitor.

- `void notifyAll()`:Wakes up all threads that are waiting on this object's monitor.
- `String toString()`: Returns a string representation of the object.
- `getClass()`: Returns the runtime class of this object. The class object which is returned is the object that is locked by static synchronized method of the represented class.
- `void wait()`: Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

Program 2.15: Program for object class.

```
class Rectangle extends Object
{
   private double length,breadth;
   Rectangle(double x,double y)
   {
   length = x ;
   breadth = y ;
   }
   public void area()
   {
      System.out.println("Area of Rectangle is = " + breadth);
   }
   public void circumference()
   {
      System.out.println("Circumferencedfcdf of Rectangle is=
                                                    "+2*(length+breadth));
   }
}
class ObjectClass
{
   public static void main(String[] args)
   {
      Rectangle r = new Rectangle(10,20);
      Rectangle rl = new Rectangle(10,20);
      System.out.println("String Representation = " + r.toString());
      System.out.println("Class Name = " + r.getClass());
      System.out.println("Hash Code = " + r.hashCode());
      System.out.println("r.rquals(rl) = " + r.equals(rl));
   }
}
```

2.9.2 String Class [April 17, 18, 19]

- The strings in Java are treated as objects of type 'String' class. This class is present in the package java.lang.
- This package contains two string classes:
  1. String class, and
  2. StringBuffer class.
- The string class is used when we work with the string which cannot change whereas StringBuffer class is used when we want to manipulate the contents of the string.
- When we create object of String Class they are designed to be immutable. As these objects are immutable, we can alias to a particular string for many time which we want. Even if we alias the string class objects, they won't affect each other because they are read only.
- We can use + operator to overload for string objects. Only two operators i.e. '+' & '+=" are overloaded for string classes. In Java, when we use + operator it allows us to concatenate two or more number of strings together.

  Example: `String str = "Kal" + "pa" + "na";`

  Output: `Kalpana`
- In the above example a method append() is called which creates the new String object having "Kal" and concatenate with the next contents.
- It generally calls toString() method which allows you to determine the String representation for objects and classes which are created.

String class Methods:

- The String class provides various methods that appear to modify strings.
- The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Sr. No.	Constructor	Description
1.	`char charAt(int index)`	This method returns char at (value for) the particular index.
2.	`int length()`	This method returns string length.
3.	`static String format(String format, Object... args)`	This method returns formatted string.
4.	`static String format(Locale l, String format, Object... args)`	This method returns formatted string with given locale.
5.	`String substring(int beginIndex)`	This method returns substring for given begin index.
6.	`String substring(int beginIndex, int endIndex)`	This method returns substring for given begin index and end index.
7.	`boolean contains(CharSequence s)`	This method returns true or false after matching the sequence of char value.

contd. ...

8.	`static String join(CharSequence delimiter, CharSequence... elements)`	This method returns a joined string with delimiter.
9.	`static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)`	This method returns a joined string.
10.	`boolean equals(Object another)`	This method checks the equality of string with object.
11.	`boolean isEmpty()`	This method checks if string is empty.
12.	`String concat(String str)`	This method concatenates specified string.
13.	`String replace(char old, char new)`	This method replaces all occurrences of specified char value.
14.	`String replace(CharSequence old, CharSequence new)`	This method replaces all occurrences of specified CharSequence.
15.	`String trim()`	This method returns trimmed string omitted leading and trailing spaces.
16.	`String split(String regex)`	This method returns splitted string matching regex.
17.	`String split(String regex, int limit)`	This method returns splitted string matching regex and limit.
18.	`String intern()`	This method returns interned string.
19.	`int indexOf(int ch)`	This method returns specified char value index.
20.	`int indexOf(int ch, int fromIndex)`	This method returns specified char value index starting with given index.
21.	`int indexOf(String substring)`	This method returns specified substring index.
22.	`int indexOf(String substring, int fromIndex)`	This method returns specified substring index starting with given index.
23	`String toLowerCase()`	This method returns string in lowercase.
24.	`String toLowerCase(Locale l)`	This method returns string in lowercase using specified locale.
25.	`String toUpperCase()`	This method returns string in uppercase.
26.	`String toUpperCase(Locale l)`	This method returns string in uppercase using specified locale.

Constructors of String:

- Constructors are used to initialize the object. The string class supports several constructors.
- The following table shows the constructors of the string class.

Sr. No.	Constructor	Description
1.	`String()`	This constructor creates an empty string.
2.	`String(String value)`	This constructor creates a new string that is a copy of the given string.
3.	`String(char[] value)`	This constructor constructs a new string based on the character array.
4.	`String(char[] value, int begin, int count)`	This constructs a new string based on the character array starting from the position begin which is 'count' characters long.
5.	`String(byte[] value)`	This constructor creates a new string by converting the given array of bytes.
6.	`String(byte[] value, int offset, int length)`	This constructor creates a new string by converting the given sub of array of bytes.
7.	`String(StringBuffer buffer)`	This constructor creates a new string based on a StringBuffer value.
8.	`String(char[] value, int begin, int count, String enc) throws UnsupportedEncoding Exception`	This constructor creates a new string based on the given byte array and uses given character encoding that is denoted by enc.
9.	`String(char[] value, String enc) throws UnsupportedEncoding Exception`	This constructor creates a new string based on the given byte array and uses given character encoding that is denoted by enc.

2.9.3 StringBuffer Class [April 17, 18]

- It is a peer class which provides the functionality of strings. The string generally represents fixed length, immutable character sequence whereas StringBuffer represents growable and writeable character sequences.

- StringBuffer may have some characters and if needed substring can be inserted in the middle or appended at the end. StringBuffer automatically provides a room to grow such additions.
- Java generally manipulate the strings using + as overloaded operator. StringBuffer class in Java is used to created mutable (modifiable) string.
- The StringBuffer class in Java is same as String class except it is mutable i.e., it can be changed.
- StringBuffer class is a mutable class unlike the String class which is immutable. Both the capacity and character string of a StringBuffer Class. StringBuffer can be changed dynamically.

Advantages of StringBuffer Class:

1. Alternative to String class.
2. Can be used wherever a string is used.
3. More flexible than String.

- The StringBuffer defines three types of constructors which we use to initialize the string value.
  1. `StringBuffer()`
  2. `StringBuffer(int size)`
  3. `StringBuffer(String str)`
- The first type is the default constructor as it has no parameters. It reserves room for 16 characters without reallocation. The second, type accepts an integer argument which explicitly sets the size of buffer. In third case, the string argument sets the initial contents of the StringBuffer object and reverse room for 16 more characters. These 16 characters room is allocated when no specific buffer length is requested. This is because reallocation is a costly process in terms of time.
- The current length of a StringBuffer can be found via length() method and the total allocated capacity can be found through capacity() method.
- The general forms are:

  `int length();`

  and

  `int capacity();`
- StringBuffer creates string objects that can be changed. StringBuffer provides different methods which are given below:

  `charAt()` method obtains the character at a specific location.

  `setCharAt()` method sets a character within the string.
- String object is manufactured which contains the substring. In this the original string is unaltered and immutability remains intact.
- String provides a substring() method which returns a new string. It contains a specified portion of the invoking string. This method is substring().
- The **syntax** is given below:

  `String subString(int start_Index, int endIndex)`
- The startIndex specifies the beginning index and endIndex tells us about stopping point.

StringBuffer Class Methods: Here, is the list of important methods supported by StringBuffer class:

1. `public StringBuffer append(String s)`: Updates the value of the object that invoked the method.
2. `public StringBuffer reverse()`: Reverses the value of the StringBuffer object that invoked the method.
3. `public delete(int start, int end)`: Deletes the string starting from start index until end index.
4. `public insert(int offset, int i): Inserts an string s at the position mentioned by offset.`
5. `replace(int start, int end, String str)`: Replaces the characters in a substring of this StringBuffer with characters in the specified String.

- Here, is the list of other methods (Except set methods) which are very similar to String class:
  1. `int capacity()`: Returns the current capacity of the String buffer.
  2. `char charAt(int index)`: The specified character of the sequence currently represented by the string buffer, as indicated by the index argument, is returned.
  3. `void ensureCapacity(int minimumCapacity)`: Ensures that the capacity of the buffer is at least equal to the specified minimum.
  4. `void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`: Characters are copied from this string buffer into the destination character array dst.
  5. `int indexOf(String str)`: Returns the index within this string of the first occurrence of the specified substring.
  6. `int indexOf(String str, int fromIndex)`: Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
  7. `int lastIndexOf(String str)`: Returns the index within this string of the rightmost occurrence of the specified substring.
  8. `int lastIndexOf(String str, int fromIndex)`: Returns the index within this string of the last occurrence of the specified substring.
  9. `int length()`: Returns the length (character count) of this string buffer.
  10. `void setCharAt(int index, char ch)`: The character at the specified index of this string buffer is set to ch.
  11. `void setLength(int newLength)`: Sets the length of this String buffer.
  12. `CharSequence subSequence(int start, int end)`: Returns a new character sequence that is a subsequence of this sequence.
  13. `String substring(int start)`: Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.The substring begins at the specified index and extends to the end of the StringBuffer.
  14. `String substring(int start, int end)`: Returns a new String that contains a subsequence of characters currently contained in this StringBuffer.
  15. `String toString()`: Converts to a string representing the data in this string buffer.

Program 2.16: Program to show use of substring.

```
class strTry
{
   public static void main(String args[])
   {
      String st1 = "Java is platform independent";
      // to construct a substring we write
      String sub = st1.substring(9, 16);
      // lets print these values
      System.out.println("Original string is: + st1");
      System.out.println("Substring is: + sub");
   } //end main
} //end class
```

Output:

```
Original string is: Java is platform independent
Substring is: platform
```

Difference between String and StringBuffer: **[April 17, 18]**

- String objects are constants and immutable whereas StringBuffer objects are not.
- StringBuffer Class supports growable and modifiable string whereas String class supports constant strings.
- Strings once created we cannot modify them. Any such attempt will lead to the creation of new strings. Whereas StingBuffer objects after creation also can be able to delete or append any characteres to it.
- String values are resolved at run time whereas StringBuffer values are resolved at compile time.

2.9.4 Formatting String

- Formatting strings in Java has always been a problem for developers because the basic System.out.println() method does not support formatting of Strings while displaying the result on screen.
- Formatted strings not only display the String content but also it displays the content in the specified sequence.
- For instance, when displaying large numbers like 100, 0000, 0000, it also displays the comma long sent while displaying decimal numbers, it prints numbers using proper decimal places like “134.43” as required.
- The need for formatted String is generally required in the modern GUI application. Java also has good formatting support for other types like Double, Integers and Date.
- Java consists of two methods of formatting the Strings:
  1. format() method in java.lang.String class.
  2. printf() method in PrintStream class.

- Let us see above methods in detail.

1. format() Method:

- The format method is a static method. It takes three arguments as input and returns a formatted String.

  Syntax:

```
public static String format(String format, Object… args)
public static String format(Locale l, String format, Object… args)
```

  where,

  - "Local l" is the locale applied during formatting. However, if it is null the localization isn't applied.
  - "format" contains the string you want formatted.
  - "args" is the parameter referenced by format specifiers in the format String. If the arguments are more than the format specifiers, the extra arguments are ignored. Arguments can vary and may be zero.

Exceptions Thrown by the Method:

1. **IllegalFormatException:** The exception is thrown due to illegal syntax or if an incompatible format specifier with respect to given arguments is given.
2. **NullPointerException:** This exception is thrown when the String argument passed is null.

Simple Example of format():

- "Local l" value is specified as "France" and the comma as written in the France number system will replace the decimal point.

Program 2.17: Program for format() method.

```
class Demo
{
    public static void main(String[] args)
    {
    System.out.format(Locale.FRANCE,
                "The value of the float " + "variable is %f  ",10.3242342);
    }
}
```

Output:

```
The value of the float variable is 10,324234.
```

Formatting Instructions:

- The basic **syntax** is:

```
string.format "%[argument number] [flags] [width] [.precision] type"
```

Where,

- % is special character denotes the start of formatting instructions. String.format() method supports multiple occurrence of "%" character in a single formatting String.
- "argument number" specifies the corresponding argument in formatting.
- "flags" are the special instructions used to print String specific formats such as flags as ";" and to print a semicolon in the output.
- "width" gives us the option to output the minimum number of characters. If the number is larger than the width, the full number is displayed. Otherwise, it's padded with zero.
- "precision" is used for printing the float point format String. It helps to specify the number of decimal values the floating number should display.
- "type" is a mandatory option. It's always specified last and specifies the type of the parameter being formatted. For instance, "%d" is written for integer values.

Simple rules of String Formatting:

- **%s:** Will print the string with no change.
- **%15s:** Prints fifteen characters. If the character count is less than 15, the output is padded on the left.
- **%-6s:** Prints 6 characters as-is. If the string is less than 6 characters, the output is padded on the right.
- **%.8s:** Prints a maximum eight characters.

Program 2.18: Program for string formatting.

```
class Test
{
   public static void main (String[] args) throws.lang.Exception
   {
      //String is passed in the format function
      String str = String.format("Hello %s", "Jack.Where is Jill?");
      System.out.println(str);
      String str1 = String.format("Hello %15s", "Jack.Where is Jill?");
      System.out.println(str1);
      String str2 = String.format("Hello %.6s", "Jack.Where is Jill?");
      System.out.println(str2);
   }
}
```

Output:

```
Hello Jack.Where is Jill?
Hello Jack.Where is Jill?
Hello Jack.W
```

- The String is passed in format function which returns a formatted String. Note that the format function does not print the value on the screen.

2. printf() Method: **[April 18]**

- This method is similar to printf used in C programming. It's used similar to a replacement of System.out.println().
- Unlike the format() method, the printf() method prints String on screen.

Syntax:

```
public PrintStream format(String format, Object… args)
```

Where, "format" is the format String which contains the format specifiers specifying how the argument will be formatted and "args" is the list of variables to be printed. "Object… args" notion is called varargs which means that the arguments may vary.

Program 2.19: Program for printf().

```
class StringFormat
{
   public static void main(String[] args)
   {
      System.out.printf("Integer: %d\n",15);
      System.out.printf("Floating number with 4 decimal digits: %.4f\n",1.123123123123);
      System.out.printf("Floating number with 6 decimal digits: %.6f\n",1.123123123123);
      System.out.printf("String: %s, integer: %d, float: %.6f", "Hello World",55,9.1234567);
   }
}
```

Output:

```
Integer: 15
Floating number with 4 decimal digits: 1.1231
Floating number with 6 decimal digits: 1.123123
String: Hello World, integer: 55, float: 9.123457
```

- The Strings, integer and floating values are directly passed to the printf() and the formatted output is displayed on the screen.

2.10 WRAPPER CLASSES [April 17, Oct. 17]

- The primitive data types are not objects; they do not belong to any class; they are defined in the language itself. Sometimes, it is required to convert data types into objects in Java language.

- A data type is to be converted into an object and then added to a Stack or Vector etc. For this conversion, the designers introduced wrapper classes.
- As the name says, a wrapper class wraps (encloses) around a data type and gives it an object appearance.
- Wherever, the data type is required as an object, this object can be used. Wrapper classes include methods to unwrap the object and give back the data type.

List of Wrapper Classes:

- In the above code, Integer class is known as a wrapper class (because it wraps around int data type to give it an impression of object). To wrap (or to convert) each primitive data type, there comes a wrapper class.
- Eight wrapper classes exist in java.lang package that represent eight data types as given in following table:.

Data Types	Wrapper Classes
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- There are mainly four uses of wrapper classes.

1. **Converting primitive numbers to objects:**

```
integer iobj=new integer(i) //conversion of primitive int(i) to integer
                            object (iobj)
float fobj=new float(f)     //conversion of primitive float(f) to float
                            object (fobj)
double dobj=new double(d)   //conversion of primitive double(d) to double
                            object (dobj)
```

2. **Converting objects to primitive numbers:**

```
int i=iobj.intValue()       //conversion of integer object (iobj) to
                            primitive int(i)
float f=fobj.floatValue()   //conversion of float object (fobj) to
                            primitive float(f)
double d=dobj.doubleValue() //conversion of double object (dobj) to
                            primitive double(d)
```

3. **Converting Numbers to Strings:**

```
String s3 = Integer.toString(i);  //conversion of primitive int(i) to
                                  String object
String s4 = Double.toString(d);   //conversion of primitive double(i) to
                                  String object
String s4 = Float.toString(d);    //conversion of primitive float(i) to
                                  String object
```

4. Converting Strings to Numbers:

```
int i=Integer.parseInt(str)
double d=Double.parseDouble(str)
```

Program 2.20: Program to demonstration of Integer wrapper class.

```
class WrapperInt
{
   public static void main(String args[])
   {
      Integer x = new Integer(141);
      byte b = x.byteValue();
      System.out.println("Byte value: "+b);
      int hash = x.hashCode();
      System.out.println("Hash code: "+hash);
      int val = Integer.parseInt("63");
      System.out.println("Parsed value: "+val);
      String s = x.toString();
      System.out.println("String value: "+s);
      Integer y = new Integer(Integer.valueOf("22"));
      System.out.println("Short value: "+y.shortValue());
      String bin = Integer.toBinaryString(59);
      String oct = Integer.toOctalString(59);
      String hex = Integer.toHexString(59);
      System.out.println("Binary of 59: "+bin);
      System.out.println("Octal of 59: "+oct);
      System.out.println("Hex of 59: "+hex);
   }
}
```

Output:

```
Byte value: -115
Hash code: 141
Parsed value: 63
String value: 141
Short value: 22
Binary of 59: 111011
Octal of 59: 73
Hex of 59: 3b
```

Program 2.21: Program to demonstration of character wrapper class.

```
class WrapperChar
{
   public static void main(String args[])
   {
      Character ch = new Character('X');
      System.out.print("Char value: ");
      System.out.println(ch.charValue());
      if(Character.isDigit('0'))
      System.out.println("0 is digit");
      if(Character.isLowerCase('R'))
         System.out.println("R is lower case");
      else
      {
         System.out.println("R is upper case");
         System.out.print("Lower case: ");
         System.out.println(Character.toLowerCase('R'));
      }
      if(Character.isWhitespace(' '))
         System.out.println("Character is white space");
   }
}
```

Output:

```
Char value: X
0 is digit
R is upper case
Lower case: r
Character is white space
```

2.11 PACKAGES [Oct. 16, 17, April 18, 19]

- Java is an object oriented programming language. Any Java programmer will quickly build a large number of different classes for use in each application that they develop.
- Initially it may be tempting to store the classes in the same directory as the end application, but as the complexity of applications grows so will the number of classes. The answer is to organize the classes into packages.
- A Java package is a mechanism for organizing Java classes. Packages are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc.

- A package can be defined as "a group of similar types of classes, interface, enumeration and sub-packages".
- Packages are collection of classes and interface or packages act as container for classes.
- By organizing our classes into packages we achieve following benefits:
  1. Easy to reuse the classes and interfaces which are defined in packages.
  2. Two classes in different packages can have same name e.g. student class in package1 and student class in package2.
  3. It provides a way to hide a class thus prevent other programs or packages to access this class.
  4. provide a way for separating design with coding.
- We can define package using 'package' keyword. There are two types of packages:
  1. **Built-in Package:** Existing built-in Java packages like java.lang, java.util, java.sql etc.
  2. **User-define Package:** Java package created by user to categorized classes, interface and enumeration.
- Java API provides a large number of classes which are grouped into different packages according to functionality.
- Fig. 2.6 shows the built in packages of Java.

Fig. 2.6

Sr. No.	Package Name	Description
1.	`java.lang`	Language support classes. They are used by java compiler itself and thus automatically imported. They include classes for primitive types, strings, math, tread and exceptions.
2.	`java.io`	Input/output support classes. They provide support for input and output.
3.	`java.util`	Language utility classes such as Vector, Date ,hash tables, random numbers etc.
4.	`java.awt`	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
5.	`java.applet`	Classes for creating and implementing applets.
6.	`java.net`	Classes for networking. They include classes for communication with local computers as well as with internet servers.

2.11.1 Creating Packages [Oct. 16, April 18, 19]

- To create a package is quite easy, simply include a package command as the first statement in a Java source file.
- Following is the general **form/syntax** of the package statement:

```
package pkg_name;
```

Here, pkg_name is the name of the package. For example, the following statement creates a package called MyPackage.

```
package MyPackage;
public class ClassA
{
    int x;
    public void accept(int x1)
    {
        x=x1;
    }
    public void display()
    {
        System.out.println("X="+x);
    }
}
```

- Note that the access specifier of class and methods is public to make it visible in all programs. Save this file as ClassA.java in MyPackage folder which we have created in bin directory of JDK.
- Compile this file as given below:

```
javac MyPackage\ClassA.java
```

2.11.2 Accessing Packages (Import a Package) [April 18]

- There are three ways to access the package from outside the package i.e., import package.*; import package.classname; fully qualified name.

1. Using packagename:

- If we use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The 'import' keyword is used to make the classes and interface of another package accessible to the current package.

Program 2.22: Program for package that import the package name.

```
//save by X.java
package pkg;
public class X
{
    public void msg(){System.out.println("Hello");}
}
```

```
//save by y.java
package mypkg;
import pkg.*;
class Y
{
   public static void main(String args[])
   {
      X obj = new X();
      obj.msg();
   }
}
```

Output:

```
Hello
```

2. Using packagename.classname:

- If we import package.classname then only declared class of this package will be accessible.

```
//save by X.java
package pkg;
public class X
{
   public void msg(){System.out.println("Hello");}
}
//save by Y.java
package mypkg;
import pkg.X;
class Y
{
   public static void main(String args[])
   {
      X obj = new X();
      obj.msg();
   }
}
```

Output:

```
Hello
```

3. Using Fully Qualified Name:

- If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But we need to use fully qualified name every time when we are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example:

```
//save by X.java
package pkg;
public class X
{
   public void msg(){System.out.println("Hello");}
}
//save by Y.java
package mypkg;
class Y
{
   public static void main(String args[])
   {
      pack.X obj = new pack.X();   //Using fully qualified name
      obj.msg();
   }
}
```

Output:

```
Hello
```

2.11.3 Using System Package

- The system packages are organized into hierarchical structure. It is shown in Fig. 2.7.

Fig. 2.7: java.lang.Math Package

- We can access the methods in our program just by including these built in packages as shown below:

```
import packagename.classname;
```

OR

```
import packagename.*;
```

- This statement should be the first statement. These packages are named by standard java naming rules:
  - packages begins with lowercase letters.
  - The class name always begins with an uppercase letter.
  - Method is again in small letters.

For example:

```
double k = java.lang.Math.sqrt(x);
```

↑ ↑ ↑

package name class method name

Program 2.23: Program shows the usage of built in packages.

```
import java.lang.Math;
class mat
{
    public static void main(String args[])
    {
        double K = java.lang.Math.sqrt(25);
        System.out.println(k);
    }
}
```

Output:

```
5.0
```

2.11.4 User Defined Packages [April 17, 18, Oct. 17]

- The packages credited by user are called as user defined package.
- User defined packages generally represent programs data.

Creating Own Packages:

- We should first declare our own package. This has to be first statement in Java source file. We can have comments and spaces at the top. And then define the class in the same program.

- The **syntax** is shown below:

```
package mypackage;      //package declared
public class myclass    //class defined
   {
      _________
      _________ (body of class)
      _________
   }
```

- When the above source file is compiled java first create ·class file and store it in the same directory. Note that ·class file must be located in a directory that has same name as the package.
- The creation of user-defined packages has to follow the following rules:
  - declare a package as shown in the syntax.
  - class is defined.
  - class should be public.
  - a subdirectory is created under main directory which is useful to store main source file.
  - store the listing as classname.java file in the subdirectory created.
  - create ·class file using javac compiler.
  - package and directory name must be same.

Accessing Own Packages:

- We should use import statement to use the packages as shown below:

```
import pack1 [.pack2] [.pack3].classname;
```

For example:

```
import mypackage·*;
import mypackage.myclass;
```

Using Own Packages:

- Consider that the directory is jdk \bin. In this, let's create one subdirectory called 'pack' in which we are creating our own packages.

```
/exports/usr$ mkdir pack
/exports/usr$ cd pack
/exports/usr/pack$
```

- Now in this we can write our own source files. Program 2.24 shows package and Program 2.25 shows the source file which imports the package.

Program 2.24:

```
//packcheck.java
package pack;
public class packcheck
{
   public void disp()
   {
      System.out.println ("packcheck class");
   }
}
```

It is stored as packcheck.java in pack subdirectory. We should compile this java file so that its ·class file is created.

Output:

```
exports/usr/pack$
javac packcheck.java
```

Program 2.25:

```
import pk.packcheck;
class test
{
   public static void main(String args[])
   {
      pack check ob1 = new packcheck();
      ob1.disp();
   }
}
```

This is stored as text ·java. Compile this program and the execute its source file.

```
/exports/usr/pack$ javac test·java
/exports/usr/pack$ java.test
```

Output:

```
packcheck class
```

- We can include or import more than one packages in our program. Program 2.26 shows another package and Program 2.26 shows our program which calls both the above packages.

Program 2.26:

```
package mypack;
public class check
{
   protected int k = 5;
   public void display()
   {
      System.out.println("k =" +k);
   }
}
```

Program 2.27:

```
//importing two packages
import pk.packcheck;
import mypack.check;
class doubletest
{
   public static void main(String args[])
   {
      packcheck ob1 = new packcheck();
      ob1.disp();
      check ob2 = new check();
      ob2.display();
   }
}
```

Output:

```
packcheck class
k = 5
```

- This shows that we can import multiple packages in our program, similarly two or more packages contains identical class names:

  For example:

```
package pck1;
public class manager
{ ............. }
public class emp
{ ............. }
package pck2;
public class CEO
{ ............. }
public class emp
{ ............. }
```

- The java runtime system uses the current working directory as its starting point. This is first way using which, Java finds its packages and second way as it sets the CLASSPATH environment variable's path. The first way is always easy to use.

PROGRAMS

Program 1: Program for class and object.

```
class Student
{
   int studno;        //declaration of variable
   String studname;
}
class Examplestud
{
   public static void main(String args[])
   {
      Student s1=new Student();
      s1.studno = 100;
      s1.studname = "JAVA";
      System.out.println("Student id  = "+s1.studno);    //display members
      System.out.println("Student name  = "+s1.studname);

    }
}
```

Output:

```
Student id  = 100
Student name  = JAVA
```

Program 2: Program for private member.

```
class X
{
     private String name;   // declaration of private variable
}
public class Testprivate
{
   public static void main(String[] main)
   {
      X  obj  = new X();
      obj.name = "Rudra";   // access private variable  from another class
      System.out.println("Welcome" + obj.name);
   }
}
```

Output:

```
Compile time error
javac Testprivate.java
Testprivate.java:15: error: name has private access in X
obj.name = "Rudra";    // access private variable  from another class
^
Testprivate.java:16: error: name has private access in X
System.out.println("Welcome" + obj.name);
^
2 errors
```

Program 3: Program to define a class 'employee' with data members as empolyee id, employee name and employee salary. Accept data for 'n' objects using Array of objects and print it.

```
import java.util.Scanner;
public class Employee_record
{
   int emp_id;
   String emp_name;
   float emp_salary;
   public void emp_getdata()
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Enter the Employee id: ");
      emp_id = in.nextInt();
      System.out.print("Enter the Employee Name:");
      emp_name = in.next();
      System.out.print("Enter the Employee Salary: ");
      emp_salary = in.nextFloat();
   }
   public void emp_putdata()
   {
      System.out.print(" " + emp_id);
      System.out.print("\t" + emp_name);
      System.out.print("\t" + emp_salary);
      System.out.println();
   }
```

```
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        Employee_record emp_obj[] = new Employee_record[50];
                                                    //create array of object
        System.out.print("How many record you want enter  of Employee  ");
        int n = sc.nextInt();
        for(int i=0; i<n; i++)
        {
            emp_obj[i] = new Employee_record();
            emp_obj[i].emp_getdata();
        }
        System.out.println("**** Employee Information ****");
        System.out.println("Emp id \t Emp Name \t Emp Salary\n");
        for(int i=0; i<n; i++)
        {
            emp_obj[i].emp_putdata();
        }
    }
}
```

Output:

```
How many record you want enter of Employee 2
Enter the Employee id: 1
Enter the Employee Name:aaa
Enter the Employee Salary: 20000
Enter the Employee id: 2
Enter the Employee Name:www
Enter the Employee Salary: 25000

**** Employee Information ****
Emp id    Emp Name    Emp Salary
   1       aaa         20000.0
   2       www         25000.0
```

Program 4: Program to display the department information using 'this' keyword.

```
class Dept
{
    int deptno;
    String deptname;
    Dept()
    {
    }
```

```
    Dept(int deptno, String deptname)
    {
        this.deptno= deptno;
        this.deptname = deptname;
    }
    void display()
    {
        System.out.println("\nDepartment No = " + deptno +
                                        "\nDepartment Name = "+deptname);
    }
} //end of Dept
class Ex_this
{
    public static void main(String args[])
    {
        Dept d = new Dept(87, "Computer Science");
        d.display();
    }
}//end of class Ex_this
```

Output:

```
Department No = 87
Department Name = Computer Science
```

Program 5: Program to display the item name and item price using static variable.

```
class Item
{
    int itemprice;            //instance variable
    String itemname;
    static String itemcode ="ISI";//static variable
    Item(int p, String n)     //constructor
    {
        itemprice = p;
        itemname = n;
    }
    void display()            //method to display the values
    {
        System.out.println(itemprice+" "+itemname+" "+itemcode);
    }
}
```

```
public class Teststaticvar
{
   public static void main(String args[])
   {
      Item i1 = new Item(11,"Cheese");
      Item i2 = new Item(22,"Ice cream");
      i1.display();
      i2.display();
   }
}
```

Output:

```
11 Cheese ISI
22 Ice cream ISI
```

Program 6: Program to get the cube of a given number using the static method.

```
import java.util.Scanner;
class Teststaticmethod
{
   static int cube(int x)
   {
      return x*x*x;
   }
   public static void main(String args[])
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Enter the Number: ");
      int num = in.nextInt();
      int cal = Teststaticmethod.cube(num);
      System.out.println("Cube is = "+cal);
   }
}
```

Output:

```
Enter the Number: 3
Cube is = 27
```

Program 7: Program for static block.

```
class Teststatic
{
   static
   {
      System.out.println("Hello Static Block ");
   }
```

```
    public static void main(String args[])
    {
        System.out.println("Hello Main Method");
    }
}
```

Output:

```
Hello Static Block
Hello Main Method
```

Program 8: Program to display the player number and name using of default constructor.

```
class TestConst
{
    int playnumber;
    String playname;
    public TestConst()
    {
        playnumber = 10  ;
        playname = "Sachin Tendulkar";
    }
    public static void main(String[] args)
    {
        TestConst tc = new TestConst();  // A default constructor is called
        System.out.println("Default Value:");
        System.out.println("Play number  = "+tc.playnumber);
        System.out.println("Play Name  = " + tc.playname);
    }
}
```

Output:

```
java TestConst
Default Value:
Play number  = 10
Play Name  = Sachin Tendulkar
```

Program 9: Program to display the information of player using parameterized constructor.

```
import java.util.Scanner;
class Player_record
{
    int player_id;
    String player_name;
    int player_run;
```

```
    public Player_record()   //default constructor
    {
       player_id = 7;
       player_name = "Ms Dhoni";
       player_run = 1000;
    }
    public Player_record(int player_id,String player_name,int player_run)
                                                  //paramterized constructor
    {
       this.player_id = player_id;
       this.player_name = player_name;
       this.player_run = player_run;
    }
    public void player_putdata()
    {
       System.out.println("Player Id = " + player_id);
       System.out.println("Player Name = " + player_name);
       System.out.println("Player Run = " + player_run);
    }
    public static void main(String[] args)
    {
       System.out.println("*** Default Player Information ***");
       Player_record obj = new Player_record(); //object create for
                    //default constructor
       obj.player_putdata();
       System.out.println("*** Player Information ***");
       Player_record ob = new Player_record(18,"Virat",1200);
                   //object create for paramterized constructor
       ob.player_putdata();
    }
}//end of class
```

Output:

```
*** Default Player Information ***
Player Id = 7
Player Name = Ms Dhoni
Player Run = 1000
 *** Player Information ***
Player Id = 18
Player Name = Virat
Player Run = 1200
```

Program 10: Program for constructor with array of objects.

```
import java.util.Scanner;
class Player_record
{
   int player_id;
   String player_name;
   int player_run;
   public Player_record()   //default constructor
   {
      player_id = 7;
      player_name = "Ms Dhoni";
      player_run = 1000;
   }
   public Player_record(int player_id,String player_name,int player_run)
                                                  //paramterized constructor
   {
      this.player_id = player_id;
      this.player_name = player_name;
      this.player_run = player_run;
   }
   public void player_getdata()
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Enter the Player id: ");
      player_id = in.nextInt();
      System.out.print("Enter the Player Name:");
      player_name = in.next();
      System.out.print("Enter the Player Run: ");
      player_run = in.nextInt();
   }
   public void player_putdata()
   {
      System.out.print(" " + player_id);
      System.out.print("\t" + player_name);
      System.out.print("\t" + player_run);
      System.out.println();
   }
}//class Player_record
```

```
class Exampleparaconst
{
   public static void main(String[] args)
   {
      int lim,i;
      Scanner in = new Scanner(System.in);
         System.out.println("*** Default Player Information ***");
         Player_record obj = new Player_record();
                                  //object create for default constructor
         obj.player_putdata();
         System.out.print("How many record you want to enter ? ");
         lim = in.nextInt();
      Player_record obj1[] = new Player_record[10];
                              //object create for paramterized constructor
      for(i=0; i<lim; i++)
      {
         obj1[i] = new Player_record();
      }
      for(i=0; i<lim; i++)
      {
         obj1[i].player_getdata();
      }
      System.out.println("*** Player Information ***");
      System.out.println("Player Id \t Player Name\tPlayer Run");
      for(i=0; i<lim; i++)
      {
         obj1[i].player_putdata();
      }
   }
}//end of class
```

Output:

```
  *** Default Player Information ***
7       Ms Dhoni        1000
How many record you want to enter ? 2
Enter the Player id: 5
Enter the Player Name:Yuvraj
Enter the Player Run: 1800
```

```
Enter the Player id: 3
Enter the Player Name:Virendra
Enter the Player Run: 2300
*** Player Information ***
Player Id    Player Name     Player Run
   5           Yuvraj           1800
   3           Virendra         2300
```

Program 11: Program to use of some of 'String' class methods.

```
public class Exstringmethod
{
   public static void main(String[] args)
   {
      int strlen;
      String str = " Core Java ";
      String s1 = "", s2 = "";
      System.out.println("String = "+str);
      strlen = str.length();      // Find length of string
      System.out.println("Lenght of String = " + strlen );
      s1 = str.replace("Core", "Advance");// Replace characters in string
      System.out.println("Before replacing:" + str);
      System.out.println("After replacing :" + s1);
      s2 = str.concat(" is Programming Language...");
                                 // Concatenating string with another string
      System.out.println("Before concate: " + str);
      System.out.println("After concate: " + s2);
   }
}
```

Output:

```
String =  Core Java
Lenght of String = 11
Before replacing: Core Java
After replacing : Advance Java
Before concate:  Core Java
After concate:  Core Java is Programming Language...
```

Program 12: Program to use of some 'StringBuffer' class methods.

```
public class Exstringbuffer
{
   public static void main(String[] args)
   {
      StringBuffer strname = new StringBuffer(50);
      System.out.println("String capacity: "+strname.capacity());
      strname = new StringBuffer("Java programming");
      System.out.println("String : "+strname);
      System.out.println("Last index of 'ing' is: "
                                          + strname.lastIndexOf("ing"));
      System.out.println("Original String is: "+strname);
      StringBuffer strrev = strname.reverse();
      System.out.println("Reverse String: "+strrev);
   }
}
```

Output:

```
String capacity: 50
String : Java programming
Last index of 'ing' is: 13
Original String is: Java programming
Reverse String: gnimmargorp avaJ
```

Program 13: Program for string methods.

```
import java.lang.*;
import java.util.Scanner;
public class Exstringformatting
{
   public static void main(String args[])
   {
      Scanner in = new Scanner(System.in);
      String pass1 = "password*";
      String pass2;
      System.out.println ("Enter the Password");
      pass2 = in.next();
      if(pass1.compareTo(pass2)==0)
      {
         System.out.println ("Correct Password");
      }
```

```
        else
        {
           System.out.println ("InCorrect Password");
           System.exit(0);
        }
        String msg = "Welcome to Java!!!";
        StringBuffer str = new StringBuffer(msg);
        str.append(pass1);
        System.out.println ("Appended  = "+str);
        StringBuffer str1 = new StringBuffer(pass1);
        str1=str1.reverse();
        System.out.println ("String in Reverse Order = "+str1);
        System.out.println
                     ("Replaced '*' with '$' = "+pass1.replace('*','$'));
     }
  }
```

Output:

```
Enter the Password
password*
Correct Password
Appended = Welcome to Java!!!password*
String in Reverse Order = *drowssap
Replaced '*' with '$' = password$
```

Program 14: Program to create a package named Series having class to print series square of numbers. Write a program to generate 'n' terms of the above series.

```
package series;
public class Sqr
{
   public void square(int n)
   {
      System.out.println("Square of the number is "+(n*n));
   }
}
import series.*;
import java.io.*;
import java.util.Scanner;
```

```
public class Seriessqr
{
   public static void main(String [] args)
   {
      Sqr sq = new Sqr()
      System.out.print("Enter the no: ");
      n = in.nextInt();
      sq.square(n);
   }
}
```

Output:

```
Enter the no: 6
Square of the number is 36
```

Program 15: Program to create a Package "SY" which has a class SYMarks, (members – ComputerTotal, MathsTotal, and ElectronicsTotal). Create another package TY which has a class TYMarks (members – Theory, Practicals). Create n objects of Student class (having rollNumber, name, SYMarks and TYMarks). Add the marks of SY and TY computer subjects and calculate the Grade ('A' for >= 70, 'B' for >= 60 'C' for >= 50 , Pass Class for > =40 else 'FAIL') and display the result of the student in proper format.

```
package SY;
import java.io.BufferedReader;
import java.io.*;
public class SYClass
{
   public int ct,mt,et;
   public void get() throws IOException
   {
      System.out.println("Enter marks of students for computer, maths and
                                 electronics subject out of 200 ");
      BufferedReader br=new BufferedReader
                                     (new InputStreamReader(System.in));
      ct=Integer.parseInt(br.readLine());
      mt=Integer.parseInt(br.readLine());
      et=Integer.parseInt(br.readLine());
   }
}
package TY;
import java.io.*;
```

```
public class TYClass
{
   public int tm,pm;
   public void get() throws IOException
   {
      System.out.println("Enter the marks of the theory out of 400 and
      practicals out of 200: ");
      BufferedReader br=new BufferedReader
                                    (new InputStreamReader(System.in));
      tm=Integer.parseInt(br.readLine());
      pm=Integer.parseInt(br.readLine());
   }
}
import SY.*;
import TY.*;
import java.io.*;
class StudentInfo
{
   int rollno;
   String name,grade;
   public float gt,tyt,syt;
   public float per;
   public void get() throws IOException
   {
      System.out.println("Enter roll number and name of the student: ");
      BufferedReader br=new BufferedReader
                                    (new InputStreamReader(System.in));
      rollno=Integer.parseInt(br.readLine());
      name=br.readLine();
   }
}
public class StudentMarks
{
   public static void main(String[] args) throws IOException
   {
      BufferedReader br=new BufferedReader
                                    (new InputStreamReader(System.in));
```

```
        System.out.println("Enter the number of students:");
        int n=Integer.parseInt(br.readLine());
        SYClass sy[]=new SYClass[n];
        TYClass ty[]=new TYClass[n];
        StudentInfo si[]=new StudentInfo[n];
        for(int i=0;i<n;i++)
        {
           si[i]=new StudentInfo();
           sy[i]=new SYClass();
           ty[i]=new TYClass();
           si[i].get();
           sy[i].get();
           ty[i].get();
           si[i].syt=sy[i].ct+sy[i].et+sy[i].mt;
           si[i].tyt=ty[i].pm+ty[i].tm;
           si[i].gt=si[i].syt+si[i].tyt;
           si[i].per=(si[i].gt/1300)*100;
           if(si[i].per>=70) si[i].grade="A";
           else if(si[i].per>=60) si[i].grade="B";
           else if(si[i].per>=50) si[i].grade="C";
           else if(si[i].per>=40) si[i].grade="Pass";
           else si[i].grade="Fail";
        }
        System.out.println
          ("Roll No\tName\tSyTotal\tTyTotal\tGrandTotal\tPercentage\tGrade");
        for(int i=0;i<n;i++)
        {
           System.out.println(si[i].rollno+"\t"+si[i].name+
           "\t"+si[i].syt+"\t"+si[i].tyt+"\t"+si[i].gt+
           "\t\t"+si[i].per+"\t\t"+si[i].grade);
        }
     }
  }
```

Output:

```
Enter the number of students:
3
Enter roll number and name of the student:
20
amita
```

```
Enter marks of students for computer, maths and electronics subject out of
200
178
168
156
Enter the marks of the theory out of 400 and practicals out of 200:
388
179
Enter roll number and name of the student:
30
ram
Enter marks of students for computer, maths and electronics subject out of
200
159
146
187
Enter the marks of the theory out of 400 and practicals out of 200:
364
144
Enter roll number and name of the student:
40
sunny
Enter marks of students for computer, maths and electronics subject out of
200
126
80
154
Enter the marks of the theory out of 400 and practicals out of 200:
205
100
Roll No   Name        SyTotal    TyTotal    GrandTotal Percentage Grade
20       amita        502.0      567.0      1069.0     89.08333    A
30       ram          492.0      508.0      1000.0     83.33333    A
40       sunny        360.0      305.0      665.0      55.416668   C
```

Program 16: Program to convert all primitives into its corresponding Wrapper objects.

```
public class Examplewrapper
{
   public static void main(String args[])
   {
      byte b=50;
      short s=40;
      int i=70;
      long l=20;
      float f=3.14F;
      double d=67.0D;
      char c='M';
      boolean bool=true;
      // Converting primitives into objects
      Byte byteobj=b;
      Short shortobj=s;
      Integer intobj=i;
      Long longobj=l;
      Float floatobj=f;
      Double doubleobj=d;
      Character charobj=c;
      Boolean boolobj=bool;
      //Printing objects
      System.out.println("---Printing object values---");
      System.out.println("Byte object: "+byteobj);
      System.out.println("Short object: "+shortobj);
      System.out.println("Integer object: "+intobj);
      System.out.println("Long object: "+longobj);
      System.out.println("Float object: "+floatobj);
      System.out.println("Double object: "+doubleobj);
      System.out.println("Character object: "+charobj);
      System.out.println("Boolean object: "+boolobj);
   }
}
```

Output:

```
--Printing object values---
Byte object: 50
Short object: 40
Integer object: 70
Long object: 20
Float object: 3.14
Double object: 67.0
Character object: M
Boolean object: true
```

Program 17: program of equal() method.

```
public class Exequals
{
   public static void main(String args[])
   {
      String st1="java";
      String st2="java";
      String st3="JAVA";
      String st4="cpp";
      System.out.println(st1.equals(st2));  //TRUE because content and case
                                            is same
      System.out.println(st1.equals(st3));  //FALSE  because  case  is  not
                                            same
      System.out.println(st1.equals(st4));  //FALSE because content is not
                                            same
   }
}
```

Output:

```
true
false
false
```

Program 18: Program to demonstrate working of getClass().

```
public class Testget
{
   public static void main(String[] args)
   {
      Object ob = new String("Welcome");
      Class cl = ob.getClass();
      System.out.println("Class of Object  is: "+ cl.getName());
   }
}
```

Output:

```
Class of Object  is: java.lang.String
```

Program 19: Program to display the person information using toString() method.

```
class Person
{
   int pno;
   String pname;
   String pcity;
```

```
    Person(int pno, String pname, String pcity)
    {
        this.pno=pno;
        this.pname=pname;
        this.pcity=pcity;
    }
    public String toString() //overriding the toString() method
    {
        return "\nPerson No = "+pno+" \nPerson Name ="+pname+"\nPerson City
                                                                = "+pcity;
    }
    public static void main(String args[])
    {
        Person p1=new Person(11,"Rma","Pune");
        Person p2=new Person(12,"Atharva","Mumbai");
        System.out.println(p1);
        System.out.println(p2);
    }
}
```

Output:

```
Person No = 11
Person Name = Rma
Person City = Pune
Person No = 12
Person Name = Atharva
Person City = Mumbai
```

Program 20: Program to create a class Account having variable accno, accname and balance. Define deposite() and withdraw() methods. Create one object of class and perform the operation.

```
import java.io.*;
class Account
{
   int accno;
   String accname;
   double balance, new_bal;
   Account(int accno, String accname, double balance)
   {
       this.accno = accno;
       this.accname = accname;
       this.balance = balance;
    }
```

```
    void deposite(double deposit_amount)
    {
       balance = balance+deposit_amount;
       System.out.println("Your new available balance is: "+balance);
    }
    void withdraw(double amount)
    {
       if(balance > amount)
       {
          balance = balance-amount;
          System.out.println("Your current balance: "+balance);
       }
       else if(balance == amount)
       {
          System.out.println("Your  current  balance  is  "+balance+".  Your
          minimum balance
                                 should be 1000. Hence cannot withdraw.");
       }
       else
       {
          System.out.println("Insufficient balance");
       }
    }
    public static void main(String args[])
    {
 BufferedReader bin = new BufferedReader(new InputStreamReader(System.in));
       Account a;
       double amount, bal;
       try
       {
 System.out.println("Enter the account name, account number and balance: ");
          String a_name = bin.readLine();
          int a_no = Integer.parseInt(bin.readLine());
          bal = Double.parseDouble(bin.readLine());
          a = new Account(a_no, a_name, bal);
    System.out.println("Enter\n 1 for depositing \n 2 for withdrawal");
          int option = Integer.parseInt(bin.readLine());
          switch(option)
          {
             case 1:
             System.out.println("Enter the amount to deposite: ");
             amount = Double.parseDouble(bin.readLine());
             a.deposite(amount);
             break;
```

```
                case 2:
                System.out.println("Enter the amount to withdraw: ");
                amount = Double.parseDouble(bin.readLine());
                a.withdraw(amount);
                break;
                default:
                System.out.println("Enter a valid option: ");
                break;
            }
        }
        catch(Exception e)
        {
            System.out.println("Exception caught: "+e);
        }
    }
}
```

Output:

```
Enter the account name, account number and balance:Saving     1001   5000
Enter
 1 for depositing
 2 for withdrawal
1
Enter the amount to deposite:5000
Your new available balance is: 10000.0
```

Program 21: Program to define a class 'employee' with data member's empid, name and salary. Accept data for two objects using array of objects and print it.

```
import java.io.*;
class employee
{
    int empid;
    String name;
    double salary;
    void getdata()
    {
        try
        {
            BufferedReader obj = new BufferedReader(new
                                        InputStreamReader(System.in));
            System.out.print("Enter Emp number: ");
            empid=Integer.parseInt(obj.readLine());
            System.out.print("Enter Emp Name: ");
            name=obj.readLine();
            System.out.print("Enter Emp Salary: ");
            salary=Double.parseDouble(obj.readLine());
        }
```

```
        catch(IOException e)
        {
            System.out.println(e);
        }
    }
    void show()
    {
        System.out.println("Emp ID: "+empid);
        System.out.println("Name: "+name);
        System.out.println("Salary: "+salary);
    }
}
class EmpDetails
{
    public static void main(String args[])
    {
        employee e[] = new employee[5];
        for(int i=0; i<5; i++)
        {
            e[i] = new employee();
        }
        for(int i=0; i<5; i++)
        {
            e[i].getdata();
        }
        System.out.println("Employee Details are: ");
        for(int i=0; i<5; i++)
        {
            e[i].show();
        }
    }
}
```

Output:

```
Enter Emp number: 1001
Enter Emp Name: Ravi
Enter Emp Salary: 50000
Enter Emp number: 1002
Enter Emp Name: Prashant
Enter Emp Salary: 40000
Employee Details are:
Emp ID: 1001
Name: Ravi
Salary: 50000.0
Emp ID: 1002
Name: Prashant
Salary: 40000.0
```

Program 22: Program to define a class 'Book' with data members bookid, bookname and price. Accept data for three objects using Array of objects and display it.

```
import java.lang.*;
import java.io.*;
class Book
{
   String bookname;
   int bookid;
   int price;
   BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
   void getdata()
   {
      try
      {
         System.out.println("Enter Book ID: ");
         bookid=Integer.parseInt(br.readLine());
         System.out.println("Enter Book Name: ");
         bookname=br.readLine();
         System.out.println("Enter Price: ");
         price=Integer.parseInt(br.readLine());
      }
      catch(Exception e)
      {
         System.out.println("Error");
      }
   }
   void display()
   {
      System.out.println("Book ID: "+bookid);
      System.out.println("Book Name: "+bookname);
      System.out.println("Price: "+price);
   }
}
class bookdata
{
   public static void main(String args[])
   {
      Book b[]=new Book[7];
      for(int i=0;i<7;i++)
      {
         b[i]=new Book();
      }
```

```
        for(int i=0;i<7;i++)
        {
           b[i].getdata();
        }
        for(int i=0;i<7;i++)
        {
           b[i].display();
        }
     }
  }
```

Output:

```
Enter Book ID:
1001
Enter Book Name:
JPR
Enter Price:
200
Enter Book ID:
1002
Enter Book Name:
MIC
Enter Price:
250
Enter Book ID:
1003
Enter Book Name:
SEN
Enter Price:
150
Book ID: 1001
Book Name: JPR
Price: 200
Book ID: 1002
Book Name: MIC
Price: 250
Book ID: 1003
Book Name: SEN
Price: 150
```

Program 23: Program to calculate gross salary of employees if Basic, HRA and DA are given.

```
import java.io.*;
import java.lang.*;
class Employee
{
   String name;
   int salary;
   double da,hra,gr;
   BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
```

```
    void get()
    {
        try
        {
            System.out.println("Enter the name: ");
            name=br.readLine();
            System.out.println("Enter the salary: ");
            salary=Integer.parseInt(br.readLine());
        }
        catch(Exception ex)
        {
            System.out.println(ex);
        }
    }
    void show()
    {
        System.out.println("Name is: "+name);
        System.out.println("Basic salary is: "+salary);
        hra=(salary*35)/100;
        da=(salary*60)/100;
        gr=hra+da+salary;
        System.out.println("Gross salary is: "+gr);
    }
}
class EmpGrossSalary
{
    public static void main(String args[])
    {
        Employee e[]=new Employee[5];
        for(int i=0;i<5;i++)
        {
            e[i]=new Employee();
        }
        for(int i=0;i<5;i++)
        {
            e[i].get();
        }
        for(int i=0;i<5;i++)
        {
            e[i].show();
        }
    }
}
```

Output:

```
Enter the name:Ravi
Enter the salary:50000
Enter the name:Prashant
Enter the salary:40000
Name is: Ravi
Basic salary is: 50000
Gross salary is: 97500.0
Name is: Prashant
Basic salary is: 40000
Gross salary is: 78000.0
```

Program 24: Program to define a class person with data member as Aadharno, name, Panno implement concept of constructor overloading. Accept data for 5 objects and print it.

```
import java.io.*;
class Person
{
   int Aadharno;
   String name;
   String Panno;
   Person(int Aadharno, String name, String Panno)
   {
      this.Aadharno = Aadharno;
      this.name = name;
      this.Panno = Panno;
   }
   Person(int Aadharno, String name)
   {
      this.Aadharno = Aadharno;
      this.name = name;
      Panno = "Not Applicable";
   }
   void display()
   {
      System.out.println("Aadharno is: "+Aadharno);
      System.out.println("Name is: "+name);
      System.out.println("Panno is: "+Panno);
   }
   public static void main(String args[])
   {
      BufferedReader br = new BufferedReader(new
                                          InputStreamReader(System.in));
      Person p, p1, p2, p3, p4;
      int a;
      String n, pno;
```

```
        try
        {
            System.out.println("Enter Aadhar no: ");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name: ");
            n = br.readLine();
            System.out.println("Enter panno: ");
            pno = br.readLine();
            p = new Person(a,n,pno);
            System.out.println("Enter Aadhar no: ");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name: ");
            n = br.readLine();
            System.out.println("Enter panno: ");
            pno = br.readLine();
            p1 = new Person(a,n,pno);
            System.out.println("Enter Aadhar no: ");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name: ");
            n = br.readLine();
            p2 = new Person(a,n);
            System.out.println("Enter Aadhar no: ");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name: ");
            n = br.readLine();
            p3 = new Person(a,n);
            System.out.println("Enter Aadhar no: ");
            a = Integer.parseInt(br.readLine());
            System.out.println("Enter name: ");
            n = br.readLine();
            System.out.println("Enter panno: ");
            pno = br.readLine();
            p4 = new Person(a,n,pno);
            p.display();
            p1.display();
            p2.display();
            p3.display();
            p4.display();
        }
        catch(Exception e)
        {
            System.out.println("Exception caught: "+e);
        }
    }
}
```

Output:

```
Enter Aadhar no: 1001
Enter name: Ravi
Enter panno: ABCD
Enter Aadhar no: 1002
Enter name: Prashant
Enter panno: SCVB
Aadharno is: 1001
Name is: Ravi
Panno is: ABCD
Aadharno is: 1002
Name is: Prashant
Panno is: SCVB
```

Program 25: Program to demonstrate the use of "this" keyword.

```
class Testthis
{
   int a;
   int b;
   Testthis(int a, int b)                 // Parameterized constructor
   {
      this.a = a;
      this.b = b;
   }
   void display()
   {
      System.out.println("a = " + a + " b = " + b); //Displaying value of
                                                       variables a and b
   }
   public static void main(String[] args)
   {
      Test this object = new Testthis(10, 20);
      object.display();
   }
}
```

Output:

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

Program 26: Program of single level inheritance.

```
class Room
{
   int length, breadth;
   Room(int a, int b)
   {
      length=a;
      breadth=b;
   }
```

```
    int area()
    {
        return(length*breadth);
    }
}
class Bedroom extends Room
{
    int height;
    Bedroom(int a, int b,int c)
    {
        super(a, b);
        height=c;
    }
    int volume()
    {
        return(length*breadth*height);
    }
}
class InnerTest1
{
    public static void main(String args[])
    {
        Bedroom room1=new Bedroom(14,12,10);
        int area1=room1.area();
        int volume1=room1.volume();
        System.out.println("Area: "+area1);
        System.out.println("Volume: "+volume1);
    }
}
```

Output:

```
Area: 168
Volume: 1680
```

Program 27: Program to create a class 'Rectangle' that contains 'length' and 'width' as data members. From this class derive class box which has additional data member 'depth'. Class 'Rectangle' consists of a constructor and an area() function. The derived 'Box' class have a constructor and override function named area() which returns surface area of 'Box' and a volume() function. Write a java program calling all the member function.

```
import java.io.*;
class Rectangle
{
    int length, width;
```

```
    Rectangle(int length, int width)
    {
       this.length = length;
       this.width = width;
    }
    int area()
    {
       return length*width;
    }
}
class Box extends Rectangle
{
    int depth;
    Box(int length, int width, int depth)
    {
       super(length, width);
       this.depth = depth;
    }
    int area()
    {
       return 2*(length*width+width*depth+depth*length);
    }
    int volume(int l, int w, int d)
    {
       return l*w*d;
    }
    public static void main(String args[])
    {
       BufferedReader bin = new BufferedReader(new
                                          InputStreamReader(System.in));
       try
       {
          System.out.println("Enter the length, width and depth: ");
          int l = Integer.parseInt(bin.readLine());
          int w = Integer.parseInt(bin.readLine());
          int h = Integer.parseInt(bin.readLine());
          Box b = new Box(l,w,h);
          Rectangle r = new Rectangle(l,w);
          System.out.println("Area of the Rectangle is: "+r.area());
          System.out.println("Area of the Box is: "+b.area());
          System.out.println("Volume of the Rectangle is: "+b.volume(l,w,h));
       }
```

```
        catch(Exception e)
        {
            System.out.println("Exception caught: "+e);
        }
    }
}
```

Output:

```
Enter the length, width and depth:10   20   30
Area of the Rectangle is: 200
Area of the Box is: 2200
Volume of the Rectangle is: 6000
```

Program 28: Program to define a class student with roll no. and name, derive class attendance with member as present days. Write method to calculate and print average attendance assuming total days as 180.

```
class stud
{
    int rn;
    String name;
    stud(int r, String n)
    {
        rn = r;
        name = n;
    }
    void put()
    {
        System.out.println("Roll no:"+rn);
        System.out.println("Name:"+name);
    }
}
class stud_att_cal extends stud
{
    double preentday;
    stud_att_cal(int r, String n, int p)
    {
        super(r,n);
        preentday = p;
    }
    void calculate()
    {
        double avg = (100 * preentday)/180;
        put();
        System.out.println("Present Days : "+preentday);
        System.out.println("Average attendance is: "+avg);
    }
}
```

```
class studAttendance
{
   public static void main(String args[])
   {
      stud_att_cal S1 = new stud_att_cal(2,"Ravi",137);
      S1.calculate();
   }
}
```

Output:

```
Roll no:2
Name:Ravi
Present Days: 137.0
Average attendance is: 76.11111111111111
```

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. Which of following method of class StringBuffer is used to concatenate the string representation to the end of invoking string?
   (a) concatenate() (b) concat()
   (c) append() (d) join()
2. Which of these access specifiers can be used for a class so that its members can be accessed by a different class in the different package?
   (a) No Modifier (b) public
   (c) protected (d) private
3. Which of these methods of class StringBuffer is used to find the length of current character sequence?
   (a) length() (b) Length()
   (c) capacity() (d) Capacity()
4. Which of these classes is used to create an object whose character sequence is mutable?
   (a) String() (b) StringBuffer()
   (c) String() and StringBuffer() (d) stringbuffer()
5. Overloading of constructors in Java means adding more than _____ constructors with the different argument list.
   (a) 1 (b) 2
   (c) 3 (d) 8
6. What is not the use of “this” keyword in Java?
   (a) Passing itself to another method
   (b) Calling another constructor in constructor chaining
   (c) Referring to the instance variable when local variable has the same name
   (d) Passing itself to method of the same class

7. What would be behaviour if the constructor has a return type?
   (a) Compilation error (b) Runtime error
   (c) Compilation and runs successfully (d) Only String return type is allowed
8. Which of these methods must be made static?
   (a) finalize() (b) delete()
   (c) run() (d) main()
9. Which of these cannot be declared static?
   (a) class (b) object
   (c) variable (d) method
10. Which of these method of String class can be used to test to strings for equality?
   (a) isequal() (b) isequals()
   (c) equal() (d) equals()
11. Which of these cannot be declared static?
   (a) class (b) object
   (c) variable (d) method
12. Choose the correct option for the code given below:
   `class A{ static int c=0; public: A(){ c++; } };`
   (a) Constructor will make c=1 for each object created
   (b) Constructor will make c=0 for each object created
   (c) Constructor will keep number of objects created
   (d) Constructor will just initialize c=0 then increment by 1
13. What will be the output of the following Java code?

```
class output
{
   public static void main(String args[])
   {
   StringBuffer sb=new StringBuffer("Hello");
   sb.replace(1,3,"Java");
   System.out.println(sb);
   }
}
```

   (a) Hello java (b) Hellojava
   (c) HJavalo (d) Hjava
14. What is the string contained in s after following lines of Java code?

```
StringBuffer s = new StringBuffer("Hello");
s.deleteCharAt(0);
```

   (a) Hell (b) ello
   (c) Hel (d) llo

15. What will be the output of the following Java program?

```
class output
{
   public static void main(String args[])
   {
      StringBuffer c = new StringBuffer("Hello");
      c.delete(0,2);
      System.out.println(c);
   }
}
```

(a) He (b) Hel
(c) lo (d) llo

16. Which of these is a mechanism for naming and visibility control of a class and its content?
(a) Object (b) Packages
(c) Interfaces (d) None of the mentioned

17. Which of this access specifies can be used for a class so that its members can be accessed by a different class in the same package?
(a) public (b) protected
(c) no modifier (d) All of the mentioned

18. Which method of the Class.class is used to determine the name of a class represented by the class object as a String?
(a) getClass() (b) intern()
(c) getName() (d) toString()

19. What are the uses of a Java Package?
(a) A package contains ready-to-use classes written for a specific purpose.
(b) Packages are easy to distribute your code. It is nothing but reusability. Instead of writing code afresh, you can take advantage of the existing classes of a package. Simply import it and use.
(c) Packages help in maintaining the code easily. Each sub-package may be maintained for more specific purposes. You can reuse the class names in sub-packages or other packages without name clash.
(d) All the above mentioned

20. The wrapper classes are part of the which package, that is imported by default into all Java programs?
(a) java.lang (b) java.awt
(c) java.io (d) java.util

21. Which is a basic unit of OOP?
(a) Object (b) Class
(c) Package (d) constructor

22. Which keyword is used for declaring a class?
    (a) new (b) class
    (c) static (d) final
23. Which is a collection of statements that are grouped together to perform a certain task?
    (a) State (b) Behavior
    (c) Method (d) Function
24. Following which access modifiers to set access (scope) levels for classes, variables, methods and constructors.
    (a) public (b) private
    (c) (d) All of mentioned
25. Which access modifier means we do not explicitly declare an access modifier for a class, method and so on?
    (a) default (b) public
    (c) private (d) protected
26. In which array elements store the location of the reference variables of the object?
    (a) Array of Objects (b) Array of Class
    (c) Array of Package (d) Array of String
27. Which is automatically called when an object is created?
    (a) Array (b) Class
    (c) Package (d) Constructor
28. The constructors which does not accept any argument is called as,
    (a) default (b) parameterized
    (c) Both (a) and (b) (d) None of mentioned
29. Which keyword is used a reference variable that refers to the current object?
    (a) final (b) static
    (c) this (d) new
30. A field that's declared with the static keyword known as,
    (a) static method (b) static block
    (c) static field (d) None of mentioned
31. Which class is the parent/root class of all the classes in Java by default?
    (a) Object (b) String
    (c) Scanner (d) BufferedReader
32. Which class is used when we want to manipulate the contents of the string?
    (a) Object (b) String
    (c) StringBuffer (d) Scanner
33. To formatting a string we use,
    (a) format() (b) printf()
    (c) Both (a) and (b) (d) None of mentioned

34. Which classes provide a way to use primitive data types (int, boolean, etc..) as objects?
    (a) Wrapper (b) String
    (c) StringBuffer (d) Scanner
35. In Java which is used to group related classes.
    (a) class (b) package
    (c) string (d) interface
36. Which package contains classes for primitive types, strings, math functions, threads, and exceptions?
    (a) java.lang (b) java.io
    (c) java.net (d) java.util

Answers

1. (c)	2. (b)	3. (a)	4. (b)	5. (a)	6. (d)	7. (a)	8. (d)	9. (b)	10. (d)
11. (b)	12. (c)	13. (c)	14. (b)	15. (d)	16. (b)	17. (d)	18. (c)	19. (d)	20. (a)
21. (a)	22. (b)	23. (c)	24. (d)	25. (a)	26. (a)	27. (d)	28. (a)	29. (c)	30. (c)
31. (a)	32. (c)	33. (c)	34. (a)	35. (b)	36. (a)				

Q.II Fill in the Blanks:

1. A class is also considered as an _____ factory.
2. An object is a _____ of a class.
3. A constructor is used when an _____ is created.
4. The keyword _____ is used to include a package in a program.
5. Wrapper class uses first letter in _____ case.
6. _____ method of class StringBuffer is used to concatenate the string representation to the end of invoking string.
7. _____ method of class StringBuffer is used to find the length of current character sequence.
8. The keyword used to declare a Java package is _____.
9. When importing a Package, the Class is actually importing _____ or interfaces from the package.
10. _____ method of object class can clone an object.
11. Under which _____ package is the string class encapsulated.
12. _____ keyword is used for accessing the features of a package.
13. String is a sequence of characters, but in java, string is an _____.
14. _____ is the return type of the hashCode() method in the Object class.
15. _____ keyword is used to refer current object of a class.
16. The _____ is called when an object of a class is created.
17. To declare any static method, we have to write a _____ keyword before the method name.

18. A class is a "blueprint" for creating _____.
19. _____ are used in Java, in-order to avoid name conflicts and to control access of class, interface and so on.
20. The string _____ method returns the formatted string by given locale, format and arguments.
21. The _____ classes in Java are used to convert primitive types (int, char, float, etc) into corresponding objects.
22. _____ modifiers are used to controls the access level for classes, methods and constructors.
23. To define a package in Java, we use the keyword _____.
24. A Java _____ is a collection of statements that are grouped together to perform a certain task.
25. String in Java is the _____ of characters.
26. We can make comparisons between two strings using the _____ method.
27. A constructor has the same name as its _____ and called for initializing objects when we create an object of that class.
28. In _____ access modifier the code is only accessible within the declared class.
29. If we declare any block as static, then such blocks are called _____ blocks in Java.
30. The _____ keyword refers to the current object in a method or constructor.
31. A _____ constructor is a constructor with no parameter.
32. We use the static block when we need to initialize the _____ variables.
33. Java has an _____ statement that allows us to import an entire package or use only certain classes and interfaces defined in the package.
34. A class is a blueprint for the object and _____ we create an object, we first need to define the class.
35. An object is called an instance of a _____.
36. When variables and methods are declared _____, they cannot be accessed outside of the class.
37. Constructor _____ in Java is a process of having more than one constructor with different parameters list.
38. When methods, variables, classes, and so on are declared _____, then we can access them from anywhere.
39. The _____ class is the root of the class hierarchy in Java.
40. The Java compiler automatically create a no-argument constructor during the execution of the program known as constructor is called constructor.
41. Java also allows us to create packages as per our need called as _____ -defined packages.
42. The strings in Java are treated as objects of type _____ class.

Answers

1. object	2. instance	3. object	4. import	5. upper case
6. append()	7. length()	8. package	9. classes	10. objectClone()
11. java.lang	12. import	13. object	14. int	15. this
16. constructor	17. static	18. objects	19. Packages	20. format()
21. wrapper	22. Access	23. package	24. method	25. sequence
26. equals()	27. class	28. private	29. static	30. this
31. Default	32. static	33. import	34. before	35. class
36. private	37. overloading	38. public	39. Object	40. default
41. user	42. String			

Q.III State True or False:

1. The string is immutable, but the StringBuffer is mutable.
2. Package names and directory structure are closely related.
3. The private method can be overridden as public method.
4. We call methods using this keyword.
5. We have multiple static blocks in our code.
6. A private member of a class cannot be accessed by the methods of the same class.
7. A protected member of a class can be accessed from its derived class.
8. The public members of class can be accessed by any code in the program can a class be declared with a protected modifier.
9. When object of class is declared, each object contains its own copy of static variables.
10. The static methods must only access static data.
11. Constructor can have a return type.
12. A package can be renamed without renaming the directory in which the classes are stored.
13. In Java, an object is created from a class.
14. The this keyword in Java is a reference variable that refers to the current object of a method or a constructor.
15. Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class.
16. A constructor in Java is similar to a method that is invoked when an object of the class is created.
17. We can create a class in Java using the class keyword.
18. Constructors are similar to methods and have the same name as the method.
19. The hashCode() method returns a hash code for the string.
20. Wrapper classes are used to provide a mechanism to ‘wrap’ or bind the values of primitive data types into an object.

21. Access modifiers provide access to members of a class within a Java program.
22. A Java access modifier specifies which classes can access a given class and its fields, methods, and constructors.
23. Before creating an array of objects, we must create an instance of the class by using the static keyword.
24. The Java Object getClass() method returns the class name of the object.
25. A package is simply a container that groups related types (Java classes, interfaces, enumerations, and annotations).
26. We can use the name of objects along with the . operator to access members of a class.
27. A method is a block of code that performs a specific task.
28. A string is a sequence of operators.
29. We can join two strings in Java using the concat() method.
30. The equals() method checks the content of strings while comparing them.
31. We can also create two or more constructors with different parameters called as constructors overloading.
32. Access modifiers are used to set the accessibility (visibility) of classes, interfaces, variables, methods, constructors.
33. A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructor with parameters).
34. The public access modifier in Java denotes that a variable or a method is private to the class and we cannot access them outside the class.
35. When methods and data members are declared protected, we can access them within the same package as well as from subclasses.
36. Java allows us to store objects in an array.
37. An class is any entity that has a state and behavior.
38. The protected access modifier in Java denotes that the variable or a method is accessible to its subclasses but private to all other classes outside the current package.
39. Declaring a class with the private access modifier would mean that no other classes can access it.
40. The 'this' keyword can be used to invoke current class constructor.
41. An array that conations class type elements are known as an array of objects which stores the reference variable of the object.

Answers

1. (T)	2. (T)	3. (F)	4. (T)	5. (T)	6. (F)	7. (T)	8. (T)	9. (F)	10. (F)
11. (F)	12. (F)	13. (T)	14. (T)	15. (T)	16. (T)	17. (T)	18. (F)	19. (T)	20. (T)
21. (T)	22. (T)	23. (F)	24. (T)	25. (T)	26. (T)	27. (T)	28. (F)	29. (T)	30. (T)
31. (T)	32. (T)	33. (T)	34. (F)	35. (T)	36. (T)	37. (F)	38. (T)	39. (T)	40. (T)
41. (T)									

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. Define is object?
2. Define class.
3. What is constructor?
4. Why static keyword is used?
5. Define package.
6. List types of constructors.
7. Give use of this keyword.
8. Define static method.
9. What is array of objects?
10. What is the purpose of getClass().
11. What is access specifier?
12. Define wrapper classes.
13. What is the use of StringBuffer class.
14. How to create our own classes?
15. What is the purpose of package keyword?
16. Which functions are used for formatting strings?

(B) Long Answer Questions:

1. With the help of example describe classes and objects.
2. Explain the significance of each of the following:
   (i) static
   (ii) public
   (iii) protected
   (iv) private.
3. What is a class? How to create it? Explain with example.
4. What are objects? How to create objects? Explain with example.
5. Define constructor. How to create it?
6. Explain default constructor with example.
7. What is parameter constructor? Give its syntax. Also explain with example.
8. When to declare a method or field static?
9. What is a package? How to create and import it?
10. Explain various types of access modifiers in detail.
11. Explain java API packages in detail.
12. Create a class circle, consider pi and radius. Calculate the area of a circle and display it.

13. Create a class distance with variable feet and inches, method get_val(), show_dist(). Use constructor. Create two objects and display their distance.
14. Write a short note on:
    (i) this keyword
    (ii) Object class
    (iii) Wrapper classes.
15. What is string? How to format a string? Describe with example.
16. How to accessing packages? Explain with example.
17. Explain the term wrapper classes in detail.
18. With the help of example describe how to create our own classes.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. What is 'this'? Give example. **[1 M]**

Ans. Refer to Section 2.7.

2. Give two ways to create string object with an example. **[5 M]**

Ans. Refer to Section 2.9.2.

October 2016

1. Can we overload main() method in Java? Justify. **[1 M]**

Ans. Refer to Section 2.6.

2. Explain any two methods of string class with example. **[2 M]**

Ans. Refer to Section 2.9.2.

3. What is the use of static methods and static variables? **[2 M]**

Ans. Refer to Section 2.8.3.

4. What is package? How to create a package? **[1 M]**

Ans. Refer to Section s 2.11 and 2.11.1.

April 2017

1. What is the use of this keyword in Java? **[1 M]**

Ans. Refer to Section 2.7.

2. What is wrapper class? Explain with example. **[5 M]**

Ans. Refer to Section 2.10.

3. Differentiate String and StringBuffer classes. **[2 M]**

Ans. Refer to Page 2.35.

4. What is Package? How to create a package? Give example. **[2 M]**

Ans. Refer to Sections 2.11 and 2.11.1.

October 2017

1. What are packages? **[1 M]**

Ans. Refer to Section 2.11.

2. Create two packages, pack1 contains two classes as student and course. Both classes have method to read corresponding Information. Pack2 contains class college with method accept(). Write a java program to display all information. **[5 M]**

Ans. Refer to Section 2.11.

3. List any two names of Wrapper class. **[2 M]**

Ans. Refer to Section 2.10.

April 2018

1. Which method is used to compare values of two string objects? **[1 M]**

Ans. Refer to Section 2.9.2.

2. How do we design, create and access a package in java? Discuss with suitable example. **[5 M]**

Ans. Refer to Sections 2.11.1 and 2.11.2.

3. What is difference between string and string buffer class? **[3 M]**

Ans. Refer to Page 2.35.

4. Explain the use of this keyword in java. **[2 M]**

Ans. Refer to Section 2.7.

5. List any two methods of object class. **[1 M]**

Ans. Refer to Section 2.9.1.

6. How do we design, create and access a package in Java? Discuss with suitable example. **[5 M]**

Ans. Refer to Sections 2.11.1 and 2.11.2.

October 2018

1. Write down syntax for equals() method. **[1 M]**

Ans. Refer to Section 2.9.1.

2. State the use of to String() method. **[1 M]**

Ans. Refer to Section 2.9.3.

3. Write a java program to create a package named student. Define class studentInfo with method to display information about student such as rollno, name, class and percentage. Create another class studentPer with method to find percentage of the student. Accept student details like rollno, name, class and marks of three subject from user. **[5 M]**

Ans. Refer to Section 2.11.

April 2019

1. What is package? How to create a package. Explain with example. **[5 M]**

Ans. Refer to Sections 2.11 and 2.11.1.

2. Enlist any two method of string with syntax. **[2 M]**

Ans. Refer to Section 2.9.2.

■■■

CHAPTER 3

Inheritance and Interface

Objectives...

- To understand Basic Concpets of Inheritance
- To study Basic Concpets in Interfaces
- To learn Abstract Class and Methods
- To learn Run-time Polymorphism using Interface

3.0 INTRODUCTION

- Inheritance can be defined as, the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.
- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
- An interface is one of the core concepts in Java and is used to achieve abstraction, polymorphism and multiple inheritances.
- In this chapter we will study inheritance, runtime polymorphism, interfaces, marker concept and so on.

3.1 INHERITANCE BASICS

- Object Oriented Programming (OOP) allows us to create new class based on a class that has been already defined using inheritance.
- Inheritance can be defined as, “the process where one object acquires the properties of another class”.
- Inheritance represents the 'IS-A relationship', also known as 'parent-child relationship'.
- In simple words, the mechanism of deriving a new class from an old class is called inheritance.
- A class that is derived from any another class is called a subclass or derived class or extended class or child class. The class from which a subclass is derived is called super class or base class or parent class.

- Fig. 3.1 shows a class called Vehicle, which has two subclasses called Two Wheeler and Four Wheeler.

Fig. 3.1

Advantages of Inheritance:

1. Code reusability through inheritance is increased.
2. Inheritance provides a clear model structure which is easy and simple to understand.
3. Code is easy to manage and divided into parent and child classes.
4. No need to write code from scratch. We can start coding with existing class.
5. Using inheritance we can very easily convert small system into large systems.
6. Good at representing the objects.

3.1.1 extends Keyword

- The keyword 'extends' is used to inherit the properties of a class through another class.
- The keyword extends indicates that, we are making a new class that derives from an existing class.
- The **syntax** is given below:

```
class subclassname extends superclassname
{
    //Body of the class …
}
```

- The keyword 'extends' specifies that the properties of superclassname are extended to subclassname. After this the subclass will contain all the methods of super class and it will add the members of its own.
- Remember, we can not extend a subclass from more than one super class. Direct multiple inheritance is not allowed in Java.

Program 3.1: A simple program for inheritance.

```
class Calculation
{
   int z;
   public void addition(int x, int y)
   {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }
   public void Subtraction(int x, int y)
   {
   z = x - y;
   System.out.println("The difference between the given numbers:"+z);
   }
}
public class My_Calculation extends Calculation
{
   public void multiplication(int x, int y)
   {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }
   public static void main(String args[])
   {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

Output:

```
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
```

- Program 3.1 shows an example of simple inheritance in which the 'Sub' class is inherited from 'Super' class. So the members of 'Super' class that is, 'i', 'j' and method 'show ()' are available in 'Sub' class.
- By creating the object of 'Sub' class we can access all these members. But the methods 'display()', 'sum()' and variable 'k' cannot accessed in 'Super' class. So 'Super' class object cannot use these members.

3.1.2 Types of Inheritance [April 16, 18]

- Fig. 3.2 shows various types of inheritance in Java.

Fig. 3.2: Types of Inheritance

1. Single Inheritance:

- When a class extends another one class only then we call it a single inheritance.
- In case of single inheritance there is only a sub class and it's parent class. It is also called simple inheritance or one level inheritance.
- Fig. 3.3 shows that class B extends only one class which is A. Here, A is a **parent class** of B and B would be a **child class** of A.

Fig. 3.3: Single Inheritance

- The Fig. 3.3 shows that class A is super class and class B is the sub class. Let us consider that we have class 'vehicle' which is inherited by class 'Taxi', (See Fig. 3.4).

Fig. 3.4: Example of Single Inheritance

Program 3.2: Program for single inheritance.

```
class Person
{
   String name;
   int age;
```

```
    Person(String n, int a)
    {
        name=n;
        age=a;
    }
    void display()
    {
        System.out.println("Name="+name);
        System.out.println("Age="+age);
    }
}
class Employee extends Person
{
    String emp_designation;
    float emp_salary;
    Employee(String p,int q,String r,float s)
    {
        super(p,q);
        emp_designation=r;
        emp_salary=s;
    }
    void display1()
    {
        display();
        System.out.println("Employee Designation="+emp_designation);
        System.out.println("Employee Salary="+emp_salary);
    }
    public static void main(String args[])
    {
        Employee e=new Employee("Akshay",18,"Manager",25000);
        e.display1();
    }
}
```

Output:

```
Name= Akshay
Age=18
Employee Designation= Manager
Employee Salary=25000
```

2. Multilevel Inheritance: **[April 16]**

- We can create any layers in the inheritance as we want, this is called Multilevel Inheritance. This hierarchy can be created at any level of inheritance.
- When a subclass is derived from a derived class then this mechanism is known as the Multilevel inheritance.
- The derived class is called the subclass or child class for it's parent class and this parent class works as the child class for it's just above (parent) class.
- The general **form/syntax** is,

```
class A
{
    ......
    ......
    ......
}
class B extends A
{
    ......
    ......
    ......
}
class C extends B
{
    ......
    ......
    ......
}
```

- Class A may be the superclass for Class B and Class B may be the superclass for Class C and Class C may be the superclass for Class D.
- In above case, Class B inherites all properties of Class A, Class C inherites all properties of Class B and Class D inherites all properties of Class C.

Fig. 3.5: Representation of Multilevel Inheritance

Program 3.3: Program for multilevel inheritance.

```
import java.io.*;
class Account
{
   String cust_name;
   int acc_no;
   Account(String n, int no)
   {
      acc_no=no;
      cust_name=n;
   }
   void display()
   {
      System.out.println("Customer name = "+cust_name);
      System.out.println("Account number = "+acc_no);
   }
}
class Saving_acc extends Account
{
   int saving_bal;
   int min_bal;
   Saving_acc(String n, int no, int sbal, int mbal)
   {
      super(n,no);
      saving_bal=sbal;
      min_bal=mbal;
   }
   void output()
   {
      display();
      System.out.println("Saving balance ="+saving_bal);
      System.out.println("Minimum balance = "+ min_bal);
   }
}
class Acc_detail extends Saving_acc
{
   int deposit, withdrawal;
   Acc_detail(String n, int no, int sbal, int mbal, int dep, int wd)
   {
      super(n, no, sbal, mbal);
      deposit=dep;
      withdrawal=wd;
   }
```

```
    void final_result()
    {
        output();
        System.out.println("Deposit amount = "+ deposit);
        System.out.println("Withdrawal amount = "+ withdrawal);
    }
}
class sample
{
    public static void main(String args[])
    {
        Acc_detail a=new Acc_detail("ABC",1803,10000,500,1000,2000);
        a.final_result();
    }
}
```

Output:

```
Customer name = ABC
Account number = 1803
Saving balance = 10000
Minimum balance = 500
Deposit amount = 1000
Withdrawal amount = 2000
```

3. Hierarchical Inheritance:

- When a class has more than one child classes (sub classes) have the same parent class then such kind of inheritance is known as Hierarchical inheritance.
- In case of hierarchical inheritance we derive more than one subclass from a single super class.
- In below example class B, C and D inherits the same class A. A is parent class (or base class) of B, C and D.

Fig. 3.6: Hierarchical Inheritance

- Fig. 3.7 represents hierarchical inheritance. Here, class student is having three child Medical, Engineering, Non science at first level, where class Engineering. itself is having some child at the second level. Hierarchical inheritance is basically the combination of more then one type of inheritance.

Fig. 3.7: Example of Hierarchical Inheritance

Program 3.4: Program for hierarchical inheritance.

```
class A
{
   public void methodA()
   {
      System.out.println("method of Class A");
   }
}
class B extends A
{
   public void methodB()
   {
      System.out.println("method of Class B");
   }
}
class C extends A
{
   public void methodC()
   {
      System.out.println("method of Class C");
   }
}
class D extends A
{
   public void methodD()
   {
      System.out.println("method of Class D");
   }
}
```

```
class MyClass
{
   public static void main(String args[])
   {
      B obj1 = new B();
      C obj2 = new C();
      D obj3 = new D();
      obj1.methodA()
      obj1.methodB();
      obj2.methodA()
      obj2.methodC();
      obj3.methodA()
      obj3.methodD();
   }
}
```

Output:

```
method of Class A
method of Class B
method of Class A
method of Class C
method of Class A
method of Class D
```

3.2 SUPERCLASS, SUBCLASS AND USE OF 'super' KEYWORD

- Inheritance is one of the key features of OOP that allows us to create a new class from an existing class.
- The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).
- The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

3.2.1 Superclass and Subclass

- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- The idea of inheritance is simple but powerful i.e., when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.

- A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).
- Fig. 3.8 (a) shows if class Y is a subclass of class X, we also say that class X is a superclass of class Y.
- A subclass can add to the structure and behavior that it inherits. It can also replace or modify inherited behavior (though not inherited structure).
- The relationship between subclass and superclass is shown in Fig. 3.8 (a).

Fig. 3.8: The Relationship between Subclass and Superclass

- In Java, to create a class named "Y" as a subclass of a class named "X", we would write,

```
class Y extends X
{
    .  // additions to, and modifications of,
    .  // stuff inherited from class X
}
```

- Several classes can be declared as subclasses of the same superclass. The subclasses, which might be referred to as "sibling classes," share some structures and behaviors -- namely, the ones they inherit from their common superclass.
- The superclass expresses these shared structures and behaviors. In the Fig. 3.8 (b) classes Y, Z and A are sibling classes.
- Inheritance can also extend over several "generations" of classes. This is shown in the Fig. 3.8 (b), where class B is a subclass of class A which is itself a subclass of class X.
- In this case, class B is considered to be a subclass of class X, even though it is not a direct subclass. This whole set of classes forms a small class hierarchy.

For example:

- Let's look at an example of Vehicles. Suppose that a program has to deal with motor vehicles, including Car, Truck and Motorcycle. The program could use a class named to represent all types of vehicles.
- Since cars, trucks, and motorcycles are types of vehicles, they would be represented by subclasses of the Vehicle class, as shown in Fig. 3.9.

Fig. 3.9: Subclasses of the Vehicle Class

- The Vehicle class would include instance variables such as registrationNumber and owner and instance methods such as transferOwnership(). These are variables and methods common to all vehicles.
- The three subclasses of Vehicle i.e., Car, Truck, and Motorcycle -- could then be used to hold variables and methods specific to particular types of vehicles.
- The Car class might add an instance variable numberOfDoors, the Truck class might have numberOfAxles, and the Motorcycle class could have a boolean variable hasSidecar.
- The declarations of these classes in a Java program would look:

```
class Vehicle
{
   int registrationNumber;
   Person owner;  // (Assuming that a Person class has been defined!)
   void transferOwnership(Person newOwner)
   {
       .........
   }
   .........
}
class Car extends Vehicle
{
   int numberOfDoors;
   .........
}
class Truck extends Vehicle
{
   int numberOfAxles;
   .........
}
class Motorcycle extends Vehicle
{
   boolean hasSidecar;
   .........
}
```

3.2.2 Use of 'super' Keyword [April 16, Oct. 16, 17]

- A subclass inherits the accessible data fields and methods from its superclass, but the constructors of the superclass are not inherited in the subclass.
- They can only be invoked from constructors of the subclass(es) using the keyword 'super'.
- The 'super' keyword is used by subclass to refer its immediate superclass. There are two different forms of using super:
  1. 'super' calls the superclass constructor.
  2. 'super' can access members of the superclass those have been hidden by members of a subclass.
- Let us see above forms of keyword super in detail.

1. 'super' class Superclass Constructor:

- In the constructor of subclass, the keyword super can be used to invoke the constructor method of the superclass.
- A subclass can call the constructor defined by super class by using the following form of the 'super' keyword.

```
class subclass extends superclass
{
   subclass(args)
   {
      super(arg_list);
      //Statements;
   }
      ....................
}
```

- The use of super() in this form must follow the following conditions:
  (i) The super() must always be the first statement in subclass constructor.
  (ii) The order and type of parameters specified in super() must matches with the parameters in constructor of superclass.

Program 3.5: Use of super to access super class constructor.

```
class Primary
{
   int cal;                  //declaration1
   Primary(int a)
   {
      cal = a;
   }
```

```
    void show()
    {
        System.out.println("Super class cal: "+cal);
    }
}
class Secondary extends Primary
{
    int cal;                    //declaration2
    Secondary(int x,int y)      //statement1
    {
        super(y);               //statement2
        cal = x;                //statement3
    }
    void display()
    {
        System.out.println("Sub class cal: "+cal);
    }
}
class SuperUse2
{
    public static void main(String args[])
    {
        Secondary s = new Secondary(15,22);
        s.show();
        s.display();
    }
}
```

Output:

```
Super class cal: 22
Sub class cal: 15
```

- The super class ‘Primary’ has been added with a constructor of one parameter. This constructor has been called in 'sub class’ constructor in statement2 by passing one parameter to super().
- If the super class contains more number of constructors all that can be called using super keyword in 'sub class’ constructor. Statement that contains ‘super’ must be a first statement in the 'sub class’ constructor.

2. 'super' Access Member of Superclass:

- Any member of immediate superclass can be referred in subclass using super keyword.

- The usage has the following **form/syntax:**

  `super.member_name`
- This use of super keyword is somewhat like this. But difference is that super always refers to the superclass of the class in which it is used, whereas this refers to the members of that class itself.
- This form of super keyword is most useful when member in subclass hides the member of superclass due to similar names.
- Consider the following Program 3.6, the variable name n is used in superclass as well as in subclass. To access a variable n of superclass in subclass, we use the super keyword.

Program 3.6: Program for super refers member of superlcass.

```
class SuperClass
{
int n;
}
class Subclass extends SuperClass
{
    int n;   // This hides n of superclass
   Subclass (int x, int y)
   {
       if (x > y)
      {
          super.n = x; n = y;
       }
       else
       {
          super.n = y; n = x;
       }
}
void display()
{
      System.out.println("n is SuperClass=" + super.n);
      System.out.println("n is SubClass=" + n);
}
}
class TestSuper2
{
public static void main(String args[])
{
//create object of subclass
Subclass myobj = new Subclass (10,4);
```

```
        //call display method
        myobj.display();
        }
    }
```

Output:

```
n in SuperClass = 10
n in SubClass = 4
```

- In similar way, super keyword can also be used to call methods of superclass that are hidden by subclass due to same names. This is called as Method Overriding.

3.3 METHOD OVERRIDING AND RUNTIME POLYMORPHISM

- If the same method is defined in both the superclass and the subclass, then the method of the subclass class overrides the method of the superclass. This is known as method overriding. **[April 17, 18]**
- Runtime polymorphism is a process in which a call to an overridden method is resolved at runtime rather than compile-time. Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding.
- Method overriding is an example of runtime polymorphism. In method overriding, a subclass overrides a method with the same signature as that of in its superclass.

Usage of Method Overriding:

1. Method overriding is used to provide specific implementation of a method that is already provided by its super class.
2. Method overriding is used for run-time polymorphism.

- In short, redefining a super class method in a sub class is called Method Overriding.

Rules for Method Overriding:

1. The method signature i.e., method name, parameter list and return type have to match exactly.
2. The overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa.

Example:

- Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks.
- For example, HDFC, ICICI and SBI banks could provide 8%, 7% and 9% rate of interest respectively.

Fig. 3.10

Program 3.7: Program for method overriding.

```
class Bank
{
   int getRateOfInterest()
   {
      return 0;
   }
}
class HDFC extends Bank
{
   int getRateOfInterest(){return 8;}
}
class ICICI extends Bank
{
   int getRateOfInterest(){return 7;}
}
class SBI extends Bank
{
   int getRateOfInterest(){return 9;}
}
class Test2
{
   public static void main(String args[])
   {
      HDFC s=new HDFC();
      ICICI i=new ICICI();
      SBI a=new SBI();
     System.out.println("HDFC Rate of Interest: "+s.getRateOfInterest());
```

```
        System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
        System.out.println("SBI Rate of Interest: "+a.getRateOfInterest());
      }
   }
```

Output:

```
HDFC Rate of Interest: 8
ICICI Rate of Interest: 7
SBI Rate of Interest: 9
```

Use the super keyword: When invoking a superclass version of an overridden method the super keyword is used.

Program 3.8: Program using the super keyword.

```
class Animal
{
   public void move()
   {
      System.out.println("Animals can move …");
   }
}
class Cat extends Animal
{
   public void move()
   {
       super.move(); // invokes the super class method
       System.out.println("Cats can Walk and Run");
   }
}
public class TestCat
{
   public static void main(String args[])
   {
      Animal b = new Cat(); // Animal reference but Cat object
      b.move(); //Runs the method in Cat class
   }
}
```

Output:

```
Animals can move
Cats can Walk and Run
```

3.4 USAGE OF 'final' KEYWORD [April 17, 19, Oct. 17]

- The "final" is a keyword in Java which generally means, cannot be changed once created.
- The "final" keyword when declared with variables, methods and classes specifically means:
  1. The final variables cannot be modified.
  2. The final method cannot be overridden.
  3. The final class cannot be inherited or extended.
- The keyword "final" is used to prevent method overriding. This is possibly done by specifying 'final' as a modifier at the start of its declaration.
- The method whenever is declared as final, it cannot be overridden.

3.4.1 'final' with Variables

- If any variable in Java is declare using 'final' keyword then they are known as final variable.

  Syntax: `final datatype variableName;`
- When a variable is declared final, it is a constant which will not and cannot change.

  Example: `final PI=3.14;`
- The value of PI is declared as 'final' so it cannot be changed in the course of the program.

Program 3.9: Program for 'final' with variables.

```
class FinalVar
{
   static
   {
      int x=10;
      final int y=20;
      System.out.println("x is:"+x);
      System.out.println("y is:"+y);
      x=30;
      y=40;
      System.out.println("x is:"+x);
      System.out.println("y is:"+y);
   }
}
```

Output:

```
Cannot assign a value to final variable y
```

3.4.2 'final' with Methods [April 18]

- We can also declare that methods are final. A method that is declared final cannot be overridden in a subclass.
- If method of any class is defined using final keyword then they are known as final methods in Java because they cannot be redefined.
- The syntax is simple, just put the keyword 'final' after the access specifier and before the return type.

  Syntax:
  ```
  access specifier final returntype methodname(arguments)
  {
     //method body……
  }
  ```
- When we try to override method that has been declared final it will generate compile time error.

Program 3.10: Program for final with methods.

```
class A
{
   final void meth()         //Final method
   {
      System.out.println("This is a final method.");
   }
}
class B extends A
{
   void meth()  // ERROR! Can't override.
   {
      System.out.println("Illegal!");
   }
}
```

3.4.3 'final' with Classes

- If we want the class not be sub-classed (or extended) by any other class, declare it final.
- In Java, if we define any class using 'final' keyword then this type of class is known as final class in Java.
- Classes declared final can not be extended. That is, any class can use the methods of a final class by creating an object of the final class and call the methods with the object(final class object).

Syntax:

```
final class ClassName
{
   //Members
}
```

Example:

```
final class finalclass
{
   //Definitions of members …
}
class errorclass extends finalclass
{
   //Definitions of members …
}
```

- The above program segment contains error since we tried to extend a final class.

Program 3.11: Program for 'final' with classes.

```
import java.io.*;
final class Demo1
{
   public void display()
   {
      System.out.println("hi");
   }
}
public class Demo3 extends Demo1
{
   public static void main(String args[])
   {
      Demo1 d=new Demo1();
      d.display();
   }
}
```

Output:

```
Cannot inherit from final Demo1
```

3.5 USAGE OF ABSTRACT CLASSES AND ABSTRACT METHODS

- Abstraction is a process of hiding the implementation details and showing only functionality to the user. Abstraction lets us focus on what the object does instead of how it does it. **[Oct. 17]**

- The methods which have only declaration and no method body in the base class are called abstract method.
- The abstract methods are declared using the keyword "abstract". The method declaration should end with semicolon (;).
- The abstract methods should be defined in the subclass. Abstract method can never be final and static.
- The keyword "abstract" is used while declaring abstract method but not at the time of defining.
  Syntax: `abstract <return_type> <method_name>(arguments);`
  Example: `abstract void area();`
- The base class in which abstract methods are declared is called as abstract class. The "abstract" keyword is used while declaring abstract class.
  Syntax:
  ```
  abstract class <class_name>
  {
     // body of class…
  }
  ```
- Abstract class can have simple methods as well as abstract methods. Abstract methods are those methods which are preceded by abstract keyword and user will declare them in abstract class the definition of abstract method will be carried out in sub class.
- Following table depicts the difference between class and abstract class:

Sr. No.	Class	Abstract Class
1.	Object of class can be created.	Object of abstract class cannot be created.
2.	It contains non abstract methods (methods with their definitions).	It can have abstract and non abstract methods.
3.	Extension of class is not mandatory.	Extension of abstract class is compulsory.

4.	**Declaration:**
	`class class_name`
	`{`
	`}`
		Declaration:
		`abstract class class_name`
		`{`
		`}`

Program 3.12: Program abstract classes and methods.

```
abstract class shape            // Abstract class
{
   double pi=3.14;
   abstract void area();        // Abstract method
   void display()
   {
      System.out.println("\n Non_abstract method of Class shape");
   }
}
```

```
class rectangle extends shape  // Sub class
{
   int l,b;
   rectangle(int x,int y)
   {
      l=x;
      b=y;
   }
   void area()                // Implementing abstract method
   {
      System.out.println("\n Area of Rectangle");
      System.out.println("Length="+ l);
      System.out.println("Breadth="+ b);
      System.out.println("Area  ="+ (l*b));
   }
}
class circle extends shape     // Sub class
{
   double r;
   circle(double x)
   {
      r=x;
   }
   void area()                // Implementing abstract method
   {
      System.out.println("\n Area of Circle");
      System.out.println("Radius="+ r);
      System.out.println("Area  ="+ (pi*r*r));
   }
}
/*  Main class */
class abst
{
   public static void main(String args[])
   {
      shape s;
```

```
        rectangle r=new rectangle(10,5);
        s=r;                    // Shape reference rectangle
        s.area();
        circle c=new circle(2.5);
        c.area();
    }
}
```

Output:

```
Area of Rectangle
Length=10
Breadth=5
Area=50
Area of Circle
Radius=2.5
Area=19.625
```

- In above Program 3.12, shape is the abstract class, its implementation is provided by the Rectangle and Circle classes.
- Mostly, we don't know about the implementation class, (i.e. hidden to the end user) and object of the implementation class is provided by the factory method.
- A factory method is the method that returns the instance of the class. We will learn about the factory method later.
- In this example, if we create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

Program 3.13:

```
abstract class Shape
{
    abstract void draw();
}
class Rectangle extends Shape
{
    void draw()
    {
        System.out.println("Drawing rectangle");
    }
}
class Circle1 extends Shape
{
```

```
    void draw()
    {
        System.out.println("Drawing circle");
    }
}
class TestAbstraction1
{
    public static void main(String args[])
    {
        Shape s=new Circle1();
        s.draw();
    }
}
```

Output:

```
Drawing circle
```

3.6 INTERFACE [Oct. 16]

- An interface in Java is a blueprint of a class. It has static constants and abstract methods only.
- An interface is a way of describing what classes should do, without specifying how they should do it.
- There are mainly following reasons to use interface:
  1. It is used to achieve fully abstraction.
  2. By interface, we can support the functionality of multiple inheritance.
- A class can implement more than one interface. In Java, an interface is not a class but a set of requirements for the class that we want to conform to the interface.
- All the methods of an interface are by default public. So, it is not required to use the keyword public when declaring a method in an interface.
- Interfaces can also have more than one method. Interfaces can also define constants but do not implement methods.

3.6.1 Defining Interfaces

- To support the concept of multiple inheritance, Java provides the alternate approach known as Interface.
- Java interface also represents IS-A relationship. Interface cannot be instantiated just like abstract class.
- Interface is similar to class which is collection of public, static, final, variables (constants) and abstract methods.
- An interface is defined like a class. The ‘interface’ keyword is used to declare an interface.

- The general **form/syntax** of an interface is as follows:

```
access_specifier interface InterfaceName
{
   Return-type Method-1(Parameters);
   Return-type Method-2(Parameters);
   Type variable-1=value;
   Type variable-2=value;
   .
   .
   .
   .
   .
   Return-type Method-N(Parameters);
   Type variable-N=value;
}
```

- Following is the definition of an interface containing two constants and four abstract methods.

```
interface conversions
{
   double GM_TO_KG = 1000;
   double CM_TO_FT = 30;
   double gmtokg (double gm);
   double cmtoft (double cm);
   double kgtogm (double kg);
   double fttocm (double ft);
}
```

Difference between Class and Interface:

Sr. No.	Class	Interface
1.	It can contain abstract methods along with method definitions.	It can contain only method declarations and not method definitions.
2.	Classes are extended by another class.	Interfaces are implemented by classes.
3.	Methods of objects can be called using the object of the class.	Methods of interfaces should be defined in the class which implements the interface.
4.	The variables of the class can be final/not final.	The variables of the interfaces are by default final.

contd. ...

5.	Object of class can be created.	Object of interface cannot be created.
6.	A class can extend only one class. **Example:** `class A extends B`	An class can implement more than one interface. **Example:** `class A implements C, D`
7.	Multiple inheritance is not possible with classes.	Interface was introduced for the concept of multiple inheritance.

8.	`class example`
	`{`
	`int x;`
	`void method1()`
	`{`
	`body`
	`}`
	`void method2()`
	`{`
	`body`
	`}`
	`}`
		`interface example`
		`{`
		`Final int x =5;`
		`void method1();`
		`void method2();`
		`}`

3.6.2 Implementation of Interface

- Once, an interface has been defined, one or more classes can implement that interface.
- To implement an interface, include the implements clause in a class definition, and then create the methods defined by the interface.
- A class uses the 'implements' keyword to implement an interface. The general **form/syntax** of a class that includes the 'implements' keyword looks like this:

```
access_specifier class class_name implements interface_name
{
    // class body……
}
```

Program 3.14: Program of interface implementation.

```
interface conversions
{
    double GM_TO_KG = 1000;
    double gmtokg (double gm);
    double kgtogm (double kg);
}
```

```
class convert implements conversions
{
   public double gmtokg (double gm)
   {
      return gm/GM_TO_KG;
   }
   pubic double kgtogm(double kg)
   {
      return kg*GM_TO_KG;
   }
}
class ImplIface1
{
   public static void main(String args[])
   {
      Convert ConvObj = new Convert();
      Conversions c;
      c = Convobj;
      System.out.println("2000 gm =" + C.gmtokg(2000) + " kg ");
      System.out.println("50 kg = " + c.kgtogm(50) + " gm ");
   }
}
```

Output:

```
2000 gm = 2.0 kg
50 kg = 50000.0 gm
```

3.6.3 Extending Interfaces

- An interface can extend another interface, similarly to the way that a class can extend another class.
- The 'extends' keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
- When any class implements an extended interface, it must implement all the methods defined within the chain of inherited interfaces.
- Consider the following Program 3.15, an interface emp_show inherits all the members of an interface emp_info.
- The class emp is implementing an interface emp_show, it must define the code of all the methods declared in emp_info as well as emp_show.

Program 3.15: Program extending Interfaces.

```
interface emp_info
{
   int empid = 10;
   String empname = "Patil Dinesh";
   String empaddr = "Gangapur Rd., Nashik";
   void disp1();
}
interface emp_show extends emp_info
{
   void disp2();
}
class emp implements emp_show
{
   public void disp1()
   {
      System.out.println("Empid="+empid);
      System.out.println("Name="+empname);
   }
   public void disp2()
   {
      System.out.println("Name="+ empname);
      System.out.println("Address="+ empaddr);
   }
}
Class ImplIface3
{
   public static void main(String args[])
   {
      emp e = new emp();
      System.out.println("From Method disp1);
      e.disp1();
      System.out.println();
      System.out.println("From Method disp2);
      e.disp2();
   }
}
```

Output:

```
From method disp1
Empid = 10;
Name = Patil Dinesh;
From method disp2
Name = Patil Dinesh;
Address = Gangapur Rd, Nashik
```

Extending Multiple Interfaces:

- A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.
- The 'extends' keyword is used once and the parent interfaces are declared in a comma-separated list.
- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

Fig. 3.11: Multiple Inheritance in Java

Program 3.16: Program to implementation of multiple inheritance.

```
class employee
{
   String ename;
   void get_ename(String name)
   {
      ename = name;
   }
   void print_ename()
   {
      System.out.println("Name = " + ename);
   }
}
class gross_sal extends employee
{
   float basic, da, hra, gsal;
   void calc_gross_sal(int b, int dapercent, int hrpercent)
   {
      basic = b;   //calculate DA & HRA
      da = basic* dapercent/100;
      hra = basic*hrpercent/100;
      gsal = basic + da + hra;
   }
```

```
    void print_gross_sal()
    {
        System.out.println("Gross sal = " + gsal);
    }
}
interface taxes
{
    float ptax = 200 F  //P.Tax Rs. 200/-
    float itaxpercent = 20F  //itax 20% on gsal
    float calc_taxes();
}
class salary extends gross-sal implements taxes
{
    public float calc_taxes()
    {            //Method from interface
        return ptax + (gsal * itaxpercent/100);
    }
    void display()
    {      //Method not from interface
        float tax = calc_taxes();
        float nsal = gsal_tax;
        print_ename();
        print_gross_sal();
        System.out.println("taxes=" + tax);
        System.out.println("Net Sal = " + nsal);
    }
}
class ImplIface6
{
    pubic static void main(String args[])
    {
        //create an object of class salary
        salary empsal = new salary();
        empsal.get_ename("Pawar Manoj');
        empsal.calc_gross_sal(9100,50,10);
        empsal.display();
    }
}
```

Output:

```
Name = Pawar Manoj
Gross sal = 14560.0
taxes = 3112.0
Net Sal = 11448.0
```

3.6.4 Nested Interfaces [April 17]

- An interface declared within another interface or class is known as Nested Interface.
- 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 cannot be accessed directly.

Points to Remember for Nested Interfaces:

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitely.

- The **syntax** of nested interface which is declared within the interface:

```
interface interface_name
{
    ...
    interface nested_interface_name
    {
        ...
    }
}
```

- The **syntax** of nested interface which is declared within the class:

```
class class_name
{
    ...
    interface nested_interface_name
    {
        ...
    }
}
```

Program 3.17: Program declare the Nested Interface and how we can access it.

```
interface ShowableExe
{
    void show();
    interface Message
    {
        void msg();
    }
}
```

```
class TestNestedInterface1 implements ShowableExe.Message
{
   public void msg()
   {
      System.out.println("Program for Nested Interface");
   }
   public static void main(String args[])
   {
      Showable.Message message=new TestNestedInterface1();
                //Upcasting Here …
      message.msg();
   }
}
```

Output:

```
Program for Nested Interface
```

3.7 RUN-TIME POLYMORPHISM USING INTERFACE

- A process in which a call to an overridden method is resolved at run-time rather than compile time is known as run-time polymorphism.
- In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
- When reference variable of Parent class refers to the object of Child class, it is known as upcasting.
- Run-time polymorphism is also called dynamic method dispatch.
- In Program 3.18, we are creating two classes Bike and Platina. Platina class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class.
- Since, it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at run-time.
- Since, method invocation is determined by the JVM not compiler, it is known as Run-time Polymorphism.

Program 3.18: Program for run-time polymorphism.

```
class Bike
{
   void run()
   {
      System.out.println("running");
   }
}
```

```
class Platina extends Bike
{
   void run(){System.out.println("Running safely with 80km");}
   public static void main(String args[])
   {
       Bike b = new Platina();//upcasting
       b.run();
   }
}
```

Output:

```
Running safely with 80 km.
```

- Runtime Polymorphism can be categorized in following two parts:

1. **Runtime Polymorphism with Data Member:**

- The method is overridden but not the data members, so run-time polymorphism cannot be done with data members.
- In the example given below, we created a Student class and Monitor class. Both the classes have data members, in other words name and roll number.
- We are accessing the data members by the reference variable of the parent class that refers to the subclass object.
- Since, we are accessing the data members of the subclass that are not overridden, the data members of the parent class will always be invoked.

Program 3.19:

```
class Student
{
   String name="Amar";
}
class Monitor extends Student
{
   String name="Yogita";
   public static void main(String args[])
   {
      Student s=new Monitor();
      System.out.println(s.name);
   }
}
```

Output:

```
Amar
```

2. Runtime Polymorphism with Multilevel Inheritance:

Program 3.20:

```
class Electronics
{
    void function()
    {
        System.out.println("Made for various purpose");
    }
}
class Phone extends Electronics
{
    void function()
    {
        System.out.println("Display,Voice");
    }
}
class Computer extends Phone
{
   void function()
   {
      System.out.println("Calculation,Internetworking");
   }
   public static void main(String args[])
   {
      Electronics e1=new Electronics();
      Electronics e2=new Phone();
      Electronics e3=new Computer();
      e1.function();
      e2.function();
      e3.function();
   }
}
```

Output:

```
Made for various purpose
Display, Voice
Calculation, Internetworking
```

3.8 CONCEPT OF MARKER AND FUNCTIONAL INTERFACES

- In this section we will study marker interface and functional interface in Java. A marker interface is an empty interface (no field or methods). Examples of marker interface are Serializable, Cloneable etc.
- A functional interface is an interface that contains only one abstract method. Runnable, ActionListener, Comparable are some of the examples of functional interfaces.

Marker Interface:

- An interface that does not contain methods, fields, and constants is known as marker interface.
- In other words, an empty interface is known as marker interface or tag interface. It delivers the run-time type information about an object. It is the reason that the JVM and compiler have additional information about an object.
- The Serializable and Cloneable interfaces are the example of marker interface. In short, it indicates a signal or command to the JVM.
- The declaration of marker interface is the same as interface in Java but the interface must be empty. For example:

```
public interface Serializable
{
}
```

- Marker interface is used as a tag that informs the Java compiler by a message so that it can add some special behavior to the class implementing it.
- Java marker interface are useful if we have information about the class and that information never changes, in such cases, we use marker interface represent to represent the same. Implementing an empty interface tells the compiler to do some operations.
- In Java, built-in marker interfaces are the interfaces that are already present in the JDK and ready to use. These interfaces includes:
  1. **Cleanable interface** in Java is also a marker interface that belongs to java.lang package. It generates replica (copy) of an object with different name. We can implement the interface in the class of which class object to be cloned. It indicates the **clone()** method of the Object class.
  2. **Serializable interface** is a marker interface in Java that is defined in the java.io package. If we want to make the class serializable, we must implement the Serializable interface. If a class implements the Serializable interface, we can serialize or deserialize the state of an object of that class.
  3. **Remote interface** is a marker interface that belong to java.rmi package. It marks an object as remote that can be accessed from another machine (host). We must implement the Remote interface if we want to make an object as remote. It identifies the interfaces whose methods can be invoked from a non-local JVM. Any remote object must implement the interface directly or indirectly.

Program 3.21: Program for Serializable interface.

```
import java.util.Scanner;
public class Product implements Cloneable
{
   int pid;
   String pname;
   double pcost;
   //Product class constructor
   public Product (int pid, String pname, double pcost)
   {
      this.pid = pid;
      this.pname = pname;
      this.pcost = pcost;
   }
   //method that prints the detail on the console
      public void showDetail()
      {
         System.out.println("Product ID: "+pid);
         System.out.println("Product Name: "+pname);
         System.out.println("Product Cost: "+pcost);
      }
      public static void main
                         (String args[]) throws CloneNotSupportedException
      {
         //reading values of the product from the user
         Scanner sc = new Scanner(System.in);
         System.out.print("Enter product ID: ");
         int pid = sc.nextInt();
         System.out.print("Enter product name: ");
         String pname = sc.next();
         System.out.print("Enter product Cost: ");
         double pcost = sc.nextDouble();
         System.out.println("-------Product Detail--------");
         Product p1 = new Product(pid, pname, pcost);
      // cloning the object of the Product class using the clone() method
         Product p2 = (Product) p1.clone();
```

```
        // invoking the method to print detail
            p2.showDetail();
    }
}
```

Output:

```
Enter product ID: 139872
Enter product name: Printer
Enter product Cost: 3459.67
-------Product Detail--------
Product ID: 139872
Product Name: Printer
Product Cost: 3459.67
```

Functional Interface:

- An interface that contains exactly one abstract method is known as functional interface.
- It can have any number of default, static methods but can contain only one abstract method. It can also declare methods of object class.
- Functional interface is also known as Single Abstract Method Interfaces or SAM interfaces. It is a new feature in Java, which helps to achieve functional programming approach.
- The @FunctionalInterface annotation is used to ensure that the functional interface can't have more than one abstract method. it is not mandatory to use this annotation.

Program 3.22: Program for functional interface.

```
@FunctionalInterface
interface sayable
{
    void say(String msg);
}
public class FunctionalInterfaceExample implements sayable
{
    public void say(String msg)
    {
        System.out.println(msg);
    }
    public static void main(String[] args)
    {
        FunctionalInterfaceExample fie = new FunctionalInterfaceExample();
                                                        fie.say("Hello there");
    }
}
```

Output:

```
Hello there
```

- A functional interface can have methods of object class. See in the following example.

```
@FunctionalInterface
interface sayable
{
   void say(String msg);   // abstract method
   // It can contain any number of Object class methods.
   int hashCode();
   String toString();
   boolean equals(Object obj);
}
public class FunctionalInterfaceExample2 implements sayable
{
   public void say(String msg)
   {
      System.out.println(msg);
   }
   public static void main(String[] args)
   {
      FunctionalInterfaceExample2 fie = new FunctionalInterfaceExample2();
                                                        fie.say("Hello there");
   }
}
```

Output:

```
Hello there
```

PROGRAMS

Program 1: Program to display the numbers of parent class and child class demonstrate single inheritance.

```
import java.util.Scanner;
class A
{
   int no1;
   void readA()
    {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the Number First : ");
        no1= in.nextInt();
    }
}
```

```
class B extends A
{
    int no2;
    void readB()
    {
       readA();
       Scanner in = new Scanner(System.in);
       System.out.print("Enter the Number Second : ");
       no2= in.nextInt();
    }
    void display()
    {
       System.out.println("First Number = "+no1);
       System.out.println("Second Number = "+no2);
    }
}
public class ExsingleInh
{
   public static void main(String[] args)
   {
      B b = new B();
      b.readB();
      b.display();
   }
}
```

Output:

```
Enter the Number First : 5
Enter the Number Second : 10
First Number = 5
Second Number = 10
```

Program 2: Program to display College and department name using single inheritance and super keyword.

```
class College
{
   String cname;
   public College(String m)
   {
      cname = m;
   }
}
```

```
public class Department extends College
{
   String dname;
   public Department(String x1, String x2)
   {
      super(x1);
            //passing argument to parent class constructor this.dname = x2;
   }
   public void display()
   {
      System.out.println(super.cname+" and "+dname);
   }
   public static void main(String[] args)
   {
      Department c = new Department("Pune University","Computer Science");
      c.display();
   }
}
```

Output:

```
Pune University and Computer Science
```

Program 3: Program for multilevel inheritance.

```
class AA
{
   public void readA()
   {
      System.out.println("Hello Class A");
   }
}
class BB extends AA
{
   public void readB()
   {
      readA();
      System.out.println("Hello Class B");
   }
}
class CC extends BB
{
   public void readC()
   {
      readB();
      System.out.println("Hello Class C");
   }
}
```

```
class Exmultilevelinh
{
   public static void main(String args[])
   {
      CC obj = new CC();
      obj.readC();
   }
}
```

Output:

```
Hello Class A
Hello Class B
Hello Class C
```

Program 4: Program for final variable.

```
public class Exfinalvar
{
   public static void main(String[] args)
   {
      final int number = 100;
      System.out.println("Number = " + number);
      number = 300;// Can't be modified
   }
}
```

Output:

```
Exfinalvar.java:11: error: cannot assign a value to final variable a
a = 100;        // Can't be modified
   ^
1 error
```

Program 5: Program for final method.

```
class Vehicle
{
   final void run()
   {
      System.out.println("Two Wheeler");
   }
}
```

```
class Honda extends Vehicle
{
   void run()
   {
      System.out.println("Running 200kmph");
   }
   public static void main(String args[])
   {
      Honda h = new Honda();
      h.run();
   }
}
```

Output:

```
Honda.java:12: error: run() in Honda cannot override run() in Vehicle
void run()
   ^
overridden method is final
   1 error
```

Program 6: Program to accessing of final class.

```
final class X
{
}
class Y extends X
{
   void msg()
   {
      System.out.println("Final class not inherited...");
   }
   public static void main(String args[])
   {
      Y ob= new Y();
      ob.msg();
   }
}
```

Output:

```
Y.java:7: error: cannot inherit from final X
class Y extends X
 ^
 1 error
```

Program 7: Program for method overriding msg() in super class and sub class.

```
class Parent
{
   void msg()
   {
      System.out.println("In Parent Class");
   }
}
class Child extends Parent
{
   void msg()     //method overrides show() of Parent
   {
      super.msg();
      System.out.println("In Child Class");
   }
}
class Exmethodoverride
{
   public static void main(String[] args)
   {
      Parent obj = new Child();
      obj.msg();
   }
}
```

Output:

```
In Parent Class
In Child Class
```

Program 8: Program for runtime polymorphism.

```
class Shape
{
   void draw()
   {
      System.out.println("Geometry Shapes");
   }
}
class Rectangle extends Shape
{
   void draw()
   {
      System.out.println("Draw Rectangle...");
   }
}
```

```
class Circle extends Shape
{
   void draw()
   {
      System.out.println("Draw Circle...");
   }
}
class Triangle extends Shape
{
   void draw()
   {
      System.out.println("Draw Triangle...");
   }
}
class Exrunpoly
{
   public static void main(String args[])
   {
      Shape sobj;
      sobj=new Rectangle();
      sobj.draw();
      sobj=new Circle();
      sobj.draw();
      sobj=new Triangle();
      sobj.draw();
   }
}
```

Output:

```
Draw Rectangle...
Draw Circle...
Draw Triangle...
*/
```

Program 9: Program for abstract class (BankIndia class) and abstract method.

```
abstract class BankIndia
{
   abstract int getRateOfInterest();
}
class SBI extends BankIndia
{
   int getRateOfInterest()
   {
      return 6;
   }
}
```

```
class MAHA extends BankIndia
{
   int getRateOfInterest()
   {
      return 5;
   }
}
class Exabstract
{
   public static void main(String args[])
   {
      BankIndia bi;
      bi=new SBI();
      System.out.println("Rate of Interest is:
                                          "+bi.getRateOfInterest()+" %");
      bi=new MAHA();
      System.out.println("Rate of Interest is:
                                          "+bi.getRateOfInterest()+" %");
   }
}
```

Output:

```
Rate of Interest is: 6 %
Rate of Interest is: 5 %
```

Program 10: Program for interface (using implements). Calculate area of circle and square.

```
interface Area
{
   float pi = 3.14f;              // value declaration.
   float calArea(float x);
}
class Circle implements Area
{
   public float calArea(float x1)
   {
      float areacir = pi * x1 * x1;
      return areacir;
   }
}
```

```
class Square implements Area
{
   public float calArea(float x1)
   {
      float areasq = x1 * x1;
      return areasq;
   }
}
class ExInterface
{
   public static void main(String[] args)
   {
      Area a;                   // interface object.
      a = new Circle(); // object of circle.
      float cir = a.calArea(1.2f);
      System.out.println("Area of Circle : " +cir);
      a = new Square(); // object of square.
      float sq = a.calArea(2.5f);
      System.out.println("Area of square : " +sq);
   }
}
```

Output:

```
Area of Circle : 4.5216007
Area of square : 6.25
```

Program 11: Program to find the cube of given number using function interface.

```
@FunctionalInterface
interface Cubenum
{
   public void print(int num1);
}
public class Exfuncinterface
{
   public static void main(String[] a)
   {
      Cubenum cn = n -> System.out.println("cube is: "+ n*n*n);
      p.print(7);
   }
}
```

Output:

```
Cube is : 343
```

Program 12: Program for interface extends another interface.

```
interface A
{
   void funa();
}
interface B extends A
{
   void funb();
}
class X implements B
{
   public void funa()
   {
      System.out.println("Good Morning");
   }
   public void funb()
   {
      System.out.println("Good Afternoon");
   }
}
public class Exinterface1
{
   public static void main(String[] args)
   {
      X obj = new X();
      obj.funa();
      obj.funb();
   }
}
```

Output:

```
Good Morning
Good Afternoon
```

Program 13: Program for hierarchical inheritance.

```
class Pclass
{
   int a;
   void getdata(int a)
   {
      this.a = a;
   }
}
```

```
class Cclass extends Pclass
{
   void show()
   {
      System.out.println("Child Class C");
      System.out.println("Value of a is " + a);
   }
}
class Dclass extends Pclass
{
   void display()
   {
      System.out.println("Child Class D");
      System.out.println("Value of a is " + a);
   }
}
public class Exhierarchicalinh
{
   public static void main(String[] args)
   {
      Cclass cobj = new Cclass();
      cobj.getdata(50);
      cobj.show();
      Dclass dobj = new Dclass();
      dobj .getdata(100);
      dobj .display();
   }
}
```

Output:

```
Child Class C
Value of a is 50
Child Class D
Value of a is 100
```

Program 14: Program to define an abstract class "Staff" with members name and address. Define two sub-classes of this class – "FullTimeStaff" (department, salary) and "PartTimeStaff" (number-of-hours, rate-perhour). Define appropriate constructors. Create n objects which could be of either FullTimeStaff or PartTimeStaff class by asking the user's choice. Display details of all "FullTimeStaff" objects and all "PartTimeStaff" objects.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
abstract class Staff
{
   String name,address;
}
class FullTimeStaff extends Staff
{
   String department;
   double salary;
   public void accept() throws IOException
   {
      System.out.println("Enter the name, address, department and salary:");
      BufferedReader br=new BufferedReader
                                        (new InputStreamReader(System.in));
      name=br.readLine();
      address=br.readLine();
      department=br.readLine();
      salary=Double.parseDouble(br.readLine());
   }
   public void display()
   {
      System.out.println("Name: "+name);
      System.out.println("Address: "+address);
      System.out.println("Department: "+department);
      System.out.println("Salary: "+salary);
      System.out.println("----------------------");
   }
}
class PartTimeStaff extends Staff
{
   int hours, rate;
   public void accept() throws IOException
   {
      System.out.println("Enter the name, address, No of working hours and
      rate per hour: ");
      BufferedReader br=new BufferedReader
                                        (new InputStreamReader(System.in));
```

```
        name=br.readLine();
        address=br.readLine();
        hours=Integer.parseInt(br.readLine());
        rate=Integer.parseInt(br.readLine());
    }
    public void display()
    {
        System.out.println("Name: "+name);
        System.out.println("Address: "+address);
        System.out.println("No of Working Hours: "+hours);
        System.out.println("Rate per hour: "+rate);
        System.out.println("----------------------");
    }
}
public class sample
{
    public static void main(String [] args) throws IOException
    {
        int i;
        System.out.println("Select Any One: ");
        BufferedReader br=new BufferedReader
                                        (new InputStreamReader(System.in));
        System.out.println("1.Full Time Staff");
        System.out.println("2.Part Time Satff");
        int ch=Integer.parseInt(br.readLine());
        switch(ch)
        {
            case 1:
            System.out.println("Enter the number of Full Time Staff: ");
            int n=Integer.parseInt(br.readLine());
            FullTimeStaff [] l=new FullTimeStaff[n];
            for(i=0;i<n;i++)
            {
                l[i]=new FullTimeStaff();
                l[i].accept();
            }
```

```
            for(i=0;i<n;i++)
            {
               l[i].display();
            }
            break;
            case 2:
            System.out.println("Enter the number of Part Time Staff: ");
            int m=Integer.parseInt(br.readLine());
            PartTimeStaff [] h=new PartTimeStaff[m];
            for(i=0;i<m;i++)
            {
               h[i]=new PartTimeStaff();
               h[i].accept();
            }
            for(i=0;i<m;i++)
            {
               h[i].display();
            }
            break;
         }
      }
   }
```

Output:

```
Select Any One:
1.Full Time Staff
2.Part Time Satff
1
Enter the number of Full Time Staff:
2
Enter the name, address, department and salary:
adhira
pune
hr
40000
Enter the name, address, department and salary:
kishor
nagar
comp
55000
```

```
Name: adhira

Address: pune
Department: hr
Salary: 40000.0
----------------------
Name: kishor
Address: nagar
Department: comp
Salary: 55000.0
----------------------
```

Program 15: Program to demonstrate runtime polymorphism using interface.

```
interface Area
{
   final static float pi=3.14f;
   float compute(float x, float y);
}
class rectangle implements Area
{
   public float compute(float x,float y)
   {
      return(x*y);
   }
}
class circle implements Area
{
   public float compute(float x,float y)
   {
      return(pi*x*y);
   }
}
class interfacetest
{
   public static void main(String args[])
   {
      rectangle rect=new rectangle();
      circle cir=new circle();
      Area area;
```

```
        area=rect;
        System.out.println("Area of rectangle =" +area.compute(10,20));
        area=cir;
        System.out.println("Area of circle="+area.compute(10,10));
    }
}
```

Output:

```
Area of rectangle =200.0
Area of circle=314.0
```

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. Which of these keyword must be used to inherit a class?
   (a) super (b) this
   (c) extent (d) extends
2. Which of these is correct way of inheriting class A by class B?
   (a) class B + class A {} (b) class B inherits class A {}
   (c) class B extends A {} (d) class B extends class A {}
3. Which of these keywords is used to refer to member of base class from a sub class?
   (a) upper (b) super
   (c) this (d) none of the mentioned
4. A class member declared protected becomes member of subclass of which type?
   (a) public member (b) private member
   (c) protected member (d) static member
5. What is not type of inheritance?
   (a) Single inheritance (b) Double inheritance
   (c) Hierarchical inheritance (d) Multiple inheritance
6. Which of the following is used for implementing inheritance through interface?
   (a) inherited (b) using
   (c) extends (d) implements
7. What would be the result if class extends two interfaces and both have method with same name and signature?
   (a) Runtime error
   (b) Compile time error
   (c) Code runs successfully
   (d) First called method is executed successfully

8. What is the purpose of Boolean Supplier function interface?
   (a) returns Boolean-valued result
   (b) returns null if Boolean is passed as argument
   (c) represents supplier of Boolean-valued results
   (d) There is no such function interface
9. What is the return type of lambda expression?
   (a) Function (b) String
   (c) Object (d) void
10. Which of these keywords can be used to prevent method overriding?
   (a) static (b) constant
   (c) protected (d) final
11. Which of the following is a marker interface?
   (a) Runnable interface (b) Remote interface
   (c) Readable interface (d) Result interface
12. Runtime polymorphism is also known as _____.
   (a) Static binding (b) Early binding
   (c) Dynamic method dispatch (d) Method overloading
13. Polymorphism that is resolved during compiler time is known as _____.
   (a) Dynamic polymorphism (b) Static polymorphism
   (c) Special Polymorphism (d) None of the above
14. In inheritance, the old class is known as _____.
   (a) Start class (b) Base class
   (c) Derived class (d) Sub class
15. Base class is also known as _____.
   (a) Super class (b) Child class
   (c) Sub class (d) Derived class
16. If a class inheriting an abstract class does not define all of its function then it will be known as?
   (a) Abstract (b) A simple class
   (c) Static class (d) final class
17. What is not type of inheritance?
   (a) Single inheritance (b) Double inheritance
   (c) Hierarchical inheritance (d) Multiple inheritance
18. All classes in Java are inherited from which class?
   (a) java.lang.class (b) java.class.inherited
   (c) java.class.object (d) java.lang.Object

19. Super class and subclass have same variable name, which keyword should be used to use super class?
    (a) upper (b) this
    (c) super (d) classname
20. The final keyword in java is used with _____.
    (a) class (b) class fields
    (c) class methods (d) All of the above
21. Which is the capability of one class to inherit capabilities or properties from another class?
    (a) Class (b) Encapsulation
    (c) Inheritance (d) Polymorphism
22. In which inheritance there is a single child class that inherits properties from single parent class.
    (a) single (b) multilevel
    (c) hierarchical (d) multiple
23. If we initialize a variable with the _____ keyword, then we cannot modify its value.
    (a) static (b) final
    (c) abstract (d) extends
24. When a subclass is derived from a derived class then this mechanism is known as,
    (a) multilevel inheritance (b) multiple inheritance
    (c) single inheritance (d) hierarchical inheritance
25. The final variables cannot be,
    (a) static (b) constant
    (c) modified (d) abstract
26. Which keyword refers to superclass (parent) objects.?
    (a) static (b) abstract
    (c) super (d) final
27. The abstract methods are declared using the keyword,
    (a) abstract (b) super
    (c) extends (d) final
28. Which is a collection of abstract methods?
    (a) inheritance (b) package
    (c) interface (d) polymorphism
29. Following which method that does not have its body,
    (a) abstract (b) static
    (c) final (d) All of mentioned

30. Which keyword is used for implementation of interface?
    (a) implements (b) extends
    (c) abstract (d) final
31. Which is one of alternative way to achieve multiple inheritance?
    (a) inheritance (b) package
    (c) interface (d) polymorphism
32. Following which simply means more than one form.
    (a) inheritance (b) package
    (c) interface (d) polymorphism
33. Marker interface is also known as,
    (a) tag interface (b) class interface
    (c) functional interface (d) None of mentioned
34. An interface that contains exactly one abstract method is known as,
    (a) tag interface (b) class interface
    (c) functional interface (d) None of mentioned

Answers

1. (d)	2. (c)	3. (b)	4. (b)	5. (d)	6. (d)	7. (b)	8. (c)	9. (a)	10. (d)
11. (b)	12. (c)	13. (b)	14. (b)	15. (a)	16. (a)	17. (b)	18. (d)	19. (c)	20. (d)
21. (c)	22. (a)	23. (b)	24. (a)	25. (c)	26. (c)	27. (a)	28. (c)	29. (a)	30. (a)
31. (c)	32. (d)	33. (a)	34. (c)						

Q.II Fill in the Blanks:

1. An interface that does not have any data members or methods is called _____ interface.
2. Inheritance represents the _____ relationship.
3. _____ keyword indicates that you are making a new class that derives from an existing class.
4. A class inherits another class, it is known as a _____ inheritance.
5. A class which is declared with the abstract keyword is known as an _____ class.
6. An interface is _____ class.
7. A Java class inherits constants and methods of an interface using _____ keyword.
8. All interface variables are _____ and final by default in Java.
9. All interface methods in Java are public and _____ by default.
10. A class implementing an interface can use _____ access modifier before the implemented methods.

11. Interface static methods have _____ and backword compatibility with the existing project code.
12. The annotation used in Java to override the method of a super-class or interface by the subclass or implementing class is _____.
13. A static method of an interface should be accessed with interface _____ and a dot operator.
14. An interface with no fields or methods is known as a _____ Interface.
15. _____ class cannot be sub classed.
16. _____ keyword is used to invoke the superclass constructor from subclass.
17. The extends is the keyword used to _____ the properties of a class.
18. An interface can extend _____ interfaces.
19. The final _____ cannot be inherited by other classes.
20. The purpose of creating the final _____ is to restrict the unwanted and improper use of method definition while overriding the method.
21. We can also declare a class with a _____ keyword in Java.
22. An _____ method contains a method signature, but no method body.
23. An _____ does not contain any constructors.
24. A class uses the_____ keyword to implement an interface.
25. The method defined in the subclass is invoked and executed instead of superclass method known as _____ overriding.
26. We use the abstract keyword to declare an abstract_____.
27. One interface can inherit another by use of the keyword _____.
28. A process in which a call to an overridden method is resolved at run-time rather than compile time is known as _____-time polymorphism.

Answers

1. marker	2. IS-A	3. extends	4. single	5. abstract
6. abstract class	7. implements	8. public	9. abstract	10. public
11. forward	12. @Override	13. name	14. Marker	15. final
16. super	17. inherit	18. multiple	19. classes	20. methods
21. final	22. abstract	23. interface	24. implements	25. method
26. class	27. extends	28. run		

Q.III State True or False:

1. All methods must be implemented of an interface.
2. Can “abstract” keyword be used with constructor, initialization block, instance initialization and static initialization block.

3. Method overriding is useful to add extra functionality or code to a method of subclass with the same name as the inherited method.
4. Final method can be overridden.
5. It can be instantiated, means we can create an object of an interface.
6. A Java Interface is not considered a class.
7. A Java Class implementing an Interface can define a variable with the same name as that of the Interface constant.
8. A Java Interface can not declare constructors.
9. An interface is a blueprint of a class. It has static constants and abstract methods.
10. Method Overloading is possible by changing the return type of the method.
11. A class can extend only one class but can implement many interfaces.
12. An interface can extend many interfaces.
13. An interface can implement a class.
14. We can define a method in an interface.
15. Multiple inheritances is not supported in case of class but it is supported in case of interface.
16. Inheritance is process of deriving a old class from derived class.
17. If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
18. Method overriding is used for runtime polymorphism
19. In method overriding Both the superclass and the subclass must have the same method name, the same return type and the same parameter list.
20. The super keyword in Java is used in subclasses to access superclass members (attributes, constructors and methods).
21. An interface is a fully abstract class which includes a group of abstract methods (methods without a body).
22. We use the interface keyword to create an interface in Java.
23. To use an interface, other classes must implement it. We use the abstract keyword to implement an interface.
24. If the same method is present in both the superclass and the subclass. Then, the method in the subclass overrides the same method in the superclass called method overriding.
25. The argument list should be exactly the same as that of the overridden method.

26. A class which contains the abstract keyword in its declaration is known as abstract class.
27. The implements keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.
28. When an interface has no members (declared or inherited), it is known as a marker interface.

Answers

1. (T)	2. (F)	3. (T)	4. (F)	5. (F)	6. (T)	7. (T)	8. (T)	9. (T)	10. (F)
11. (T)	12. (T)	13. (F)	14. (F)	15. (T)	16. (F)	17. (T)	18. (T)	19. (T)	20. (T)
21. (T)	22. (T)	23. (F)	24. (T)	25. (T)	26. (T)	27. (F)	28. (T)		

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. Define inheritance.
2. Which keyword is used for creating inheritance?
3. What is abstract class?
4. Define run time polymorphism.
5. What are the types of inheritance?
6. What is final?
7. What is purpose of super?
8. Define interface?
9. Which keyword is used for implementing interface?
10. Define abstract method.
11. What is marker interface?
12. Give uses of functional interface.
13. Define superclass and subclass.
14. State any two advantages of inheritance.
15. Define final class.

(B) Long Answer Questions:

1. What is inheritance? How to create it? Explain with example.
2. With the help of example describe following inheritances in Java:
   (i) Single (ii) Multilevel (iii) Multiple.
3. Describe abstract classes and methods? Also compare them.
4. How to override methods in Java? Describe with example.
5. What is hierarchical inheritance? Explain in detail with example.
6. What is interface? How to implement it? Explain with example.
7. Explain nested interface with example.
8. How to extending Interface? Explain with example.

9. Define abstract class. Which keyword is used to define abstract class?
10. Write short notes on: Runtime polymorphism using interface.
11. With the help of example describe following interface:
    (i) Marker (ii) Functional
12. Describe super keyword with example.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. "super keyword is used by subclass to refer its immediate superclass". State true or false. Justify. **[1 M]**

Ans. Refer to Section 3.2.2.

2. Write a Java program to illustrate multilevel inheritance such that state is inherited from country. City is inherited from state. Display the areas, city, state and country. **[5 M]**

Ans. Refer to Section 3.1.2 Point (2).

October 2016

1. What is the use of super keyword? **[1 M]**

Ans. Refer to Section 3.2.2.

2. What is interface? How does it differ from abstract class? Explain. **[5 M]**

Ans. Refer to Section 3.6.

April 2017

1. What is the use of final keyword? **[1 M]**

Ans. Refer to Section 3.4.

2. One interface can implement another interface. True/False – Justify. **[1 M]**

Ans. Refer to Section 3.6.4.

3. Explain method overriding with example. **[3 M]**

Ans. Refer to Section 3.3.

October 2017

1. Create an abstract class Employee. Derive two classes manager and worker from it. Use proper method to accept and display the details for the same. The fields of manager are mid, mname and phno. Similarly, fields for worker are name and working hours. **[5 M]**

Ans. Refer to Section 3.5.

2. Explain the use of keywords super and final with reference to inheritance. **[3 M]**

Ans. Refer to Sections 3.2.2 and 3.4.

April 2018

1. When we declare a method or class final? **[1 M]**

Ans. Refer to Section 3.4.2.

2. What is runtime polymorphism? How is it implemented in Java? Give suitable example. **[5 M]**

Ans. Refer to Section 3.3.

3. Explain two types of Inheritance with suitable example. **[3 M]**

Ans. Refer to Section 3.1.2.

October 2018

1. Write down any two differences between the abstract class and interface. **[2 M]**

Ans. Refer to Sections 3.5 and 3.6.

April 2019

1. What is use of final keyword? **[1 M]**

Ans. Refer to Section 3.4.

2. Write difference between Abstract class and Interface. **[3 M]**

Ans. Refer to Sections 3.5 and 3.6.

■■■

CHAPTER 4

Exception and File Handling

Objectives...

- To study Errors and Exceptions
- To understand Basic Concepts of Exception Handling
- To learn File Handling

4.0 INTRODUCTION

- It is a common to take mistakes while developing as well as typing a program. A mistake might lead to an error causing a program to produce unexpected results.
- An error may produce an incorrect output or may terminate the execution of the program abruptly.
- It is therefore important to detect and manage properly all the possible error conditions in the program so that the program will not terminate during execution.
- An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run-time, which disrupts the normal flow of the program's instructions.
- When an exception occur the normal flow of the program is disrupted and the program terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.
- Java uses Exception Handling mechanism to handle the runtime errors so that normal flow of the application can be maintained.
- File handling in Java is one such functionality that allows us to work with files. A named location used to store related information is known as a file.
- There are several file operations like creating a new file, getting information about file, writing into a file, reading from a file and deleting a file.
- Java has several methods for handling file operations such as creating, reading, updating and deleting files.

4.1 DEALING WITH ERROR

- When executing Java code, different errors can occur like coding errors made by the programmer, errors due to wrong input or other unforeseeable things.

- When an error occurs, Java will normally stop and generate an error message. The technical term for this is Java will throw an exception (throw an error).
- An error is a situation or condition when a compiler either doesn't execute statements or either compiler will produce wrong result.
- Error is an illegal operation performed by the user which results in the abnormal working of the program.
- Programming errors often remain undetected until the program is compiled or executed. So errors should be removed before compiling and executing of the program.
- Various types of errors in Java are explained below:

1. **Syntax Error or Compile Time Error:**

- Errors that are detected by the compiler in compilation are called as compile errors or syntax errors.
- Compiler time error means Java compiler identify the syntax error at the time of compilation. And without successfully compilation, compiler does not create .class file.
- The compile time errors include misspelled variable and function names, missing semicolons, improperly matches parentheses, square brackets and curly braces, used undefined variable, incorrect format in selection and loop statements etc.

2. **Run Time Error:**

- Several time program may compile successfully and compiler creates the .class file of the program but when the time of running the program, it shows the error and that type of error called run-time error.
- Run-time errors are errors which occur when the program is running. Run time errors are not detected by the java compiler. It is the JVM which detects it while the program is running.
- Run-time errors are caught by interpreter or at the time of execution when we execute out program the error is encountered.
- Run time error includes operations those are not possible but a user trying to do them. Common examples of run-time error are, trying to divide by a variable that contains a value of zero, trying to open a file that doesn't exist etc.

3. **Logical Error:**

- Logic errors occur when there is a design flaw in the program. Logic errors occur when a program does not perform the way it was intended to.
- This type of error doesn't halt or stops the execution of Java program but they produce wrong results.
- This type of error is occurred when there is some mistake in logic of user like user wants to multiply two numbers and user puts addition mark instead of multiplication then it is called as logical error.

Program 4.1: Program for error.

```
public class ErrorExample
{
   public static void main(String[] args)
   {
      recursiveMethod(10)
   }
   public static void recursiveMethod(int i)
   {
      while(i!=0)
       {
         i=i+1;
         recursiveMethod(i);
      }
   }
}
```

Output:

```
Exception in thread "main" java.lang.StackOverflowError
   at ErrorExample.ErrorExample(Main.java:42)
```

4.2 EXCEPTION AND EXCEPTION CLASS

- An exception is an event that occurs during the execution of a program and it interrupts the normal flow of program executions.
- In this section we will study exception and exception class of Java.

4.2.1 Exceptions

- Exception is an abnormal condition disrupts the normal flow of the program. An exception (or exceptional event) is a problem that arises during the execution of a program.
- Exception is a condition that is caused by runtime errors in program. We can define exception as “an event occurs during the execution of a program that interrupts the normal flow of the program".
- Exception is a runtime error which can be handled or prevented and occurs during the execution of the program.
- Some exceptions can be detected by java compiler while compiling and are called checked exceptions while those detected during runtime by JVM are called unchecked exceptions.

- An exception can occur for many different reasons. Following are some scenarios where an exception occurs.
  - A user has entered an invalid data.
  - A file that needs to be opened cannot be found.
  - A network connection has been lost in the middle of communications or the JVM has run out of memory.
- Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Program 4.2: Program for exception.

```
public class ExceptionExample
{
   public static void main(String[] args)
   {
      int x = 100;
      int y = 0;
      int z = x / y;
   }
}
```

Output:

```
java.lang.ArithmeticException: / by zero
   at ExceptionExample.main(ExceptionExample.java:7)
```

Difference between Exception and Error in Java:

- Exceptions are related to the application and an error is related to the environment in which the application is running.
- An Error cannot be recovered as it is fatal in nature, whereas Exception that may not be fatal in all cases.
- Errors mostly occur at runtime that's they belong to an unchecked type. Exceptions are the problems which can occur at runtime and compile time.
- The error indicates a problem that mainly occurs due to the lack of system resources and the program should not catch these types of problems. Exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's Instructions.
- An error is irrecoverable while an exception is recoverable.
- Exceptions are divided into two categories such as checked exceptions and unchecked exceptions. Error classified as an unchecked type
- Exceptions and errors both are subclasses of Throwable class. Example of error is OutOfMemoryError, IOError while example of exception NullPointerException, SqlException

4.2.2 Exception Class

- All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class.
- Throwable is a class in java.lang package and it represents all errors and exceptions in Java.
- The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses namely, Exception class and Error.
- The Exception class has two main subclasses namely, IOException class and RuntimeException class.
- A hierarchy of Java Exception classes is given in Fig. 4.1.

Fig. 4.1

4.2.3 Types of Exceptions [Oct. 18]

- The categories of Exceptions are explained below:

1. **Checked Exceptions:** [Oct. 18]

- A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions.
- These exceptions cannot simply be ignored, the programmer should take care of (handle) these exceptions.

  For example, if we use FileReader class in the program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs and the compiler prompts the programmer to handle the exception.

Program 4.2: Program for checked exception.

```
import java.io.File;
import java.io.FileReader;
public class FilenotFound_Demo
{
   public static void main(String args[])
   {
      File file = new File("E://file.txt");
      FileReader fr = new FileReader(file);
   }
}
```

Output:

```
FilenotFound_Demo.java:8: error: unreported exception
FileNotFoundException; must be caught or declared to be thrown
FileReader fr = new FileReader(file);
                      ^
1 error
```

2. Unchecked Exceptions: **[Oct. 18, April 19]**

- An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions.
- These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

  For example, if we have declared an array of size 5 in the program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptionexception occurs.

Program 4.3: Program for unchecked exception.

```
public class Unchecked_Demo
{
   public static void main(String args[])
   {
      int num[] = {1, 2, 3, 4};
      System.out.println(num[5]);
   }
}
```

Output:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
   at Exceptions.Unchecked_Demo.main(Unchecked_Demo.java:8)
```

3. Errors:

- These are not exceptions at all, but problems that arise beyond the control of the user or the programmer.
- Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs.
- Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.
- Errors are typically ignored in the code because we can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.

4.2.4 Exception Handling [Oct. 17, 18]

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a program code.
- When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on.
- When an Exception has occurred then compiler will never reach to the code that is placed after the block which contains an error.
- For handling Exceptions java provides us special mechanism which is called as exception handling mechanism.
- Java exception handling mechanism is based on following three keywords namely try, catch and finally.

  1. **try:** The try block contain any statements that might cause an exception to occur. A try statement is used to catch exceptions that might be thrown as the program executes. Java try block must be followed by either 'catch' or 'finally' block.
  2. **catch:** The catch block(s) provide a place to handle the exception thrown by the statements within a try block. The catch block is defined directly after the try block. It must specify the type of exception it is handling. A catch statement involves declaring the type of exception we are trying to catch.
  3. **finally:** The statements in the finally block are always executed. This is useful to clean up resources in the event of the try block executing without an exception and in the cases when there is an exception. The finally block follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an exception. The finally block appears after the last catch block. **[Oct. 16, 18]**

- This is the general **form/syntax** of an exception-handling mechanism:

```
try
{
   // block of code to monitor for errors
}
```

```
catch (ExceptionType1 exOb)
{
   // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
   // exception handler for ExceptionType2
}
   // ...
finally
{
   // block of code to be executed after try block ends
}
```

Program 4.4: Program using try and catch blocks. An array declared with two elements. Then the code tries to access the 3rd element of the array which throws an exception.

```
import java.io.*;
public class ExcepTest
{
   public static void main(String args[])
   {
      try
      {
         int a[] = new int[2];
         System.out.println("Access element three:" + a[3]);
      } catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Exception thrown:" + e);
      }
      System.out.println("Out of the block");
   }
}
```

Output:

```
Exception thrown:java.lang.ArrayIndexOutOfBoundsException: 3
   Out of the block
```

Program 4.5: Program using try, catch and finally blocks.

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

```
    {
        int a[] = new int[2];
        try {
            System.out.println("Access element three:" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown:" + e);
        }finally {
            a[0] = 6;
            System.out.println("First element value: " + a[0]);
            System.out.println("The finally statement is executed");
        }
    }
}
```

Output:

```
Exception thrown:java.lang.ArrayIndexOutOfBoundsException: 3
First element value: 6
The finally statement is executed
```

4.3 CATCHING EXCEPTIONS

- Java exception arises during the execution of a program. Exceptions occur unexpectedly and can result in disrupting the normal flow of the program.
- A catch block must specify what type of exception it will catch.

  Syntax:

```
catch (ExceptionClassName exceptionObjectName)
{
    // Code using methods from exceptionObjectName
}
```

- A method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code.
- The try statement allows us to define a block of code to be tested for errors while it is being executed. The catch statement allows us to define a block of code to be executed, if an error occurs in the try block.

Syntax:

```
try
{
   // Protected code…
}
catch (ExceptionName e1)
{
   // Catch block…
}
```

- The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it.
- Every try block should be immediately followed either by a catch block or finally block. A catch statement involves declaring the type of exception we are trying to catch.
- If an exception occurs in protected code, the catch block (or blocks) that follows the try is checked.
- If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Program 4.6: Program for catch block.

```
import java.io.*;
public class ExcepTest
{
   public static void main(String args[])
   {
      try
      {
         int a[] = new int[2];
         System.out.println("Access element three:" + a[3]);
      } catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println("Exception thrown:" + e);
      }
      System.out.println("Out of the block");
   }
}
```

Output:

```
Exception thrown:java.lang.ArrayIndexOutOfBoundsException: 3
   Out of the block
```

4.3.1 Multiple Catch Blocks

- Several types of exceptions may arise when executing the program. They can be handled by using multiple catch blocks for the same try block.
- When an exception is thrown, each catch statement is inspected in order and the first one whose type matches that of the exception is executed.
- After one catch statement executes, the others are bypassed, and execution continues after the try/catch block.
- The **syntax for multiple catch blocks** looks like the following:

```
try
{
   // Protected code …
}catch(ExceptionType1 e1)
{
   // Catch block …
}catch(ExceptionType2 e2)
{
   // Catch block …
}catch(ExceptionType3 e3)
{
   // Catch block …
}
```

- If an exception occurs in the protected code, the exception is thrown to the first catch block in the list. If the type of the exception thrown matches ExceptionType1, it gets caught there.
- If not, the exception passes down to the second catch statement. This continues until the exception either is caught or falls through all catches.
- In which case the current method stops execution and the exception is thrown down to the previous method on the call stack.

Program 4.7: Program for using multiple catch statements.

```
class MultiCatch
{
   public static void main(String args[])
   {
      int x,z;
      Scanner in = new Scanner(System.in);
      System.out.print("Enter number: ");
      x = in.nextInt();
```

```
        try
        {
            z = 50 / x; //statement1
            System.out.println("Division: "+z);
            Short arr[] = {5,7,8};
            arr[10] = 120; //statement2
            System.out.println("Try end...");
        }
        catch(ArithmeticException e)
        {
            System.out.println("Division by zero");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Array indexing wrong");
        }
        System.out.println("Program end...");
    }
}
```

Output:

```
Enter number: 0
Division by zero
Program end...
Enter number: 5
Division: 10
Array indexing wrong
Program end...
```

- Above Program 4.7 illustrates the use of multiple catch statements with single try. We have assume that the code written in the try block will cause either divide by zero error or array index will go out of bounds. If any one of these exceptions is occurred it will be caught appropriate catch statement and action will be taken.
- Depending upon the type of exception object thrown only one of the catch statements will be executed. If the exception thrown does not match with the exception object written in catch block, program will be terminated.
- In above program we have taken input from keyboard. If value of that input i.e. x is 0 then the divide by zero exception will occur at statement1 else array index out of bounds exception will occur at statement2. These exceptions will be caught by appropriate catch clause.

4.3.2 throws and throw Keywords [April 16]

- If a method does not handle a checked exception, the method must declare it using the throws keyword.
- We can throw an exception, either a newly instantiated one or an exception that we just caught, by using the throw keyword.

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   // Remainder of class definition
}
```

- The difference between throws and throw keywords is, throws is used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly.
- A throws statement/keyword lists the types of exceptions that a method might throw. The general **form/syntax** of the throws keyword is given below:

```
type method-name(parameter-list) throws exception-list
{
   // body of method…
}
```

- When we define function and we don't want to put try and catch block than we can use throws keyword. Throws keywords with a method declare that a risky method is being called.

Program 4.8: Program for throws keyword.

```
import java.util.*;
public class Sample
{
   public static void main(String[] args)
   {
      int a=5;
      int b=10;
      int res= compute(a,b);
      System.out.println("a="+a);
      System.out.println("b="+b);
      System.out.println("res="+res);
   }
```

```
    staticint compute(inta,int b) throws ArithmeticException
    {
        return(a / (b-10));
    }
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
atSample.compute(Sample.java:15)
atSample.main(Sample.java:8)
```

- The 'throw' keyword can be used whenever an exception needs to be thrown explicitly.
- The general **form/syntax** of throw is shown below:

```
throw new ExceptionObj
```

Program 4.9: Program for throw keyword.

```
class ThrowDemo
{
    public static void main(String args[])
    {
        int x = 10, y = 20;
        int z;
        z = x + y;
        try
        {
            throw new ArithmeticException();
        }
        catch(Exception e)
        {
            System.out.println("Exception caught");
            System.out.println("Addition: "+z);
        }
    }
}
```

Output:

```
Exception caught
Addition: 30
```

- In above program, an ArithmeticException has been thrown by throw statement and it is caught by the catch clause if the exception thrown by throw and caught by catch does not matches the program, will get terminated before printing any output. Just change the catch clause as,

```
catch(NegativeArraySizeException e)
```

- We will get the output:

```
Exception in thread "main" java.lang.ArithmeticException
atThrowDemo.main(ThrowDemo.java:11)
```

- We can give the description to the exception class also. Such as,

```
throw new ArithmeticException("Divide Error");
```

- Just make the changes in the program as,

```
try
{
throw new ArithmeticException("Divide Error");
}
catch(Exception e)
{
   System.out.println("Exception:"+e);
   System.out.println("Addition: "+z);
}
```

- We will get the following output:

```
Exception:java.lang.ArithmeticException: Divide Error
Addition: 30
```

- All of Java's built-in run-time exceptions have at least two constructors: one with no parameter and one that takes a string parameter. When the second form is used, the argument specifies a string that describes the exception.
- This string is displayed when the object is used as an argument to print() or println(). It can also be obtained by a calling method getMessage(), which is defined by Throwable.

Difference between throw and throws in Java: **[April 16, Oct. 17]**

Sr. No.	throw	throws
1.	Java, throw keyword is used to explicitly throw an exception.	Java, throws keyword is used to declare an exception.
2.	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3.	Throw is followed by an instance.	Throws is followed by class.

contd. ...

4.	Throw is used within the method.	Throws is used with the method signature.
5.	We cannot throw multiple exceptions.	We can declare multiple exceptions e.g.public void method()throws IOException, SQLException.

6.	Example:
	`void m(){`
	`throw new ArithmeticException("sorry");`
	`}`
		Example:
		`void m()throws ArithmeticException{`
		`//method code`
		`}`

4.3.3 Nested try Block

- Sometimes, a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
- The try block within a try block is known as nested try block.

Syntax:

```
try
{
    // Statement1;
    // Statement2;
    try
    {
        // Statement1;
        //  Statement2;
    }
    catch(Exception excep)
    {
    }
}
catch(Exception excep)
{
}
```

Program 4.10: Program illustrates the use of nested try.

```
class NestTry
{
   public static void main(String args[])
   {
      int x,z;
      Scanner in = new Scanner(System.in);
      System.out.print("Enter number: ");
      x = in.nextInt();
      try
      {
         z = 82 / x; //statement1
         System.out.println("Division: "+z);
         try
         {
            int a = 100 / (x-1); //statement2
            short arr[] = {15};
            arr[10] = 25; //statement3
            System.out.println("Inner try end...");
         }
         catch(ArrayIndexOutOfBoundsException e) //1
         {
            System.out.println("Array indexing wrong");
         }
         System.out.println("Outer try end...");
      }
      catch(ArithmeticException e) //2
      {
         System.out.println("Division by zero");
      }
      System.out.println("Program end...");
   }
}
```

Outputs:

```
Enter number: 0
Division by zero
Program end...
Enter number: 1
Division: 82
Division by zero
Program end...
Enter number: 2
Division: 41
Array indexing wrong
Outer try end...
Program end...
```

- Above Program 4.10 has nested two try statements in each other. We have taken input as a number from keyboard and stored it into variable x. When value of x is inputted as 0, statement1 will cause an exception “Divide by zero”.
- So program control will directly transfer to catch statement no.2 to take the action. When value of x inputted is 1. The outer 'try' will work properly. In inner try, the statement2 will cause the exception of “Divide by zero”.
- In this case, the program control will get directly transferred to inner try’s catch block i.e. catch clause no. 1. But the exception object will not match. So it will check for outer try’s catch clause. Now, the match is found. That’s why, we got the output no.2.
- In the third case, when the input is any number other than 0 and 1, statement3 will cause an exception “Array index out of bounds”. This will be caught by inner try’s catch clause.
- That is, the outer try’s catch clause will catch all the exception occurred inside it. Remember, if the exception of “array index out of bounds” is occurred in outer try then this will not be caught by inner try’s catch clause.

4.4 CREATING USER DEFINED EXCEPTIONS [April 17, 18, 19, Oct. 17]

- Though Java provides an extensive set of in-built exceptions, there are cases in which we may need to define our own exceptions in order to handle the various application specific errors that we might encounter.
- If a user is creating his/her own Exception is known as user defined exception (or custom exception). Java user defined exceptions are used to customize the exception according to user need.

- In java we can create our own exception class and throw that exception using throw keyword. There are following two basic reasons for defining our own exception classes:
  1. We want to add information when a standard exception occurs, and
  2. We may have error conditions that arise in your code.

Syntax:

```
class exceptionclassname extends Exception
{
    exceptionclassname(parameter)
    {
        //statements
    }
    public String toString()
    {
        return String
    }
}
```

- The user defined exceptions must be explicitly thrown using the 'throw' statement whose syntax is,

```
throw ThrowableInstance
```

- ThrowableInstance must be an object of type Throwable or a subclass of Throwable. Following are some of the methods defined by Class Throwable.

Method	Meaning
1. `String getMessage()`	Returns a description of the exception.
2. `Throwable getCause()`	Returns the exception that underlies the current exception.
3. `StackTraceElement[]getStackTrace()`	Returns an array that contains the stack trace.
4. `Void printStackTrace()`	Prints the Stack trace.
5. `String toString()`	Returns a String object containing a description of the exception.

Program 4.11: Program for creating own exception.

```
class NegativeOutputException extends Exception
{
    private int det;
    NegativeOutputException(int a)
    {
        det = a;
    }
```

```
    public String toString()
    {
        return "NegativeOutputException["+det+"]";
    }
}
class OwnException
{
    public static void main(String args[])
    {
        int x = Integer.parseInt(args[0]);
        int y = Integer.parseInt(args[1]);;
        int z;
        try
        {
            z = x * y;
            if(z<0) //statement1
            throw new NegativeOutputException(z);
            System.out.println("Output: "+z);
        }
        catch (NegativeOutputException e)
        {
            System.out.println("Caught: "+e);
        }
    }
}
```

Output:

```
javaOwnException 4 8
Output: 32
javaOwnException 4 -3
Caught: NegativeOutputException[-12]
javaOwnException -4 -3
Output: 12
```

- Observe the above Program 4.11. In order to create our own exception we have inherited our exception class from java.lang.Exception. Name of the defined exception is NegativeOutputException.
- The method toString() has been overridden to print our own custom message with exception class. The condition for the exception defined is that when the result of the expression is negative then exception must occur. It is defined in statement1.

- The input has taken from the command line. First argument is passed to x and second to y. First time and third time when both the arguments are positive/negative the result is positive so we will get the output properly.
- But when any one of the arguments is negative the result is also negative. The statement1 will be true. Now exception is thrown using the throw clause.
- This exception is caught by catch clause. If we do not catch it using catch the compile time error will occur.

4.5 INTRODUCTION TO FILES AND STREAMS [Oct. 16, 18, April 19]

- A file is a named location that can be used to store related information. The File class from the java.io package (java.io.File), allows us to work with files.
- The java.io package provides an extensive library of classes for dealing with Input and Output (I/O).
- Java provides streams as a general mechanism for dealing with data I/O. Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.
- Streams implement sequential access of data. There are two kinds of streams: byte streams and character streams.
- An input stream is an object that an application can use to read a sequence of data, and an output stream is an object that an application can use to write a sequence of data.
- An input stream acts as a source of data, and an output stream acts as a destination of data.
- Java I/O stream is the flow of data that we can either read from, or we can write to. It is used to perform read and write operations in file.
- Java I/O stream is also called File Handling, or File I/O. It is available in java.io package. Java.io package provides classes for system input and output through files.
- A directory is a collection of files and subdirectories. A directory inside a directory is known as subdirectory.
- A file can be read either sequentially (continuously) or randomly (here and there).

Constructors:

- `File(path)`: Create File object for default directory (usually where program is located).
- `File(dirpath,fname)`: Create File object for directory path given as string.
- `File(dir, fname)`: Create File object for directory.

1. Sequential Access File:

- Sequential file access allows data to be read from a file or written to a file from beginning to end.

- It is not possible to read data starting in the middle of the file, nor is it possible to write data to the file starting in the middle using sequential methods.
- The input and output streams in Java are sequential.

Methods:

(i) `getName()`: Return the name of file.

(ii) `getParent()`: Return the name of Parent directory.

(iii) `length()`: Return the length of the file.

(iv) `isFile()`: Returns true if invoking object is file.

(v) `getPath()`: Returns relative path of the file.

(vi) `isDirectory()`: Returns true, if the invoking object is directory.

(vii) `isAbsolute()`: Returns true, if the path is absolute.

Program 4.12: Program for sequential access.

```
import java.io.File;
class FileClass
{
   public static void main(String args[])
   {
      File f = new File("sample.txt");
      System.out.println("is File =" +f.isFile());
      System.out.println("Name =" +f.getName());
      System.out.println("Size =" +f.length());
      System.out.println("Path =" +f.getPath());
      System.out.println("Parent =" +f.getParent());
      System.out.println("Hidden=" +f.isHidden());
   }
}
```

Output:

```
is File =true
Name = sample.txt
Size = 52
Path = root/sample.txt
Parent = root/
Hidden=false
```

2. Random Access File:

- A random access files, permit non-sequential, or random, access to a file's contents.
- One way to read or modify a file is to use the java.io.RandomAccessFile class. This class presents a model of files that is incompatible with the stream/reader/writer model described later in this chapter.
- It is possible to "seek" to any particular location within a file specifying the number of bytes.
  - RandomAccessFile(File obj, String access)
  - RandomAccessFile(String filename, String access)

Access:

(i) "r": File can be read, but not written.

(ii) "rw": File is opened in read-write mode.Existing file can be updated using this.

(iii) "rws": File is opened for read-write operations and every change to the file's data or metadata will be immediately written to the disk file.

(iv) "rwd": File is opened for read-write operations and every change to the file's data will be immediately written to the disk file.

Methods:

(i) `void seek(long pos):` Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

(ii) `int read():` Returns the next byte from the file stored in the low-order 8 bits of an int or –1 if at the end of the file.

(iii) `void write(byte buf[]):` Write all of byte array buf[].

Program 4.13: Program for random access file.

```
import java.io.*;
class RandomAccess
{
   public static void main(String args[ ]) throws Exception
   {
      RandomAccessFile f = new RandomAccessFile("sample.txt","rw");
      String s = "It is very easy to use the class RandomAccessFile.";
      byte [] buf = s.getBytes();  //convert to bytes
      longlen;
      f.write(buf);                // write to file
      f.seek(0);                   // move to start
      len = f.length( );           // find size
      System.out.println("Contents of file:");
```

```
        while(len>0)
        {
            System.out.print((char) f.read( ));
            len = len - 1;
        }
        System.out.println("\n\nSize of file:"+f.length( ));
    }
}
```

Output:

```
Contents of file:
It is very easy to use the class RandomAccessFile.
Size of file:50
```

4.5.1 Creating, Reading and Deleting Files

- To create a file in Java, we can use the createNewFile() method. This method returns a Boolean value true if the file was successfully created, and false if the file already exists.

  Syntax:

  ```
  public boolean createNewFile()
  ```

Program 4.14: Program for file creation.

```
import java.io.File;
public class FileDemo {
   public static void main(String[] args) {
      File f = null;
      boolean bool = false;
      try{
         // create new file
         f = new File("test.txt");
         // tries to create new file in the system
         bool = f.createNewFile();
         // prints
         System.out.println("File created: "+bool);
         // deletes file from the system
         f.delete();
         // delete() is invoked
         System.out.println("delete() method is invoked");
         // tries to create new file in the system
         bool = f.createNewFile();
```

```
            // print
            System.out.println("File created: "+bool);
         }catch(Exception e){
            e.printStackTrace();
         }
      }
   }
```

Output:

```
File created: false
delete() method is invoked
File created: true
```

Program 4.15: Program for file creation.

```
import java.io.File;  // Import the File class
import java.io.IOException; //Import the IOException class to handle errors
public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Output:

```
File created: filename.txt
```

- In the following program, we use the Scanner class to read the contents of the text file.

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException; // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files
public class ReadFile {
  public static void main(String[] args) {
```

```
      try {
        File myObj = new File("filename.txt");
        Scanner myReader = new Scanner(myObj);
        while (myReader.hasNextLine()) {
          String data = myReader.nextLine();
          System.out.println(data);
        }
        myReader.close();
      } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
      }
    }
  }
```

Output:

```
Files in Java might be tricky, but it is fun enough!
```

- In the following example, we use the FileWriter class together with its write() method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the close() method.

```
import java.io.FileWriter;   //Import the FileWriter class
import java.io.IOException; //Import the IOException class to handle errors
public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
            myWriter.write("Files in Java might be tricky, but it is fun
                                                                  enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Output:

```
Successfully wrote to the file.
```

- To delete a file in Java, use the delete() method. **[Oct. 17]**

```
import java.io.File;  // Import the File class
public class DeleteFile {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.delete()) {
      System.out.println("Deleted the file: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the file.");
    }
  }
}
```

Output:

```
Deleted the file: filename.txt
```

File I/O: **[Oct. 17]**

- A file is an ordered and named collection of bytes that has persistent storage.
- A file is a self contained piece of information available to the OS and its programs. A file is a collection of data or information.
- Java comes with many handy I/O classes to support the input and output through bytes stream and file system.
- To read and write files, Java provides various classes and methods.
- Java provides methods to read and write bytes because in Java, all the files are byte-oriented.
- The stream class FileInputStream and FileOutputStream are commonly used for file I/O.
- To create an input file, create an object using the following call:

  `FileInputStream(filename)`
- To create an output file, create an object using the following call:

  `FileOutputStream(filename)`
- While creating a file stream object using above calls, when a file does not exit or file cannot be created, it throws FileNotFoundException.
- The read and write operations can be performed on files using read( ) and write( ) methods on file stream objects.
- If in the program, no further operations will be on the file then it can be closed using close() method.

- Program 4.16 reads the content of a text file and displays it. In this program to read a filename readLine() method is used on BufferedReader object which may throws IOException.
- Creating an object of type FileInputStream opens file. The try/catch blocks are used to handle the exception while opening file. When EOF is encountered read( ) method returns –1.

Program 4.16: Reading a file.

```
import java.io.*;
class readfile
{
   public static void main(String args[]) throws IOException
   {
      int x;
      FileInputStream fis;
      BufferedReader br=new BufferedReader
                                    (new InputStreamReader(System.in));
      System.out.println("\nEnter filename to read: ");
      String fname=br.readLine();
      try
      {
      fis=new FileInputStream(fname);
      }
      catch(FileNotFoundException e)
      {
         System.out.println("File not found..");
      return;
      }
      //read from file
      do
      {
         x=fis.read();
         if (x!=-1)
         System.out.print((char)x);
      }
      while (x!=-1);
      fis.close();
   }
}
```

Output:

```
Enter a filename to read:
   abc.txt
Sachin Tendulkar is my favourite player
```

- Following Program 4.17 copies the content of source file and target file. The program takes filename through command line arguments. If source file does not exist or target file can't create then program catches the FileNotFoundException.
- Program uses read() method of FileInputStream object and return –1 when EOF is encountered. The ArrayIndexOutOfBoundException is caught to check for number of command-line arguments passed.

Program 4.17: Copy the content of a file into another. **[April 16]**

```
import java.io.*;
class dup
{
   public static void main(String args[]) throws IOException
   {
      int x;
      FileInputStream fis;
      FileOutputStream fos;
      try
      {
         try
            {
            fis=new FileInputStream(args[0]);
            }
            //source file not available
            catch(FileNotFoundException e)
            {
               System.out.println("File not found..");
            return;
            }
       try
         {
            fos=new FileOutputStream(args[1]);
         }
         //error in creating target file
         catch(FileNotFoundException e)
         {
            System.out.println("Can not create a file");
         return;
         }
      }
```

```
        //source/target filename not specified
        catch(ArrayIndexOutOfBoundsException e)
        {
           System.out.println("Error:Argument list");
           System.out.println("Usage:java dup sourcefile targetfile");
           return;
        }
        //read from sourcefile and write into targetfile
        do
        {
           x=fis.read();
        if (x!=-1) fos.write(x);
        }
        while (x!=-1);
        fis.close();
        fos.close();
     }
  }
```

4.5.2 Streams [April 16, Oct. 17, 18]

- All the input and output operation of file uses streams. A stream is an abstraction of I/O devices.
  - A stream is a sequence of data. Stream represents flow of data.
  - Stream is a logical entity that will be tied up with a data source or destination.
- The stream presents an object oriented and uniform interface between the program and the input/output devices.
- Every Java program has a source and destination. Source is in the form of keyboard, mouse, memory, disk, etc. Destination is screen, printer, memory, disk etc. There is an input and output streams.
- An input stream takes data from source and sends it to the program. An output stream takes data from the program and sends it to the destination.
- A flow of data is often referred to as a data stream. A stream is an ordered sequence of bytes that has a SOURCE (input stream) or a DESTINATION (output stream).
- In simple terms, a stream can be defined as "a logical device that represents the flow of a sequence of characters".
- Programs can get inputs from a data source by reading a sequence of characters from the input stream. Similarly, programs can produce outputs by writing a sequence of characters onto an output stream.
- Stream is a logical entity that will be tied up with a data source or destination. Stream represents flow of data. The representation of stream is as shown in Fig. 4.2.

Fig. 4.2: Stream Representation

Standard Streams:

- Standard Streams are a feature provided by many operating systems. By default, they read input from the keyboard and write output to the display. They also support I/O operations on files.
  1. **Standard Input:** Accessed through **System.in** which is used to read input from the keyboard.
  2. **Standard Output:** Accessed through **System.out** which is used to write output to be display.
  3. **Standard Error:** Accessed through **System.err** which is used to write error output to be display.
- Fig. 4.3 shows classification of Streams.

Fig. 4.3: Classification of streams

- In general, streams are classified into two types known as character streams and the byte streams.

1. Byte Stream:

- It provides continent means of handling input and output of byte.
  - **(i) Input Stream (Reading information into a program):** A program uses an input stream to read data from a source, one item at a time.
  - **(ii) Output Stream (Writing information from a program):** A program uses an output stream to write data to a destination, one item at time.

- Working of Java OutputStream and InputStream by the Fig. 4.4.

Fig. 4.4: Working of java output stream and input stream

2. Character Stream:

- It provides continent means of handling input and output of character.

(i) Reader: are operated on *characters* for reading.

Fig. 4.5: Reader

(ii) Writer: are operated on *characters* for writing, respectively.

Fig. 4.6: Writer

- Java's stream based I/O has four abstract classes namely, InputStream, OutputStream, Reader and Writer.
- Inputstream and OutputStream deal with bytes i.e. read/write bytes. Reader and Writer are designed for character streams.

4.6 INPUT-OUTPUT STREAMS

- In Java, streams are the sequence of data (bits) that are read from the source and written to the destination.

- An input stream is used to read data from the source. An output stream is used to write data to the destination.

Fig. 4.7

- An input stream reads data from the source file and sends it to the program while output stream takes data from the program and sends it to the destination file.
- Depending upon the data a stream holds, it can be classified into Byte Stream and Character Stream.
- Byte stream is used to read and write a single byte (8 bits) of data. All byte stream classes are derived from base abstract classes called InputStream and OutputStream.
- Character stream is used to read and write a single character of data. All the character stream classes are derived from base abstract classes Reader and Writer.

4.6.1 InputStream [April 17, Oct. 17]

- java.io.InputStream is an abstract class that contains the basic methods for reading raw bytes of data from a stream.
- Although InputStream is an abstract class, many methods in the class library are only specified to return an InputStream.
- Java application uses an input stream to read data from a source. It may be a file, an array, peripheral device or socket.

Fig. 4.8: Input stream

- Fig. 4.9 shows classes of InputStream.

Fig. 4.9: Classes of input stream

- The InputStream class defines following different methods to perform input operation:
  - **read():** it reads a byte from input stream.
  - **read(byte arr[]):** it reads on array of bytes into arr.
  - **read (byte arr[], int s, int n):** reads in bytes in arr.start position is S.
  - **available():** returns available number of bytes in the input.
  - **reset():** moves to the beginning of the stream.
  - **close():** the input stream is closed.
- The class DataInputStream implements the interface DataInput. All the methods of it are listed below:

```
readShort()        readFloat()        readChar()
readInt()          readDouble()       readBoolean()
readLong()         readLine()
```

Program 4.18: Program to use of InputStream to count the total number of bytes in a file if no file is specified.

```
import java.io.*;
class count
{
   public static void main(String args[])
   throws IOException
   {
      InputStream in;
      if (args.length = 0)
         in = System.in;
      else
         in = new FileInputStream (args[0]);
         int total = 0;
         while (in.read() != - 1)
            total ++;
         System.out.println(total + "bytes");
   }
}
```

- This program takes filename from command line. If a filename is not provided, the standard inputstream i.e. system in is used. The while loop counts the total number of bytes in a file. At the end, result will be printed.

Output:

```
212 bytes
```

4.6.2 OutputStream

- Java application uses an output stream to write data to a destination. It may be a file, an array,peripheral device or socket.

Fig. 4.10: Output stream

- Fig. 4.11 shows various classes of OutputStream.

Fig. 4.11: Classes of output stream

- The OutputStream class defines following different methods to perform output operation.
  - **write():** it write a byte to output stream.
  - **write(byte arr[]):** it writes an array of bytes in arr to the output stream.
  - **write(byte arr[], int s, int n):** writes n byte from arr to output screen with start positions.
  - **close():** the output stream is closed.
- The class DataOutputStream implements the interface DataOutput. All the methods of it are listed below:

```
writeShort()        writeInt()
writeLong()         writeFloat()
writeDouble()       writeLine()
writeChar()         writeBoolean()
```

Program 4.19: Program to copies its input to its output by translating one particular byte value to a different one.

```
import java.io.*;
class trbyte
{
   public static void main(String args[])
```

```
    throws IOException
    {
        byte from = (byte)args[0].charAt(0);
        byte to = (byte)args[1].charAt(0);
        int b;
        while ((b = System.in.read() != - 1)
        System.out.write(b == from? to: b);
    }
}
```

- We should invoke this program as,

  `java trbyte b B`

  and entered the text abracadabra !
- We will get the output:

  `aBracadaBra !`
- To summarize Byte stream classes:

Byte Stream Class	Purpose
DataInputStream	To read java standard data types.
DataOutputStream	To write java standard data types.
BufferedInputStream	For buffered input operations.
BufferedOutputStream	For buffered output operations.
FileInputStream	To read data from the file.
FileOutputStream	To write data to the file.
ByteArrayInputStream	To read data from byte array.
ByteArrayOutputStream	To write data into byte array.
PrintStream	It contains print() and println().

4.6.3 FileInputStream

- This stream is used for reading data from the files. Objects can be created using the keyword new and there are several types of constructors available.
- Following constructor takes a file name as a string to create an input stream object to read the file:

  ```
  InputStream f = new FileInputStream("/java/nirali");
  ```
- A FileInputStream obtains input bytes from a file. It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader.
- It should be used to read byte-oriented data. For example, to read image etc.

Program 4.20: Program for FileInputStream.

```
import java.io.*;
class SimpleRead{
   public static void main(String args[]){
   try{
      FileInputStream fin=new FileInputStream("abc.txt");
      int i;
      while((i=fin.read())!=-1)
         System.out.println((char)i);
   fin.close();
   }catch(Exception e){system.out.println(e);}
 }
}
```

Output:

```
Sachin is my favourite player
```

4.6.4 FileOutputStream

- It is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.
- A FileOutputStream is an output stream for writing data to a file.
- If we have to write primitive values then use FileOutputStream. Instead, for character-oriented data, prefer FileWriter. But we can write byte-oriented as well as character-oriented data.
- FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.
- Constructors are:
  1. `FileOutputStream(String filePath)`
  2. `FileOutputStream(File fileObj)`
  3. `FileOutputStream(String filePath, boolean append)`
  4. `FileOutputStream(File fileObj, boolean append)`

Program 4.21: Program for FileOutputStream.

```
import java.io.*;
class Test
{
   public static void main(String args[]){
   try{
   FileOutputstream fout=new FileOutputStream("abc.txt");
```

```
        String s="Sachin Tendulkar is my favourite player";
        byte b[]=s.getBytes();
        fout.write(b);
        fout.close();
        System.out.println("success...");
        }catch(Exception e){system.out.println(e);}
        }
    }
```

Output:

```
success...
```

4.6.5 BufferedInputStream

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:
  1. When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
  2. When a BufferedInputStream is created, an internal buffer array is created.

Constructors:

`BufferedInputStream(InputStream IS)`	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
`BufferedInputStream(InputStream IS, int size)`	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

Methods:

`int available()`	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
`int read()`	It read the next byte of data from the input stream.
`int read(byte[] b, int off, int ln)`	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.

contd. …

`void close()`	It closes the input stream and releases any of the system resources associated with the stream.
`void reset()`	It repositions the stream at a position the mark method was last called on this input stream.
`void mark(int readlimit)`	It sees the general contract of the mark method for the input stream.
`long skip(long x)`	It skips over and discards x bytes of data from the input stream.
`boolean markSupported()`	It tests for the input stream to support the mark and reset methods.

Program 4.22: Program to read data of file using BufferedInputStream.

```
import java.io.*;
public class BufferedInputStreamExample
{
   public static void main(String args[])
   {
      try
      {
         FileInputStream fin=new FileInputStream("testout.txt");
         BufferedInputStream bin=new BufferedInputStream(fin);
         int i;
         while((i=bin.read())!=-1)
         {
            System.out.print((char)i);
         }
         bin.close();
         fin.close();
      }catch(Exception e){System.out.println(e);}
   }
}
```

4.6.6 BufferedOutputStream

- Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data.
- It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class.

Constructors:

`BufferedOutputStream(OutputStream os)`	It creates the new buffered output stream which is used for writing the data to the specified output stream.
`BufferedOutputStream(OutputStream os, int size)`	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

Methods:

`void write(int b)`	It writes the specified byte to the buffered output stream.
`void write(byte[] b, int off, int len)`	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
`void flush()`	It flushes the buffered output stream.

Program 4.23: Program to write data of file using BufferedOutputStream.

```
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{
   FileOutputStream fout=new FileOutputStream("testout.txt");
   BufferedOutputStream bout=new BufferedOutputStream(fout);
   String s="Welcome to javaTpoint.";
   byte b[]=s.getBytes();
   bout.write(b);
   bout.flush();
   bout.close();
   fout.close();
   System.out.println("success");
   }
}
```

Output:

```
Success
testout.txt
Welcome to javaTpoint.
```

4.6.7 DataInputStream

- Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.
- Java application generally uses the data output stream to write data that can later be read by a data input stream.
- Following is the constructor to create an InputStream –

```
InputStream in = new DataInputStream(InputStream in);
```

Methods:

`int read(byte[] b)`	It is used to read the number of bytes from the input stream.
`int read(byte[] b, int off, int len)`	It is used to read len bytes of data from the input stream.
`int readInt()`	It is used to read input bytes and return an int value.
`byte readByte()`	It is used to read and return the one input byte.
`char readChar()`	It is used to read two input bytes and returns a char value.
`double readDouble()`	It is used to read eight input bytes and returns a double value.
`boolean readBoolean()`	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
`int skipBytes(int x)`	It is used to skip over x bytes of data from the input stream.
`String readUTF()`	It is used to read a string that has been encoded using the UTF-8 format.
`void readFully(byte[] b)`	It is used to read bytes from the input stream and store them into the buffer array.
`void readFully(byte[] b, int off, int len)`	It is used to read len bytes from the input stream.

Program 4.24: Program to read data of file using DataInputStream.

```
import java.io.*;
public class DataStreamExample {
  public static void main(String[] args) throws IOException {
    InputStream input = new FileInputStream("testout.txt");
```

```
        DataInputStream inst = new DataInputStream(input);
        int count = input.available();
        byte[] ary = new byte[count];
        inst.read(ary);
        for (byte bt: ary) {
          char k = (char) bt;
          System.out.print(k+"-");
        }
      }
    }
```

Output:

- Here, we are assuming that you have following data in "testout.txt" file:

```
JAVA
J-A-V-A-V-A
```

4.6.8 DataOutputStream

- Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.
- Java application generally uses the data output stream to write data that can later be read by a data input stream.

Methods:

`int size()`	It is used to return the number of bytes written to the data output stream.
`void write(int b)`	It is used to write the specified byte to the underlying output stream.
`void write(byte[] b, int off, int len)`	It is used to write len bytes of data to the output stream.
`void writeBoolean(boolean v)`	It is used to write Boolean to the output stream as a 1-byte value.
`void writeChar(int v)`	It is used to write char to the output stream as a 2-byte value.
`void writeChars(String s)`	It is used to write string to the output stream as a sequence of characters.
`void writeByte(int v)`	It is used to write a byte to the output stream as a 1-byte value.

contd. ...

`void writeBytes(String s)`	It is used to write string to the output stream as a sequence of bytes.
`void writeInt(int v)`	It is used to write an int to the output stream
`void writeShort(int v)`	It is used to write a short to the output stream.
`void writeShort(int v)`	It is used to write a short to the output stream.
`void writeLong(long v)`	It is used to write a long to the output stream.
`void writeUTF(String str)`	It is used to write a string to the output stream using UTF-8 encoding in portable manner.
`void flush()`	It is used to flushes the data output stream.

Program 4.25: Program to writing data of file using DataOutputStream.

```
import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Succcess...");
    }
}
```

Output:

```
Success
testout.txt
A
```

4.7 READER AND WRITER

- The abstract classes for reading and writing streams of character are Reader and Writer.
- Following classes are some of the important character stream classes:

Character Stream Class	Purpose
`InputStreamReader`	Translates bytes to characters.
`OutputStreamWriter`	Translates character to bytes.
`BufferedReader`	For buffered input character streams.
`BufferedWriter`	For buffered output character streams.

contd. ...

`FileReader`	To read data from file.
`FileWriter`	To write data to the file.
`CharArrayReader`	To read from char array.
`CharArrayWriter`	To writes to char array.
`StringReader`	To read from string.
`StringWriter`	To write to string.
`PrintWriter`	It contains print() and println().

- Reader stream classes are used to read character from files. Reader stream classes are similar to input stream classes but difference is that input streams uses bytes whereas reader streams uses characters as the basic unit of read operations.
- Writer stream classes are use to write character into files. Writer stream classes are similar to output stream classes but difference is that output streams are used to write bytes whereas writer streams are used to write characters.

4.7.1 Reader [Oct. 16]

- Java Reader is an abstract class for reading character streams. Reader is the base class of the character oriented input streams. It provides methods to read a sequence of Unicode characters.

Fig. 4.12: Classes of Reader

- It is character oriented input operations. The following table shows all the Reader methods and its description.
- It is an abstract class for character-input streams.
- Java provides the standard I/O facilities for reading text from either the file or the keyboard on the command line.
- The Reader class is used for this purpose that is available in the java.io package. It acts as an abstract class for reading character streams.
- Fig. 4.12 shows classes of Reader. The methods for Reader are given in following table:

Methods	Description
`int read()`	Reads a single character.
`int read(char[]buff)`	Reads character into the array buff.
`int read(char[]buff, int index, int length)`	Reads character into a portion of the array buff starting from index upto the length given.
`Boolean ready()`	Returns true if the stream is ready for reading.
`long skip(long n)`	Skips n characters.
`void reset()`	Resets the stream.
`void mark(int readLimit)`	Marks the current position of the stream.
`Boolean markSupported()`	Returns true if the mark operation is executed.
`static void close()`	Closes the stream.

- The implementation of Reader requires that a subclass provide a implementation of the read method that reads into a char array and the close method.

Difference between Reader and InputStream:

Reader	InputStream
1. Reads into a char array.	1. Reads in a single byte.
2. Subclass must implements the abstract close method.	2. Close() is usually overridden.
3. It has a ready method which tells us if there is any data.	3. It has available method which tells us how much data is available to read.

- Program for counts number of whitespace characters in character stream.

Program 4.26:

```
import java.io.*;
class space1
{
   public static void main(String args[])
      throws IOException
   {
      Reader in;
      if (args.length == 0)
      in = new InputStreamReader(System.in);
   else
      in = new FileReader(args[0]);
   int ch;
```

```
        int total;
        int space = 0;
        for (total=0; (ch = in.read())!= - 1; total++)
           {
              if (Character.isWhitespace ((char)ch))
              space ++;
           }
           system.out.println(total + "chars" + space + "spaces");
        }
    }
```

Output:

```
If the same program name given 453 chars 111 spaces.
```

4.7.2 Writer

- Writer is an abstract class for writing to character streams. It is an abstract class. It provides a stream analogous of OutputStream but it uses characters and not bytes.
- The data can be written into a file, memory buffer, or can be displayed in standard output device.
- It is an abstract class for character-output streams. Fig. 4.13 shows classes and writer.

Fig. 4.13: Classes of writer

- The following table shows methods of class writer:

Methods	Description
`void write(int p)`	Writes lower 16 bits of p into the stream.
`void write(char buff[])`	Writes contents of buff into the stream.
`void write(char buff[], int index, int length)`	Writes a portion of buff into the stream.
`void write(string S)`	Writes the string S into the stream starting from index and writes n characters.
`void close()`	Closes the stream.
`void flush()`	Writes buffered data to the device.

- The subclass of writer implements the array writing variant of write, the close and the flush method.

Program 4.27: Program for Writer.

```
import java.io.*;
public class WriterExample {
   public static void main(String[] args) {
     try {
         Writer w = new FileWriter("output.txt");
         String content = "I love my country";
         w.write(content);
         w.close();
         System.out.println("Done");
     } catch (IOException e) {
         e.printStackTrace();
     }
   }
}
```

Output:

```
Done
testout.txt
I love my country
```

4.7.3 FileReader [Oct. 16]

- FileReader class is used to read data from the file. It translates bytes from a file into a character stream.

- Constructors are:
  1. `FileReader(String path)`
  2. `FileReader(File obj)`

Program 4.28: Program for FileReader.

```
import java.io.FileReader;
import java.io.File;
class FileRead
{
   public static void main(String args[ ])
   throws Exception
   {
      File f = new File("sample.txt");
      if(f.exists( ))
      {
         FileReader fr = new FileReader(f);
         int ch;
         while((ch = fr.read()) != -1)
         System.out.print((char) ch);
         fr.close( );
      }
   }
}
```

Output:

```
It is very easy to use the class in Java.
```

4.7.4 FileWriter

- FileWriter class is used to write character into the file.
- Constructors are:
  1. `FileWriter(String path)`
  2. `FileWriter(String path, boolean append)`
  3. `FileWriter(File obj)`
  4. `FileWriter(File obj, boolean append)`

Program 4.29: Program for FileWriter.

```
import java.io.FileWriter;
class FileWrite
{
   public static void main(String args[ ])throws Exception
   {
```

```
        String str = "I would like to learn AdvancedJava Programming also";
        char buf[] = str.toCharArray( );
        int len = buf.length;
        FileWriter f = new FileWriter("sample1.txt");
        f.write(buf);
        f.close( );
        FileWriter f3 = new FileWriter("sample2.txt");
        f3.write(buf,buf.length-buf.length/4,buf.length/4);
        f3.close( );
    }
}
```

Output:

Open sample1 and sampel2.txt files

4.7.5 BufferedReader

- Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method.
- It makes the performance fast. It inherits Reader class.

Constructors:

Constructor	Description
`BufferedReader(Reader rd)`	It is used to create a buffered character input stream that uses the default size for an input buffer.
`BufferedReader(Reader rd, int size)`	It is used to create a buffered character input stream that uses the specified size for an input buffer

Methods:

Method	Description
`int read()`	It is used for reading a single character.
`int read(char[] cbuf, int off, int len)`	It is used for reading characters into a portion of an array.
`boolean markSupported()`	It is used to test the input stream support for the mark and reset method.
`String readLine()`	It is used for reading a line of text.
`boolean ready()`	It is used to test whether the input stream is ready to be read.
`long skip(long n)`	It is used for skipping the characters.
`void reset()`	It repositions the stream at a position the mark method was last called on this input stream.
`void mark(int readAheadLimit)`	It is used for marking the present position in a stream.
`void close()`	It closes the input stream and releases any of the system resources associated with the stream.

Program 4.30: Program for reading the data from the text file **testout.txt** using Java BufferedReader.

```
import java.io.*;
public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
      FileReader fr=new FileReader("testout.txt");
      BufferedReader br=new BufferedReader(fr);
      int i;
      while((i=br.read())!=-1){
      System.out.print((char)i);
      }
      br.close();
      fr.close();
   }
}
```

- Here, we are assuming that we have te data (Welcome to javaTpoint.) in "testout.txt" file.

Output:

```
Welcome to javaTpoint.
```

4.7.6 BufferedWriter

- Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class.
- The buffering characters are used for providing the efficient writing of single arrays, characters and strings.

Constructors:

`BufferedWriter(Writer wrt)`	It is used to create a buffered character output stream that uses the default size for an output buffer.
`BufferedWriter(Writer wrt, int size)`	It is used to create a buffered character output stream that uses the specified size for an output buffer.

Methods:

`void newLine()`	It is used to add a new line by writing a line separator.
`void write(int c)`	It is used to write a single character.
`void write(char[] cbuf, int off, int len)`	It is used to write a portion of an array of characters.
`void write(String s, int off, int len)`	It is used to write a portion of a string.
`void flush()`	It is used to flushes the input stream.
`void close()`	It is used to closes the input stream

Program 4.31: Program for writing the data from the text file testout.txt using Java BufferedReader.

```
import java.io.*;
public class BufferedWriterExample {
   public static void main(String[] args) throws Exception {
      FileWriter writer = new FileWriter("testout.txt");
      BufferedWriter buffer = new BufferedWriter(writer);
      buffer.write("Welcome to javaTpoint.");
      buffer.close();
      System.out.println("Success");
      }
}
```

Output:

```
Success
testout.txt:
Welcome to javaTpoint.
```

4.7.7 InputStreamReader

- An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset.
- The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

Constructors:

Constructor Name	Description
`InputStreamReader(InputStream in)`	It creates an InputStreamReader that uses the default charset.
`InputStreamReader(InputStream in, Charset cs)`	It creates an InputStreamReader that uses the given charset.
`InputStreamReader(InputStream in, CharsetDecoder dec)`	It creates an InputStreamReader that uses the given charset decoder.
`InputStreamReader(InputStream in, String charsetName)`	It creates an InputStreamReader that uses the named charset.

Methods:

Modifier and Type	Method	Description
void	`close()`	It closes the stream and releases any system resources associated with it.
String	`getEncoding()`	It returns the name of the character encoding being used by this stream.
int	`read()`	It reads a single character.
int	`read(char[] cbuf, int offset, int length)`	It reads characters into a portion of an array.
boolean	`ready()`	It tells whether this stream is ready to be read.

Program 4.32: Program for InputStreamReader.

```
public class InputStreamReaderExample {
   public static void main(String[] args) {
    try  {
         InputStream stream = new FileInputStream("file.txt");
         Reader reader = new InputStreamReader(stream);
         int data = reader.read();
```

```
            while (data != -1) {
               System.out.print((char) data);
               data = reader.read();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
I love my country
The file.txt contains text "I love my country" the InputStreamReader
reads Character by character from the file
```

4.7.8 OutputStreamWriter

- OutputStreamWriter is a class which is used to convert character stream to byte stream, the characters are encoded into byte using a specified charset.
- The write() method calls the encoding converter which converts the character into bytes.
- The resulting bytes are then accumulated in a buffer before being written into the underlying output stream. The characters passed to write() methods are not buffered.
- We optimize the performance of OutputStreamWriter by using it with in a BufferedWriter so that to avoid frequent converter invocation.

Constructors:

`OutputStreamWriter(OutputStream out)`	It creates an OutputStreamWriter that uses the default character encoding.
`OutputStreamWriter(OutputStream out, Charset cs)`	It creates an OutputStreamWriter that uses the given charset.
`OutputStreamWriter(OutputStream out, CharsetEncoder enc)`	It creates an OutputStreamWriter that uses the given charset encoder.
`OutputStreamWriter(OutputStream out, String charsetName)`	It creates an OutputStreamWriter that uses the named charset.

Methods:

Modifier and Type	Method	Description
void	`close()`	It closes the stream, flushing it first.
void	`flush()`	It flushes the stream.
String	`getEncoding()`	It returns the name of the character encoding being used by this stream.
void	`write(char[] cbuf, int off, int len)`	It writes a portion of an array of characters.
void	`write(int c)`	It writes a single character.
void	`write(String str, int off, int len)`	It writes a portion of a string.

Program 4.33: Program for OutputStreamReader.

```
public class OutputStreamWriterExample {
   public static void main(String[] args) {
   try {
         OutputStream outputStream = new FileOutputStream("output.txt");
         Writer outputStreamWriter = new OutputStreamWriter(outputStream);
          outputStreamWriter.write("Hello World");
          outputStreamWriter.close();
      } catch (Exception e) {
         e.getMessage();
      }
   }
}
```

Output:

```
output.txt file will contains text  "Hello World"
```

PROGRAMS

Program 1: Program for try-catch statement to handle the exception.

```
public class Demoexception
{
   public static void main(String args[])
   {
   try
   {
       int data=100/0;  // raise exception
   }
```

```
        catch(ArithmeticException e)  //catch the exception
        {
             System.out.println(e);
        }
        }
    }
```

Output:

```
java.lang.ArithmeticException: / by zero
```

Program 2: Program for multiple catch to handle the exception.

```
public class Demomultiplecatch
{
    public static void main(String[] args)
    {
        try
        {
            int array[] = new int[5];
            array[5]=77/0;
        }
        catch(ArithmeticException e)
        {
            System.out.println("Arithmetic Exception " +e);
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("ArrayIndexOutOfBounds Exception "+e);
        }
        catch(Exception e)
        {
            System.out.println("Exception "+e);
        }
    }
}
```

Output:

```
Arithmetic Exception java.lang.ArithmeticException: / by zero
```

Program 3: Program for throw keyword.

```
import java.util.Scanner;
public class Testvaccine
{
   static void checkAge(int age)
   {
      if (age < 18)
      {
         throw new ArithmeticException("No Covid Vaccine - You must be at
                                                least 18 years old.");
      }
      else
      {
         System.out.println("Covid Vaccine Granted !");
      }
   }
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);
      System.out.print("Enter the person age: ");
      int age = in.nextInt();
      checkAge(age);
   }
}
```

Output:

```
Enter the person age: 15
Exception in thread "main" java.lang.ArithmeticException: No Covid Vaccine
- You must be at least 18 years old.
   at Testvaccine.checkAge(Testvaccine.java:11)
   at Testvaccine.main(Testvaccine.java:26)
javac Testvaccine.java
java Testvaccine
Enter the person age: 25
Covid Vaccine Granted!
```

Program 4: Program for ArrayIndexOutofBound.

```
public class Demofinally
{
   public static void main(String[] args)
   {
      try
      {
         int[] myarr = {11, 22, 33, 55 };
         System.out.println(myarr[10]);
      }
      catch (Exception e)
      {
         System.out.println("Error"+e);
      }
      finally
      {
         System.out.println("Finally Block");
      }
   }
}
```

Output:

```
Errorjava.lang.ArrayIndexOutOfBoundsException: 10
Finally Block
```

Program 5: Program to find the prime numbers 1 to 100 using throws.

```
import java.util.Scanner;
import java.io.*;
class Exprimeno
{
   public static void main (String[] args) throws IOException
   {
      Scanner scanner = new Scanner(System.in);
      int i =0;
      int n1 =0;
      System.out.println("Enter the Limit:");
      int lim = scanner.nextInt();
      System.out.println("Prime numbers from 1 to n are:");
```

```
        for (i = 1; i <= lim; i++)
        {
           int cnt = 0;
           for(n1 = i; n1 >= 1; n1--)
           {
              if(i%n1==0)
              {
                 cnt = cnt + 1;
              }
           }
           if (cnt ==2)
           {
              System.out.println( " " +i);
           }
        }
     }
  }
```

Output:

```
Enter the Limit:
50
Prime numbers from 1 to n are:
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
```

Program 6: Program for File class.

```
import java.io.*;
public class Filecreate
{
   public static void main(String[] args)
   {
      try
      {
         File fp = new File("abc.txt");
         if (fp.createNewFile())
         {
            System.out.println("File created: " + fp.getName());
         }
         else
         {
            System.out.println("File already exists.");
         }
      }
      catch (IOException e)
      {
         System.out.println("An error occurred.");
         e.printStackTrace();
      }
   }
}
```

Output:

```
File created: abc.txt
```

Program 7: Program for File class with readable and writable.

```
import java.io.*;
public class Fileinfo
{
   public static void main(String[] args)
   {
      File f = new File("xyz.txt");
```

```
        if (f.exists())
        {
           System.out.println("File name: " + f.getName());
           System.out.println("Absolute path: " + f.getAbsolutePath());
           System.out.println("Writeable: " + f.canWrite());
           System.out.println("Readable: " + f.canRead());
           System.out.println("File size in bytes:" + f.length());
        }
        else
        {
           System.out.println("Sorry !!! The file does not exist.");
        }
     }
  }
```

Output:

```
Writeable: true
Readable: true
File size in bytes:8
```

Program 8: Program to write the string into file using Writer.

```
import java.io.*;
public class Exwriter
{
   public static void main(String[] args)
   {
      try
      {
         Writer w = new FileWriter("test.txt");
         String s = "Covid Vaccine";
         w.write(s);
         w.close();
         System.out.println("Successfully Done...");
      }
      catch (IOException e)
      {
         e.printStackTrace();
      }
   }
}
```

Output:

```
Successfully Done...
```

Program 9: Program for reading data from console by InputStreamReader and BufferedReader.

```
import java.io.*;
public class Ex_bufferedreader
{
   public static void main(String args[])throws Exception
   {
      InputStreamReader in = new InputStreamReader(System.in);
      BufferedReader br=new BufferedReader(in);
      System.out.println("Enter your name");
      String name=br.readLine();
      System.out.println("Hello "+name);
   }
}
```

Output:

```
Enter your name rahat
Hello rahat
```

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. Which of these packages contain all the Java's built in exceptions?
   (a) java.io (b) java.util
   (c) java.lang (d) java.net
2. Which of these classes is used to create user defined exception?
   (a) java.lang (b) Exception
   (c) RunTime (d) System
3. What will be the output of the program?

```
public class X
{
    public static void main(String [] args)
    {
        try
        {
            badMethod();
            System.out.print("A");
        }
```

```
            catch (Exception ex)
            {
                System.out.print("B");
            }
            finally
            {
                System.out.print("C");
            }
            System.out.print("D");
        }
        public static void badMethod() {}
    }
```

(a) AC (b) BC
(c) ACD (d) ABCD

4. Following which statement is true,
   (a) catch(X x) can catch subclasses of X where X is a subclass of Exception.
   (b) The Error class is a RuntimeException.
   (c) Any statement that can throw an Error must be enclosed in a try block.
   (d) All of mentioned
5. When does Exceptions in Java arises in code sequence?
   (a) Run Time (b) Compilation Time
   (c) Can Occur Any Time (d) None of the mentioned
6. Which of these keywords must be used to monitor for exceptions?
   (a) try (b) finally
   (c) throw (d) catch
7. Which of the following handles the exception when a catch is not used?
   (a) finally (b) throw handler
   (c) default handler (d) java run time system
8. Which of these exception is thrown in cases when the file specified for writing is not found?
   (a) FileNotFoundException (b) FileException
   (c) IOException (d) FileInputException
9. Which of these values is returned by read() method is end of file (EOF) is encountered?
   (a) Null (b) 0
   (c) 1 (d) −1

10. Which of the following classes can catch all exceptions which cannot be caught?
    (a) RuntimeException (b) Error
    (c) ParentException (d) Exception
11. Which are the valid ways to create DataInputStream streams?
    (a) new DataInputStream();
    (b) new DataInputStream("in.dat", "r");
    (c) new DataInputStream(new FileInputStream("in.dat");
    (d) new DataInputStream("in.dat")
12. Which of the following strings can be used as mode strings for creating a RandomAccessFile object?
    (i) "r" (ii) "w"
    (iii) "rw" (iv) "wr"
    (v) "0"
    (a) (i), (ii), (iii) & (v) (b) (i), (ii) & (iii)
    (c) (i) & (iii) (d) (ii), (iv) & (v)
13. DataInput is _____.
    (a) an abstract class defined in java.io.
    (b) a class we can use to read primitive data types.
    (c) an interface that defines methods to open files.
    (d) an interface that defines methods to read primitive data types.
14. The try, catch, finally, throw and throws keywords in java are related to,
    (a) Event handling (b) string handling
    (c) exception handling (d) inheritance
15. What is subclass in java?
    (a) A subclass is a class that extends another class
    (b) A subclass is a class declared inside a class
    (c) Both (a) and (b)
    (d) None of mentioned
16. If class B is subclassed from class A then which is the correct syntax _____.
    (a) class B:A{} (b) class B extends A{}
    (c) class B extends class A{} (d) class B implements A{}
17. Which of these is a type of stream in Java?
    (a) Integer stream (b) Short stream
    (c) Byte stream (d) Long stream

18. Which of these classes is used to read characters and strings in Java from console?
    (a) BufferedReader (b) StringReader
    (c) BufferedStreamReader (d) InputStreamReader
19. Which of these classes is used to read from a file?
    (a) InputStream (b) BufferedInputStream
    (c) FileInputStream (d) BufferedFileInputStream
20. Which of these method(s) is/are used for writing bytes to an outputstream?
    (a) put() (b) print() and write()
    (c) printf() (d) write() and read()
21. Which occur naturally when we create, write and develop programs?
    (a) Errors (b) Exceptions
    (c) Mistakes (d) All of mentioned
22. Which exceptions occurs at the compile time?
    (a) unchecked (b) checked
    (c) error (d) All of mentioned
23. All the errors and exceptions in Java are the subclasses of which class,
    (a) Object (b) String
    (c) Throwable (d) Scanner
24. Which occurs during the execution of the program?
    (a) unchecked (b) checked
    (c) error (d) All of mentioned
25. A catch block always follows the _____ block.
    (a) try (b) throw
    (c) throws (d) finally
26. Which block is a block where we can handle the exceptions?
    (a) try (b) finally
    (c) throw (d) catch
27. Which is an unexpected event that occurs during program execution?
    (a) error (b) exception
    (c) mistake (d) All of mentioned
28. Which statement declares the type of exception that we try to catch?
    (a) try (b) throw
    (c) catch (d) finally

29. Which block is used to handle exceptions in Java?
    (a) try-catch (b) try-throw
    (c) try-throws (d) try-catch-throw
30. When an exception occurs, the finally block is executed _____ the try... catch block.
    (a) before (b) after
    (c) before throw (d) after throws
31. Which the sequence of data that are read from the source and written to the destination?
    (a) file (b) stream
    (c) I/O (d) All of mentioned
32. Which stream is used to read and write a single byte (8 bits) of data.?
    (a) byte (b) character
    (c) InputStream (d) OutputStream
33. Which stream is used to read and write a single character of data?
    (a) byte (b) character
    (c) InputStream (d) OutputStream
34. Which stream is used for reading data from the files?
    (a) FileInputStream (b) FileOutputStream
    (c) BufferInputStream (d) DataInputStream
35. Java provides the following three standard streams,
    (a) system.in (b) system.out
    (c) system.err (d) All of mentioned
36. Which stream class is used for convert data in bytes into data in characters?
    (a) InputStreamReader (b) InputStreamWriter
    (c) InputStreamBuffer (d) DataInputStream

Answers

1. (c)	2. (b)	3. (c)	4. (a)	5. (a)	6. (a)	7. (c)	8. (a)	9. (d)	10. (b)
11. (c)	12. (c)	13. (d)	14. (c)	15. (a)	16. (b)	17. (c)	18. (a)	19. (c)	20. (b)
21. (a)	22. (b)	23. (c)	24. (a)	25. (a)	26. (d)	27. (b)	28. (c)	29. (a)	30. (b)
31. (b)	32. (a)	33. (b)	34. (a)	35. (d)	36. (a)				

Q.II Fill in the Blanks:

1. Exception generated in try block is caught in _____ block.
2. _____ keywords must be used to monitor for exceptions.

3. _____ exception is thrown when divide by zero statement executes.
4. BufferedWriter can be used along with _____ to improve speed of execution.
5. _____ class obtains input bytes from a file.
6. In character stream I/O, a single read/write operation performs _____ bytes read/write at a time.
7. An exception is a condition that is caused by a _____ error in the program
8. Dividing an integer by zero is a _____ error
9. _____ streams provide functionally for threads to communicate and exchange data between them.
10. The class _____ is a subclass of object class which can be used for breaking up a stream of text from an input text file into meaningful pieces.
11. Combining two or more input streams into a single input stream can be achieved using the _____ class.
12. The _____ package contains a large number of stream classes that provide capabilities for processing all types of data.
13. _____ class is related to all the exceptions that are explicitly thrown.
14. The _____ method of the BufferedReader class is used for reading lines of text from the console, the file or other input streams.
15. The _____ class implements the DataInput and DataOutput interfaces for performing I/O using the primitive data types.
16 The Java _____ keyword is used to explicitly throw a single exception.
17. An _____ exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions.
18. We use the _____ keyword in the method declaration to declare the type of exceptions that might occur within it.
19. The _____ block follows a try block or a catch block.
20. A _____ is a sequence of data.
21. All _____ stream classes are derived from base abstract classes called InputStream and OutputStream.
22. The _____ class of the java.io package is an abstract superclass that represents a stream of characters.
23. The _____ class of the java.io package can be used to write data (in characters) to files.
24. DataInputStream allows an application to read _____ data from the input stream in a machine-independent way.
25. The try block within a try block is known as _____ try block.
26. A _____ is a collection of related data or information.

27. To delete a file in Java, use the _____ method.
28. Java performs I/O through _____.

Answers

1. catch	2. try	3. ArithmeticException	4. FileWriter
5. FileInputStream	6. Two	7. Execution	8. Runtime
9. Piped	10. Stream Tokenizer	11. SequenceInputStream	12. java.io
13. Throwable	14. readLine()	15. RandomAccessFile	16. throw
17. unchecked	18. throws	19. finally	20. stream
21. byte	22. Reader	23. FileWriter	24. primitive
25. nested	26. file	27. delete()	28. streams

Q.III State True or False:

1. Can we perform file handling in Java by Java I/O API?
2. Any statement that can throw an Error must be enclosed in a try block.
3. The catch(X x) can catch subclasses of X where X is a subclass of Exception.
4. The createNewFile() method returns true when it successfully creates a new file.
5. readChar( ) not reads a unicode character.
6. Unchecked exceptions are checked at compile-time rather they are checked at runtime.
7. Exception Handling is a mechanism to handle runtime errors.
8. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
9. A finally block is executed before the catch block but after the try block.
10. A finally block is executed whether an exception is thrown or not.
11. A finally block is executed, only after the catch block is executed.
12. The two basic streams used are the input and output streams.
13. Filters are used to read data from one stream and write it to another stream.
14. Java application uses an output stream to read data from a source, it may be a file, an array, peripheral device or socket.
15. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
16. An exception (or exceptional event) is a problem that arises during the execution of a program.
17. A checked exception is an exception that is checked (notified) by the compiler at compilation-time, these are also called as compile time exceptions.

18. All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class.
19. The Exception class has two main subclasses nsmely, IOException class and RuntimeException Class.
20. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally
21. A finally block appears at the end of the catch blocks.
22. The FileWriter class of the java.io package can be used to read data (in characters) from files.
23. FileOutputStream is used to create a file and write data into it.
24. To create a file in Java, we can use the createNewFile() method.
25. We use the FileWriter class together with its write() method to write some text to the file.

Answers

1. (T)	2. (F)	3. (T)	4. (T)	5. (F)	6. (F)	7. (T)	8. (T)	9. (F)	10. (T)
11. (F)	12. (T)	13. (T)	14. (F)	15. (T)	16. (T)	17. (T)	18. (T)	19. (T)	20. (T)
21. (T)	22. (F)	23. (T)	24. (T)	25. (T)					

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. Define exception.
2. What is error?
3. List types of exceptions.
4. Which statement is used for handling exceptions.
5. Define stream.
6. What is file?
7. What is the purpose of FileReader and FileWriter?
8. Name any two checked exceptions.
9. What is a user-defined exception?
10. Define nested try statement with it syntax.
11. Give the purpose of throw and throws.
12. Define exception handling.
13. What is the use of Reader and Writer?
14. What is the purpose of BufferedReader and BufferedWriter?

(B) Long Answer Questions:

1. What is an exception? In which package exceptions are defined?
2. Can user program throw own exceptions? How? Explain with example.
3. Explain try-catch-finally exception handling mechanism in detail.
4. What is the purpose of multiple catch blocks in the program?
5. With the help of example explain how nested try blocks are handled in Java.
6. Explain following terms:
   (i) throws
   (ii) try
   (iii) catch
   (iv) finally.
7. What is a stream? List types of streams.
8. Which are the predefined streams? Explain diagrammatically.
9. Define file? How to handle files in Java? Explain in detail.
10. Explain various file operation with example.
11. Describe the term file I/O basic.
12. How to read/write characters in a file? Explain with example.
13. How to read/write bytes in a file?
14. Write a note on:
    (i) BufferedInputStream and BufferedOutputStream
    (ii) DataInputStream and DataOutputStream
    (iii) BufferedReader and BufferedWriter
    (iv) FileReader and FileWriter

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. What is the basic difference between 'throw' and 'throws' keywords? **[1 M]**

Ans. Refer to Page 4.15.

2. Write a Java program to count number of whitespace character in character stream. **[5 M]**

Ans. Refer to Section 4.5.2.

3. Write a Java program to copy a file to another file. Use proper error handler to handle runtime exception. File names are accepted as argument from command prompt. **[5 M]**

Ans. Refer to Program 4.17.

4. Explain different types of streams. **[3 M]**

Ans. Refer to Section 4.5.2.

October 2016

1. What is the use of finally keyword? **[1 M]**

Ans. Refer to Section 4.2.4 Point (3).

2. Write a Java program to get file name from command prompt. Check whether a file by given name exists. If file is a regular file, then display various details about that file. But if it is a directory, then display number of files in that directory. **[5 M]**

Ans. Refer to Section 4.5.

3. List and describe any five methods of Reader class along with their syntax. **[5 M]**

Ans. Refer to Section 4.7.1.

April 2017

1. Write a Java program to accept a file name from command prompt. If the file exists, then display number of words and lines in that file using FileReader class. **[5 M]**

Ans. Refer to Section 4.7.3.

2. List and describe any five methods of InputStream class along with their syntax. **[5 M]**

Ans. Refer to Section 4.6.1.

October 2017

1. Give difference between throw and throws. **[2 M]**

Ans. Refer to Page 4.15.

2. Which stream is used to read primitive data from file? **[1 M]**

Ans. Refer to Section 4.5.2.

3. Write a java program to delete all the files from the current directory having extension as .doc also display the count number of .doc files deleted separately. **[5 M]**

Ans. Refer to Page 4.27.

4. Explain any three methods of the Input Stream class. **[3 M]**

Ans. Refer to Section 1.6.1.

5. Write a java program which stores the username and password in two variables. If username and password are not same, then raise "Invalid password" with appropriate message. **[5 M]**

Ans. Refer to Section 4.2.4.

April 2018

1. Write a Java program which displays the contents of file in reverse order. **[5 M]**

Ans. Refer to Section 4.5.

2. What are user-defined exceptions ? Illustrate them with an example. **[5 M]**

Ans. Refer to Section 4.4.

October 2018

1. What is the use of finally block? **[1 M]**

Ans. Refer to Section 4.2.4 Point (3).

2. Which method is used to set the current position of the file pointer within the file. **[1 M]**

Ans. Refer to Section 4.5.

3. What is stream ? Explain the types of streams supported by java. **[5 M]**

Ans. Refer to Section 4.5.2.

4. Write a java program to accept the details of product as productcode, productname and weight. If weight > 100 then throw an exception as InvalidProduct Exception and give the proper message. Otherwise display the product details. Define required exception class. **[5 M]**

Ans. Refer to Section 4.2.4.

5. What is checked and unchecked exceptions? **[2 M]**

Ans. Refer to Section 4.2.3.

6. Write a java program to display the contents of the file in reverse order. **[4 M]**

Ans. Refer to Section 4.5.

April 2019

1. Define unchecked exception. **[1 M]**

Ans. Refer to Section 4.2.3 Point (2).

2. List any two methods of File class. **[1 M]**

Ans. Refer to Section 4.5.

3. Write a Java program to appends the content of one file to another. **[5 M]**

Ans. Refer to Section 4.5.

4. What are user defined exceptions? Illustrate them with an example. **[5 M]**

Ans. Refer to Section 4.4.

■■■

CHAPTER 5

User Interface with AWT and Swing

Objectives...

- To undertand Concepts in AWT
- To study Swing Concepts
- To study Layout Manager in AWT
- To learn Dialogs
- To study EventHandling in Java

5.0 INTRODUCTION

- User Interface abbreviated as UI. It is the junction/interface between a user and a computer program.
- Every user interface considers the following three main aspects:
  1. **UI elements:** These are the core visual elements the user eventually sees and interacts with.
  2. **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).
  3. **Behavior:** These are events which occur when the user interacts with UI elements.
- The human computer interface is what allows the user to communicate with the computer and is often called as User Interface (UI).
- There exist several types of user interfaces. We here give you just two examples:
  1. **Command-Line Interface (CLI):** In CLI user provides the input by typing a command string with the computer keyboard and the system provides output by printing text on the computer monitor.
  2. **Graphical User Interface (GUI):** The use of pictures rather than just words to represent the input and output of a program, for example, Linuxjunkies. Input is accepted via devices such as keyboard and mouse.
- User generally wants the user friendly program. The program which is easily understood and its functionality is easy to handle. Java provides such controls which provides this easiness.
- To develop GUI Java provided AWT and Swings. To handle these components we need event handling. Event Handling is also needed for applet programming.

- The Abstract Window Toolkit (AWT) is Java's original platform-independent windowing, graphics, and user-interface widget toolkit.
- The AWT is now part of the Java Foundation Classes (JFC) the standard API for providing a GUI for a Java program.
- Swing is a GUI widget toolkit for Java language, swing is a part of JFC; an API for providing a GUI for Java programas.
- There are two sets of java API for graphics programming i.e., AWT and Swing.

5.1 OVERVIEW OF AWT AND SWING

- The Abstract Window Toolkit (AWT) is Java's original platform-dependent windowing, graphics, and user-interface widget toolkit, preceding Swing.
- The AWT is part of the Java Foundation Classes (JFC) the standard API for providing a graphical user interface (GUI) for a Java program.
- in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications. It is a part of the JFC (Java Foundation Classes).

5.1.1 What is AWT? [April 16]

- The Java programming language class library provides a user interface toolkit called the Abstract Window Toolkit, or the AWT. Java AWT is an API to develop GUI or window-based application in Java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components uses the resources of system.
- The AWT is now part of the Java Foundation Classes (JFC) - the standard API for providing a Graphical User Interface (GUI) for a Java program.
- The main purpose for using the AWT is, using for all the components displaying on the screen.
- AWT defines all the windows according to a class hierarchy those are useful at a specific level or we can say arranged according to their functionality.

AWT Classes:

- The AWT classes are contained in the java.awt package. It is one of Java's largest packages.
- Fortunately, because it is logically organized in a top-down, hierarchical fashion, it is easier to understand and use than we might at first believe.
- The hierarchy of Java AWT classes are given below:

Class	Description
AWTEvent	Encapsulates AWT events.
AWTEventMulticaster	Dispatches events to multiple listeners.
BorderLayout	The border layout manager. Border layouts use five components i.e. North, South, East, West and Centre.
Button	Creates a push button control.

contd. …

Canvas	A blank, semantics-free window.
CardLayout	The card layout manager. Card layouts emulate indexcards. Only the one on top is showing.
Checkbox	Creates a check box control.
CheckboxGroup	Creates a group of check box controls.
CheckboxMenuItem	Creates an on/off menu item.
Choice	Creates a pop-up list.
Color	Manages colors in a portable, platform-independent fashion.
Component	An abstract super-class for various AWT components.
Container	A subclass of Component that can hold other components.
Cursor	Encapsulates a bitmapped cursor.
Dialog	Creates a dialog window.
Dimension	Specifies the dimensions of an object. The width is stored in width, and the height is stored in height.
Event	Encapsulates events.
EventQueue	Queues events.
FileDialog	Creates a window from which a file can be selected.
FlowLayout	The flow layout manager. Flow layout positions components left to right, top to bottom.
Font	Encapsulates a type font.
FontMetrics	Encapsulates various information related to a font.
Frame	Creates a standard window that has a title bar, resize corners, and a menu bar.
TextArea	Creates a multiline edit control.
TextComponent	A superclass for TextArea and TextField.
TextField	Creates a single-line edit control.
Toolkit	Abstract class implemented by the AWT.
Window	Creates a window with no frame, no menu bar.

5.1.2 What is Swing? [April 16]

- Swing is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) - an API for providing a Graphical User Interface (GUI) for Java programs.
- Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT).
- Swing provides a native, pluggable look and feel that emulates of several platforms, that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT.

- In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.
- Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent. The term "lightweight" is used to describe such an element.

5.1.3 Difference between AWT and Swing [April 16]

- There are many differences between AWT and Swing that are given below:

Sr. No.	AWT	Swing
1.	AWT components are platform dependent.	Java swing components are platform-independent.
2.	AWT components are heavy weight.	Swing components are light weight.
3.	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4.	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5.	AWT components require java.awt package.	Swing components require javax.swing package.
6.	Advanced features is not available in AWT.	Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT.
7.	Different looked feel feature is not supported in AWT.	We can have different look and feel in Swing.
8.	Using AWT, we have to implement a lot of things ourself.	Swing has them built in.

5.2 MVC ARCHITECTURE AND SWING

- The design of swing component classes is based on Model-View-Controller (MVC) architecture.
- MVC architectural pattern is used in software engineering to allow for the separation of three common features of GUI applications:
  1. The data access (typically via a database),
  2. The business logic (how the data will be used), and
  3. User interaction (how the data and actions will be visually presented).
- MVC is a method of building reusable components that logically separate the structure, presentation and behaviour of a component into separate pieces.

- The fundamental idea behind MVC is the separation of data model from its presentation. MVC has mainly three parts:
  1. The model, which stores the data of component.
  2. The view, which creates visual representation of the component using the data in the model.
  3. The controller, which deals with the user interaction and modifies the model or the view in response to user action.

Fig. 5.1: MVC Architecture

- For example, a data in a spread-sheet can be represented using different types of charts like bar chart, pie chart, etc. Here data in a spreadsheet is the model and various chart types are views.
- A single model can have many views. The change to the model causes the changes in a view.
- The view and controller can be represented by a single composite object called UI-delegate.
- UI-delegate is responsible for the view and controller roles. It takes care of drawing the component and representing to user events.
- Fig. 5.2 shows Separate Model Architecture in which the different UI-delegate can provide a component with a new look and feel.
- The way in which swing objects communicate, by passing events from sources to listeners is a part of MVC concept of separation.

Fig. 5.2: Separable Model Architecture

- Event listeners are observers (controllers) and event sources are observerables (models). When an observable changes or performs a function, it notifies all of its observers of the activity.
- The java.util.observer interface and java.util.observable class are utilities which provides fundamental design pattern under the concept of separable model architecture.

5.3 LAYOUTS AND LAYOUT MANAGERS [April 16, 18, Oct. 16, 18]

- Layout means the arrangement of components within the container. In other way we can say that placing the components at a particular position within the container.
- The task of layouting the controls are done automatically by the Layout Manager. Layout manager is an object which determines the way that components are arranged in a frame window.
- All of the components that we have shown so far have been positioned by the default layout manager.
- A layout manager automatically arranges our controls within a window by using some type of algorithm.
- A layout manager is an instance of any class that implements the LayoutManager interface.
- The layout manager is set by the setLayout() method. If no call to setLayout() is made, then the default layout manager is used.
- Whenever, a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.
- The setLayout() method has the following general **form/syntax**:

  ```
  void setLayout(LayoutManager layoutObj)
  ```

  Here, layoutObj is a reference to the desired layout manager. If we wish to disable the layout manager and position components manually, pass null for layoutObj.
- If we do this, we will need to determine the shape and position of each component manually, using the setBounds() method defined by Component. Normally, we will want to use a layout manager.
- Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time we add a component to a container.
- Whenever, the container needs to be resized, the layout manager is consulted via its minimum LayoutSize() and preferred LayoutSize() methods.
- Each component that is being managed by a layout manager contains the getPreferredSize() and getMinimumSize() methods. These return the preferred and minimum size required to display each component.
- The layout manager will honor these requests if at all possible, while maintaining the integrity of the layout policy. We may override these methods for controls that we subclass.
- The layout manager automatically positions all the components within the container. Default values are provided otherwise.
- Java has several predefined layout manager classes. Layout manager classes are founded in the java.awt package same as the classes used for displaying graphical components.
- Several layout managers of which are described next sections. We can use the layout manager that best fits our application.

- The java.awt package provides five layout manager classes as given below:
  1. **FlowLayout:** Arranges components in variable-length rows.
  2. **BorderLayout:** Arranges components along the sides of the container and in the middle.
  3. **CardLayout:** Arrange components in "cards" Only one card is visible at a time.
  4. **GridBagLayout:** Aligns components horizontally and vertically; components can be of different sizes.
  5. **GridLayout:** Arranges components in fixed-length rows and columns.

5.3.1 FlowLayout Manager

- The FlowLayout is one the most popular and simplest layout manager. FlowLayout manager places components in a container from left to right in the order in which they were added to the container.
- When one row is filled, layout advances to the next row. It is analogous to lines of text in a paragraph.
- Components managed by a FlowLayout are always set to their preferred size (both width and height) regardless of the size of the parent container.
- Whenever the container is resized then the components in the container are also adjusted from the top-left corner. The FlowLayout is the default layout manager for a Panel, Applet and JPanel.
- **Following are constructors for FlowLayout:**
  1. `FlowLayout()`: This constructor centers all components and leaves five pixel spaces between each component.
  2. `FlowLayout(int align)`: This constructor allows to specify how each line of components is aligned:
     i.e., `Flowlayout.LEFT`
     `FlowLayout.CENTER`
     `FlowLayout.RIGHT`
  3. `FlowLayout(int align, int hspace, int vspace)`: This constructor allow to specify the alignment as well as the horizontal and vertical space between components.

Program 5.1: Program for FlowLayout manager.

```
import java.awt.*;
class FlowLayoutDemo extends Frame
{
   FlowLayoutDemo()
   {
      Button[] button =new Button[10];
      setLayout(new FlowLayout());
      for(int i=0;i<button.length;i++)
      {
         button[i]=new Button("Button "+(i+1));
         add(button[i]);
      }
   }
}
```

```
class FlowLayoutJavaDemo
{
    public static void main(String args[])
    {
        FlowLayoutDemo frame = new FlowLayoutDemo();
        frame.setTitle("FlowLayout in Java Example");
        frame.setSize(400,150);
        frame.setVisible(true);
    }
}
```

Output:

5.3.2 BorderLayout Manager

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as north (upper region), south (lower region), east (left region), and west (right region). The middle area is called the center (central region).
- Five zones, one for each component, as can be seen in Fig. 5.3.

(a)

(b)

Fig. 5.3: Border Layout Manager

- The constructors defined by BorderLayout as given below:
  1. `BorderLayout()`: This constructor creates a default border layout.
  2. `BorderLayout(int hspace, int vspace)`: This constructor allows us to specify the horizontal and vertical space left between components in horz and vert, respectively.
- To add components in a frame using BorderLayout manager, following form of add method is used:

```
void add (Component_obj, Object region)
```

where region specifies any of the following five constants:

1. BorderLayout.CENTER: Places component at center.
2. BorderLayout.EAST: Places component at right border.
3. BorderLayout.WEST: Places component at left border.
4. BorderLayout.SOUTH: Places component at lower border.
5. BorderLayout.NORTH: Places component at upper border.

Program 5.2: Program to demonstrate border layout.

```
import java.awt.*;
class myFrame extends Frame
{
   TextField name, pass;
   Button E,W,S,N,C;
   BorderLayout B1;
   myFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      B1=new BorderLayout();
      setLayout(B1);
      E=new Button(“EAST”);
      W=new Button(“WEST”);
      S=new Button(“SOUTH”);
      N=new Button(“NORTH”);
      C=new Button(“CENTER”);
      add(E,BorderLayout.EAST);
      add(W,BorderLayout.WEST);
      dd(S,BorderLayout.SOUTH);
      add(N,BorderLayout.NORTH);
      add(C,BorderLayout.CENTER);
   }
   public static void main(String[] args)
   {
      myFrame f= new myFrame("Demonstrating Frame");
   }
}
```

Output:

5.3.3 GridLayout Manager

- GridLayout lay out components in a two-dimensional grid. When we instantiate a GridLayout, we define the number of rows and columns.
- The Grid layout manager places items in rows (left to right) and columns (top to bottom). The number of rows and columns is defined when the Grid layout manager is created.
- The GridLayout layout manager divides the container into a rectangular grid so that component can be placed in rows and column. The intersection of each row and column is known as a cell.
- The components are laid out in cells and each cell has the same size, components are added to a GridLayout starting at the top left cell of the grid and continuing to the right until the row is full. The process continues left to right on the next row of the grid and so on.

Fig. 5.4: Grid Layout Manager

- The constructors supported by GridLayout are shown below:
  1. `GridLayout()`: Creates a single-column grid layout.
  2. `GridLayout(int numRows, int numColumns)`: Creates a grid layout with the specified number of rows and columns.
  3. `GridLayout(int numRows, int numColumns, int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in horz and vert, respectively. Either numRows or numColumnscan be zero. Specifying numRows as zero allows for unlimited-length columns. Specifying numColumns as zero allows for unlimited-length rows.
- Here, is a sample program that creates a 4 × 4 grid and fills it in with 15 buttons, each labeled with its index:

Program 5.3: Program to demonstrate GridLayout.

```
import java.awt.*;
import javax.swing.*;
public class ExGridLayout
{
   JFrame jframe;
   ExGridLayout()
   {
      jframe=new JFrame();
      JButton jb1=new JButton("ONE");
      JButton jb2=new JButton("TWO");
```

```
        JButton jb3=new JButton("THREE");
        JButton jb4=new JButton("FOUR");
        JButton jb5=new JButton("FIVE");
        JButton jb6=new JButton("SIX");
        JButton jb7=new JButton("SEVEN");
        JButton jb8=new JButton("EIGHT");
        JButton jb9=new JButton("NINE");
        jframe.add(jb1);
        jframe.add(jb2);
        jframe.add(jb3);
        jframe.add(jb4);
        jframe.add(jb5);
        jframe.add(jb6);
        jframe.add(jb7);
        jframe.add(jb8);
        jframe.add(jb9);
        jframe.setLayout(new GridLayout(3,3));
                                //setting grid layout of 3 rows and 3 columns
        jframe.setSize(300,300);
        jframe.setVisible(true);
    }
    public static void main(String[] args)
    {
        new ExGridLayout();
    }
}
```

Output:

5.3.4 GridBagLayout Manager

- The java.awt.GridBagLayout layout manager is the most powerful and flexible of all the predefined layout managers but more complicated to use.
- Unlike GridLayout where the component are arranged in a rectangular grid and each component in the container is forced to be the same size, in GridBagLayout, components are also arranged in rectangular grid but can have different sizes and can occupy multiple rows or columns, (See Fig. 5.5).

- A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.
- GridBagLayout places components in rectangles (cells) in a grid and then uses the components preferred sizes to determine how big the cells should be.
- The class GridBagLayout arranges components in a horizontal and vertical manner.

Fig. 5.5: GridBagLayout

Constructor:

1. `GridBagLayout()`: Creates a grid bag layout manager.

- Instance variables to manipulate the GridBagLayout object Constraints are:

Variable	Role
`gridx and gridy`	Shows co-ordiantes of the origin of the grid and allows positioning.
`gridwidth, gridheight`	Define how many cells will occupy component (height and width), by the default is 1. The indication is relative to the other components of the line or the column. The GridBagConstraints. REMAINDER value specifies that the next component inserted will be the last of the line or the current column. The value GridBagConstraints. RELATIVE up the component after the last component of a row or column.
`fill`	Defines the fate of a component smaller than the grid cell. GridBagConstraints. NONE retains the original size: Default GridBagConstraints. HORIZONTAL expanded horizontally GridBagConstraints. VERTICAL GridBagConstraints: BOTH expanded vertically expanded to the dimensions of the cell.
`ipadx, ipady`	Used to define the horizontal and vertical expansion of components, not works if expansion is required by fill. The default value is (0,0).
`anchor`	When a component is smaller than the cell in which it is inserted, it can be positioned using this variable to define the side from which the control should be aligned within the cell. Possible variables NORTH, NORTHWEST, NORTHEAST, SOUTH, SOUTHWEST, SOUTHEAST, WEST and EAST.
`weightx, weighty`	Used to define the distribution of space in case of change of dimension.

Program 5.4: Program to demonstrate Gridbaglayout.

```
import java.awt.*;
import javax.swing.*;
public class Main {
    public static void main(String[] args) {
        JFrame frame = new JFrame("GridBagLayout Demo");
        JButton btn1 = new JButton("Button 1");
        JButton btn2 = new JButton("Button 2");
        JButton btn3 = new JButton("Button 3");
        JButton btn4 = new JButton("Button 4");
        JButton btn5 = new JButton("Button 5");
        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints cst = new GridBagConstraints();
         // add button 1 to the panel
        cst.fill = GridBagConstraints.HORIZONTAL;
 cst.gridx = 0;
 cst.gridy = 0;
        panel.add(btn1,cst);
         // add button 2 to the panel
        cst.fill = GridBagConstraints.HORIZONTAL;
 cst.gridx = 1;
 cst.gridy = 0;
        panel.add(btn2);
         // add button 3 to the panel
 cst.gridx = 2;
 cst.gridy = 0;
 panel.add(btn3, cst);
          // add button 4 to the panel
         cst.fill = GridBagConstraints.HORIZONTAL;
 cst.gridwidth = 3;
 cst.gridx = 0;
 cst.gridy = 1;
        panel.add(btn4,cst);
        // add button 5 to the panel
        cst.fill = GridBagConstraints.HORIZONTAL;
 cst.gridx = 2;
 cst.gridwidth = 1;
 cst.gridy = 2;       //third row
        panel.add(btn5,cst);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300,200);
        frame.getContentPane().add(panel);
        frame.setVisible(true);
    }
}
```

Output:

5.3.5 CardLayout Manager

- The CardLayout manager provides the means to manage multiple components, displaying one at a time.
- Components in CardLayout are displayed in the order in which they are added to the layout or in an arbitrary order by using an assignable name.
- The CardLayout class is unique among the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.
- CardLayout can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- The class CardLayout arranges each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.
- We can prepare the other layouts and have them hidden, ready to be activated when needed.
- CardLayout provides following two constructors:
  1. `CardLayout()`: Creates a default card layout.
  2. `CardLayout(int horz, int vert)`: Allows us to specify the horizontal and vertical space left between components in horz and vert, respectively.
- Uses of CardLayout require a bit more work than the other layouts. The cards are typically held in an object of type Panel.
- This panel must have CardLayout selected as its layout manager. The cards that form the declare also typically objects the type panel.
- We can use add() method for adding cards. It takes following general **form/syntax**:

  ```
  void add(Component panelObj, Object name);
  ```

  Here, name is a string that specifies the name of the card whose panel is specified by panelObj.
- After we have created a deck, our program activates a card by calling one of the following methods defined by CardLayout:

  ```
  void first(Container deck)
  void last(Container deck)
  void next(Container deck)
  void previous(Container deck)
  void show(Container deck, String cardName)
  ```

Here, deck is a reference to the container (usually a panel) that holds the cards, and cardName is the name of a card. Calling first() causes the first card in the deck to be shown. For showing the last card, call last() and for the next card, call next().

- To show the revious card, call previous(). Both next() and previous() automatically cycle back to the top or bottom of the deck, respectively. The show() method displays the card whose name is passed in cardName.

Program 5.5: Program to demonstrate CardLayout.

```
import java.awt.*;
import java.awt.event.*;
//Class definition
class Panel1 extends Panel
{
   //Variable declaration
   Label la1,la2,la3;
   TextField te1,te2,te3;
   Button bu1, bu2, bu3, bu4, bu5;
   public Panel1()
   {
      la1=new Label("Number 1:");
      la2=new Label("Number 2:");
      la3=new Label("Result:");
      te1=new TextField(10);
      te2=new TextField(10);
      te3=new TextField(10);
      te3.setEditable(false);
      bu1=new Button("Add");
      bu2=new Button("Sub");
      bu3=new Button("Mul");
      bu4=new Button("Div");
      bu5=new Button("Exit");
      add(la1);
      add(te1);
      add(la2);
      add(te2);
      add(la3);
      add(te3);
      add(bu1);
      add(bu2);
      add(bu3);
      add(bu4);
      add(bu5);
   }
}
```

```
class Panel2 extends Panel
{
   Checkbox ch1, ch2, ch3, ch4;
   public Panel2()
   {
      ch1=new Checkbox("Windows 8");
      ch2=new Checkbox("Linux Ubuntu");
      ch3=new Checkbox("Android");
      ch4=new Checkbox("macOS");
      setLayout(new GridLayout(2,2));
      add(ch1);
      add(ch2);
      add(ch3);
      add(ch4);
   }
}
class Panel3 extends Panel
{
   TextArea tax;
   TextField te1, te2;
   Label la1, la2;
   Button bu1, bu2, bu3, bu4;
   public Panel3()
   {
      tax=new TextArea(10,50);
      te1=new TextField(10);
      te2=new TextField(10);
      la1=new Label("Find What:");
      la2=new Label("Replace With:");
      bu1=new Button("Find");
      bu2=new Button("Replace");
      bu3=new Button("Replace All");
      bu4=new Button("Cancel");
      add(tax);
      add(la1);
      add(te1);
      add(la2);
      add(te2);
      add(bu1);
      add(bu2);
      add(bu3);
      add(bu4);
   }
}
```

```
class CardLayoutDemo extends Frame implements ActionListener
{
   Button bu1, bu2;
   Panel cards;
   CardLayout c;
   Panel1 pa1;
   Panel2 pa2;
   Panel3 pa3;
   public CardLayoutDemo(String title)
   {
      super(title);
      bu1=new Button("Next");//Object creation
      bu2=new Button("Previous");
      cards=new Panel();
      pa1=new Panel1();
      pa2=new Panel2();
      pa3=new Panel3();
      c=new CardLayout();
      cards.setLayout(c);
      cards.add(pa1,"Panel1");
      cards.add(pa2,"Panel2");
      cards.add(pa3,"Panel3");
      bu1.addActionListener(this);
      bu2.addActionListener(this);
      Panel p=new Panel();
      p.add(bu1);
      p.add(bu2);
      add("North",p);
      add("Center",cards);
      setSize(300,300);
      setVisible(true);
   }
   public void actionPerformed(ActionEvent e)
   {
      if(e.getSource()==bu1)//Decision making statement
      c.next(cards);
      else
      c.previous(cards);
   }
   public static void main(String args[])
   {
      new CardLayoutDemo("Card Layout Demo");
   }
}
```

Output:

5.4 AWT HIERARCHY / AWT CONTROLS

- Window is a rectangular area which is displayed on the screen. In different window we can execute different program and display different data.
- Window provides us with multitasking environment. A window must have a frame, dialog or another window defined as its owner when it's constructed.
- Different windows classes are defined by AWT which adds functionality with each level as shown in Fig. 5.6.

Fig. 5.6: Windows classes of AWT

- Fig. 5.6 shows following Windows classes of AWT:
  1. **Component:** It is an abstract class and all GUI components that interact with the user are subclasses of component class. It defines various methods for managing events, such as keyboard and mouse input. It also defines attributes to store current foreground and background colours and font.
  2. **Container:** It inherits various methods and attributes of component class and having additional methods of nesting other component objects. A container positions various GUI components within it, with the help of different layout managers.
  3. **Panel:** Panel is a window that does not contain a title bar, menu bar or border. It is a superclass of Applet class. Any screen output written an applet is drawn on the surface of a Panel object. Appletviewer provides title bar, and border to an applet. We can add GUI components such as buttons, list box etc. to a panel object using add() method.

4. **Window:** This class is used to create a top level window. Generally, windows objects are not created directly, subclass of window is used to place on the desktop.
5. **Frame:** Frame class is a subclass of window and it has title bar, menu bar, borders etc. To create a normal window within an Java application, Frame class is used. In applets, we need not use frame class to create window because appletviewer provides title bar, menu bar and borders to an applet.
6. **Canvas:** Canvas is not subclass of either window or panel. It is one another type of window which displays a blank window upon which you can draw.

Working with Frame Window:

- A Frame is a top-level window with a title and a border.
- The type of window we will be creating for stand alone application is subclass of Frame.
- Two of Frame's constructors:
  1. `Frame()`: The first form creates a standard window that does not contain a title.
  2. `Frame(String title)`: The second form creates a window with the title specified by title.

Setting the Windows Dimensions:

- The setSize() method is used to set the size of the window.
- It's general structure/syntax is shown here:

```
void setSize(int newWidth, int newHeight)
void setSize(Dimension newSize)
```

- The new size of the Frame is specified by newWidth and newHeight or by the width and height fields of the Dimension object passed in newSize. The dimensions are specified in terms of pixels.
- The getSize() method is used to obtain the current size of a window. Its general **syntax** is, `Dimension getSize()`.
- The getSize() method returns the current size of the Frame as the width and height fields of a Dimension object.

Showing and Hiding Frame:

- After a frame has been created, it will not be visible until we call setVisible() method. Its general **syntax** is given below:

```
void setVisible(boolean visibleFlag)
```

- The Frame is visible if the argument to this method is true. Otherwise, it is hidden.

Setting a Frame's Title:

- We can set the title of a frame window using setTitle(), which has following general **form/syntax**:

```
void setTitle(String newTitle)
```

Program 5.6: Program for frame.

```
import java.awt.*;
class MyFrame extends Frame
{
    MyFrame(String title)
    {
```

```
        setVisible(true);
        setSize(100,100);
        setTitle(title);
    }
    public static void main(String args[])
    {
        MyFrame f= new MyFrame("My Frame");
    }
}
```

Output:

Closing a Frame Window:

- When using a frame window, our program must remove that window from the screen when it is closed, by calling setVisible(false).
- To intercept a window close event, we must implement the windowClosing( ) method of the WindowListener interface.
- Inside windowClosing(), we must remove the window from the screen.

AWT Controls:

- A GUI is built of graphical elements called components. Controls are components that allow a user to interact with your application in various ways.
- Every AWT controls inherit properties from Component class. Typical components include such items as buttons, scrollbars, text fields, and so on.
- Components allow the user to interact with the program and provide the user with visual feedback about the state of the program.
- To develop GUI based applications, the fundamental elements that are needed and are available in two packages java.awt and javax.swing.

How to Add or Remove AWT Controls?

- Create an instance of desired control and then add it to the window by calling add() method.

  Syntax: `Component add(Component obj)`
- We can also remove our control from window by calling remove() method.

  Syntax: `void remove(Component obj)`
- We can also used for removeall() method for removing all controls:

1. **Labels:**

- The easiest control to use in Java is a label. A label displays a single line of read-only text.

- Label class is having following constructors:
  (i) `Label()`: The first constructor creates a blank label.
  (ii) `Label(String)`: Creates a label containing a specified string in parameter.
  (iii) `Label(String, int)`: Creates a label containing a specified string with alignment which are specified in second parameter. The alignment must be one of the three constant values: Label.LEFT, Label.RIGHT, Label.CENTER.
- For label following methods can be used:
  (i) `void SetText(String)`: Set or change the text in a label.
  (ii) `String getText()`: Obtain the text in a label.
  (iii) `void setAlignment(int)`: Set or change the alignment of a text in a label.
  (iv) `int getAlignment()`: Obtain the current alignment.

Program 5.7: Program to demonstrate label.

```
import java.awt.*;
class myFrame extends Frame
{
   myFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      Label one = new Label("Label One");
      Label two = new Label("Label Two");
      Label three = new Label("Label Three");
                                          // add labels to applet window
      setLayout(new FlowLayout());
      add(one);
      add( two);
      add( three);
   }
   public static void main(String[] args)
   {
      myFrame f= new myFrame("Demonstrating Frame");
   }
}
```

Output:

2. Push Button:

- A push button is a component that contains a label and that generates an event when it is pressed.

- Push buttons has two constructors.
  (i) `Button()`: Creates a button with no label.
  (ii) `Button(String)`: Creates a button that contains the specified string.
- For button following methods can be used:
  (i) `void setLabel(String)`: Sets a label specified in parameter.
  (ii) `string getLabel()`: Retrieves the label of a button.

Program 5.8: Program to demonstrate buttons.

```
import java.awt.*;
class MyFrame extends Frame
{
   Button ok, cancel;
   MyFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      ok = new Button("OK");
      cancel = new Button("Cancel");
      setLayout(new FlowLayout());
      add(ok);
      add(cancel);
   }
   public static void main(String[] args)
   {
      MyFrame f= new MyFrame("Demonstrating Frame");
   }
}
```

Output:

3. Check Box: [April 18]

- A check box is a control that is used to turn an option on (true) or off (false). It consists of a small box that can either contain a check mark or not.
- Check boxes can be used individually or as part of a group. Check boxes are objects of the Checkbox class.
- Checkbox class supports following constructors:
  (i) `Checkbox()`: Creates checkbox without label. Initial state is unchecked.
  (ii) `Checkbox(String)`: Creates a checkbox with label specified Initial state is unchecked.
  (iii) `Checkbox (String, boolean)`: A checkbox with label and Initial state is specified either true or false in second parameter.

- A checkbox also supports following methods:
  (i) `Boolean getState()`: Retrieves the current state.
  (ii) `void setState(boolean)`: Sets the state true/false.
  (iii) `String getlabel()`: Retrieves the label associated.
  (iv) `void setLabel(String)`: Sets the label as specified.

Program 5.9: Program to demonstrate checkboxes.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.border.EmptyBorder;
public class DemoCheckbox
{
   Frame frame;
   Checkbox chk1, chk2, chk3, chk4;
   DemoCheckbox()
   {
      frame= new JFrame("Demo of Checkbox");
      chk1 = new Checkbox("Tea",true);
            //Creating a pre-selected Checkbox with a message
      chk2 = new Checkbox("Coffee");
      chk3 = new Checkbox("Cold Coffee");
            //Creating an unselected Checkbox with a message
      chk4 = new Checkbox("Green Tea");
      frame.add(chk1);
      frame.add(chk2);
      frame.add(chk3);
      frame.add(chk4);
      frame.setLayout(new FlowLayout());
      frame.setSize(300,120);
      frame.setVisible(true);
   }
   public static void main(String ar[])
   {
      new DemoCheckbox();
   }
}
```

Output:

4. Checkbox Group:

- This class is used to create group of checkboxes. From the group of check boxes only one check box can be checked at a time.
- These check boxes are also called radio buttons. It supports following constructors:
  (i) `CheckboxGroup CheckboxGroup():` Returns group name.
- Checkboxes of Checkboxgroup can be created using following constructors:
  (i) `Checkbox(String, boolean, CheckboxGroup)`
  (ii) `Checkbox(String, CheckboxGroup, boolean)`
- These creates checkbox with specified label and groupname as parameters. The boolean type parameter specifies the initial state true/false.

Program 5.10: Program to demonstrate Checkbox group.

```
import java.awt.*;
class MyFrame extends Frame
{
   String msg="";
   Checkbox Win98, winNT, solaris, mac;
   CheckboxGroup cbg;
   MyFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      cbg = new CheckboxGroup();
      Win98 = new Checkbox("Windows 98/XP", cbg, true);
      winNT = new Checkbox("Windows NT/2000", cbg, false);
      solaris = new Checkbox("Solaris", cbg, false);
      mac = new Checkbox("MacOS", cbg, false);
      setLayout(new FlowLayout());
      add(Win98);
      add(winNT);
      add(solaris);
      add(mac);
   }
public void paint(Graphics g)
{
   msg = "Current selection: ";
   msg += cbg.getSelectedCheckbox().getLabel();
   g.drawString(msg, 6, 100);
}
   public static void main(String[] args)
   {
   MyFrame f= new MyFrame("Demonstrating Frame");
   }
}
```

Output:

5. Choice List:

- The class choice is used to create a drop-down or pop-up list of strings.
- It occupies less screen space than list box control, since it shows only the currently selected item.
- When user clicks on it, the list of choices drop down. It uses only default constructor i.e., choice().
- **Choice class provides following methods:**

(i) `void add(String)`: To add string in the choice list.

(ii) `String getSelectedItem()`: Determines which item is currently selected.

(iii) `int getSelectedIndex()`: Determines the index of currently selected item.

(iv) `int getItemCount()`: Returns number of items in the list.

(v) `void Select (int)`: Selects the item whose index is specified as parameter.

(vi) `void Select(String)`: Selects the item matches with specified string as parameter.

(vii) `String getItem(int)`: Obtains the name associated with the item at specified index as parameter.

Program 5.11: Program to demonstrate choice controls.

```
import java.awt.*;
import javax.swing.*;
class Exchoice
 {
   static Choice c;   // create choice object
   static JFrame jframe;
   Exchoice()
   {
   }
   public static void main(String args[])
   {
      jframe = new JFrame("My Choice");// create a frame
      JPanel p1 = new JPanel();// create e panel
      c = new Choice();// create a choice
```

```
        // add element to the list
        c.add("Anand");
        c.add("Smile");
        c.add("Happy");
        c.add("Anandi");
        p1.add(c);
        jframe.add(p1);
        jframe.show();
        jframe.setSize(300, 300);
    }
}
```

Output:

6. Lists:

- List class provides a list of items, which can be scrolled. From list of items, single or multiple items can be selected.
- It provides three constructors:
  - (i) `List()` : Creates a list box in which single selection is possible.
  - (ii) `List(int)` : Creates a list-box, specified number of items will always be visible selection. Selection is possible.
  - (iii) `List(int, boolean)` : Creates a list box, first parameter specifies number of items to be visible. Second parameter specifies whether multiple selection of item is allowed. True indicates multiple selection.
- Following methods are supported by list class:
  - (i) `void add(String)`: Adds item to the end of list.
  - (ii) `void add(String, int)`: Adds item at the index specified in second parameter.
  - (iii) `String getSelectedItem()`: Returns a string which is selected.
  - (iv) `int getSelectedIndex()`: Returns an index of the selected item.
  - (v) `String[] getSelectedItems()`: Returns an array contains strings which are selected (Multiselect)
  - (vi) `int[] getSelectedIndexes()`: Returns an array contains indexes of the selected items (Multiselect)
  - (vii) `int getItemCount()`: Returns number of items in the list.
  - (viii) `void select(int)`: Selects the item of a specified index in parameter.
  - (ix) `String getItem(int)`: Obtains the name of item having specified index.

Program 5.12: Program to demonstrate lists.

```
import java.awt.*;
class MyFrame extends Frame
{
   List l;
   MyFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      l= new List(3);
      setLayout(new FlowLayout());
      l.add("India");
      l.add("America");
      l.add("Australia");
      l.add("Pakistan");
      add(l);
   }
   public static void main(String[] args)
   {
      MyFrame f= new MyFrame("Demonstrating Frame");
   }
}
```

Output:

7. ScrollBars:

- Scrollbar control represents a scroll bar component in order to enable user to select from range of values.
- Scroll bars may be oriented horizontally or vertically.
- Following constructors are in scrollbar:
  - (i) `Scrollbar()` : Creates a vertical scroll bar.
  - (ii) `Scrollbar(int style)` : Creates a scrollbar of specified style. It can be scrollbar.vertical or scrollbar.horizontal.
  - (iii) `Scrollbar (int style, int start, int thumb, int min, int max)` : Creates scrollbar with specified style. Initial value is in second parameter. In third parameter thumbsize is specified. The minimum and maximum value is specified in min and max.
- Following methods are used by scrollbar:
  - (i) `int getValue()`: To get current value of scroll bar.
  - (ii) `void SetValue(int)`: To set new value.
  - (iii) `int getMinimum()`: To get min value of scrollbar

(iv) `int getMaximum()`: To get max value of scrollbar.

(v) `void setUnitIncrement(int)`: To change unit increment. Default is 1.

(vi) `void setBlockIncrement(int)`: To change block increment. Default is 10.

Program 5.13: Program to demonstrate scroll bar.

```
import java.awt.*;
import javax.swing.*;
public class ExScroll
{
   Frame frame;
   Panel panel;
   Label label1, label2;
   ExScroll()
   {
      frame = new Frame("Scrollbar");
      panel = new Panel();
      label1 = new Label("Displaying a horizontal and verticial Scrollbar
      ",Label.CENTER);
      Scrollbar scrollB1 = new Scrollbar(Scrollbar.HORIZONTAL, 10, 40, 0,
      100);
      Scrollbar scrollB2 = new Scrollbar(Scrollbar.VERTICAL, 10, 60, 0,
      100);

      frame.add(label1,BorderLayout.NORTH);
      frame.add(scrollB2,BorderLayout.EAST);
      frame.add(scrollB1,BorderLayout.SOUTH);

      frame.setSize(370,270);
      frame.setVisible(true);
   }
   public static void main(String s[])
   {
      new ExScroll();
   }
}
```

Outptut:

8. TextField:

- The TextField class implements a single-line text-entry area, usually called an edit control.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- It defines following constructors:
  (i) `Textfield()`: Creates a default textfield.
  (ii) `Textfield(int)`: Creates a textfield with specified number of characters.
  (iii) `TextField(String)`: Creates a textfield with a specified string initially.
  (iv) `TextField (String, int)`: Creates a textfield with a initial text and set its width.
- This class uses following methods:
  (i) `String getText()`: To obtain the content to textfield.
  (ii) `void setText(String)`: To set the Text.
  (iii) `String getSelectedText()`: To get currently selected text.
  (iv) `void select (int StartIndex, int endIndex)`: To select a portion of a text.
  (v) `boolean is Editable()`: To determine editability.
  (vi) `void setEditable(boolean)`: To set or reset editability of textfield.

Program 5.14: Program to demonstrate textfield.

```
import java.awt.*;
class MyFrame extends Frame
{
   TextField name, pass;
   MyFrame(String s)
   {
      super(s);
      setVisible(true);
      setSize(500,300);
      Label namep = new Label("Name: ", Label.RIGHT);
      Label passp = new Label("Password: ", Label.RIGHT);
      name = new TextField(12);
      pass = new TextField(8);
      setLayout(new FlowLayout());
      pass.setEchoChar('?');
      add(namep);
      add(name);
      add(passp);
      add(pass);
   }
   public static void main(String[] args)
   {
      MyFrame f= new MyFrame("Demonstrating Frame");
   }
}
```

Output:

9. TextArea:

- Sometimes, a single line of text input is not enough for a given task.
- To handle these situations, the AWT includes a simple multi-line editor called TextArea.
- Following constructors are available in TextArea:
  - (i) `TextArea()` : Creates a default text area.
  - (ii) `TextArea(int, int)` : Creates a text area for specified number of lines and number of characters.
  - (iii) `TextArea (String)` : Creates with initial string.
  - (v) `TextArea(String, int, int)`: Creates text area with initial string and for specified number of lines and number of characters.
  - (v) `Area (String, int numliTextnes, int numChar, int sBars)` : Creates a text area with scroll bars. The value of sBars must one of following constants. SCROLLBARS_BOTH, SCROLLBARS_NONE, SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_VERTICAL_ONLY
- TextArea also supports the methods getText(), setText(), getSelectedText(), select(), is Editable and setEditable(). Methods are similiar to TextField.
- TextArea adds the following methods:
  - (i) `void append (String)`: Appends specified string at the end of text.
  - (ii) `void insert(String, int)`: Inserts specified string at the specified index.
  - (iii) `void replaceRange(String, int startIndex, int endIndex)`: It replaces the character from start_Index to endIndex–1 with given string.

Program 5.15: Program to demonstrate textarea.

```
import java.awt.*;
public class Demotextarea
{
   Frame frame;
   TextArea textArea1, textArea2, textArea3, textArea4;
   Demotextarea()
   {
      frame= new Frame("Demo of TextArea");
      textArea1 = new TextArea() //TextArea()
      textArea2 = new TextArea(2,2); //TextArea(int rows, int columns)
      textArea3 = new TextArea("Third textarea", 10,10);
      textArea4 = new TextArea("Fourth textarea", 5,20);
```

```
        textArea1.append("First textarea");
        textArea2.append("Second textarea");
        frame.add(textArea1);
        frame.add(textArea2);
        frame.add(textArea3);
        frame.add(textArea4);
        frame.setLayout(new FlowLayout());
        frame.setSize(500,400);
        frame.setVisible(true);
    }
    public static void main(String []arg)
    {
        new Demotextarea();
    }
}
```

Output:

10. Menus: [Oct. 18]

- A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop down menu.
- Fig. 5.7 shows menu hierarchy.

Fig. 5.7

- Menu controls are listed in following table:

Sr. No.	Control	Description
1.	MenuComponent	It is the top level class for all menu related controls.
2.	MenuBar	The MenuBar object is associated with the top-level window.
3.	MenuItem	The items in the menu must belong to the MenuItem or any of its subclass.
4.	Menu	The Menu object is a pull-down menu component which is displayed from the menu bar.
5.	CheckboxMenuItem	CheckboxMenuItem is subclass of MenuItem.
6.	PopupMenu	PopupMenu can be dynamically popped up at a specified position within a component.

- The Menu class represents pull-down menu component which is deployed from a menu bar.
- Following are the constructors for Menu:
  (i) Menu()
  (ii) Menu(String optionName)
  (iii) Menu(String optionName, boolean removable)

  Here, optionName specifies the name of the menu selection. If removable is true, the pop-up menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar.

  The first form creates an empty menu. Individual menu items are of type MenuItem. It defines these constructors:
  - `MenuItem()`: Constructs a new MenuItem with an empty label and no keyboard shortcut.
  - `MenuItem(String itemName)`: Creates a menuItem where itemName is the name shown in the menu.
  - `MenuItem(String itemName, MenuShortcut keyAccel)`: Create a menu item with an associated keyboard shortcut.

  Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.
- We can disable or enable a menu item by using the setEnabled() method. Its **form/syntax** is shown below:

```
void setEnabled(boolean enabledFlag)
```

Program 5.16: Program to demonstrate menu.

```
import java.awt.*;
import java.awt.event.*;
public class AWTMenuDemo
{
    private Frame mainFrame;
    private Label headerLabel;
    private Label statusLabel;
```

```
      private Panel controlPanel;
      public AWTMenuDemo(){
      prepareGUI();
    }
    public static void main(String[] args)
    {
      AWTMenuDemo awtMenuDemo = new AWTMenuDemo();
      awtMenuDemo.showMenuDemo();
    }
      private void prepareGUI(){
      mainFrame = new Frame("Java AWT Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent windowEvent)
      {
         System.exit(0);
      }
    });
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      statusLabel = new Label();
      statusLabel.setAlignment(Label.CENTER);
      statusLabel.setSize(350,100);
      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
    }
      private void showMenuDemo()
      {
         //create a menu bar
         final MenuBar menuBar = new MenuBar();
         //create menus
         Menu fileMenu = new Menu("File");
         Menu editMenu = new Menu("Edit");
         final Menu aboutMenu = new Menu("About");
         //create menu items
         MenuItem newMenuItem =
                         new MenuItem("New",new MenuShortcut(KeyEvent.VK_N));
         newMenuItem.setActionCommand("New");
```

```
        MenuItem openMenuItem = new MenuItem("Open");
        openMenuItem.setActionCommand("Open");
        MenuItem saveMenuItem = new MenuItem("Save");
        saveMenuItem.setActionCommand("Save");
        MenuItem exitMenuItem = new MenuItem("Exit");
        exitMenuItem.setActionCommand("Exit");
        MenuItem cutMenuItem = new MenuItem("Cut");
        cutMenuItem.setActionCommand("Cut");
        MenuItem copyMenuItem = new MenuItem("Copy");
        copyMenuItem.setActionCommand("Copy");
        MenuItem pasteMenuItem = new MenuItem("Paste");
        pasteMenuItem.setActionCommand("Paste");
        MenuItemListener menuItemListener = new MenuItemListener();
        newMenuItem.addActionListener(menuItemListener);
        openMenuItem.addActionListener(menuItemListener);
        saveMenuItem.addActionListener(menuItemListener);
        exitMenuItem.addActionListener(menuItemListener);
        cutMenuItem.addActionListener(menuItemListener);
        copyMenuItem.addActionListener(menuItemListener);
        pasteMenuItem.addActionListener(menuItemListener);
        final CheckboxMenuItem showWindowMenu =
        new CheckboxMenuItem("Show About", true);
        showWindowMenu.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e)
    {
        if(showWindowMenu.getState()){
        menuBar.add(aboutMenu);
    }else{
        menuBar.remove(aboutMenu);
    }
  }
  });
  //add menu items to menus
        fileMenu.add(newMenuItem);
        fileMenu.add(openMenuItem);
        fileMenu.add(saveMenuItem);
        fileMenu.addSeparator();
        fileMenu.add(showWindowMenu);
        fileMenu.addSeparator();
        fileMenu.add(exitMenuItem);
        editMenu.add(cutMenuItem);
        editMenu.add(copyMenuItem);
        editMenu.add(pasteMenuItem);
        //add menu to menubar
        menuBar.add(fileMenu);
        menuBar.add(editMenu);
        menuBar.add(aboutMenu);
```

```
        //add menubar to the frame
        mainFrame.setMenuBar(menuBar);
        mainFrame.setVisible(true);
    }
    class MenuItemListener implements ActionListener
    {
        public void actionPerformed(ActionEvent e) {
        statusLabel.setText(e.getActionCommand() + " MenuItem clicked.");
        }
    }
}
```

Output:

- If the argument enabledFlag is true, the menu item is enabled. If false, the menu item is disabled.
- We can determine an item's status by calling isEnabled( ). Its form/syntax is shown below:

  `boolean isEnabled()`
- The isEnabled() returns true if the menu item on which it is called is enabled. Otherwise, it returns false.
- We can change the name of a menu item by calling setLabel(). Its form/syntax is shown below:

  `void setLabel(String newName)`

  Here, newName becomes the new name of the invoking menu item.
- We can retrieve the current name by using getLabel(). Its form/syntax is shown below:

  `String getLabel()`

  `getLabel()` returns the current name.
- We can create a checkable menu item by using a subclass of MenuItem called CheckboxMenuItem(). It has following constructors:

  (i) `CheckboxMenuItem()`: Create a check box menu item with an empty label.

  (ii) `CheckboxMenuItem(String itemName)`: Create a check box menu item with the specified item name.

  (iii) `CheckboxMenuItem(String itemName, boolean on)`: Create a check box menu item with the specified label and state. Here, itemName is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes.
- In the first two forms, the checkable entry is unchecked. In the third form, if on is true, the checkable entry is initially checked. Otherwise, it is cleared.

- We can obtain the status of a checkable item by calling getState(). We can set it to a known state by using setState(). Its form/syntax is shown below:
  `void setState(boolean checked)`
- If the item is checked, getState() returns true, otherwise, it returns false. Its form/syntax is shown below:
  `boolean getState()`
- For checking an item, pass true to setState(). To clear an item, pass false. Once we have created a menu item, we must add the item to a Menu object by using add( ), which has the following general **form/syntax**:
  `MenuItem add(MenuItem item)`
  Here, item is the item being added. Items are added to a menu in the order in which the calls to add( ) take place. The item is returned.
  Once we have added all items to a Menu object, we can add that object to the menu bar by using this version of add( ) defined by MenuBar.
  `Menu add(Menu menu)`
- Here, menu is the menu being added. The menu is returned. Menus only generate events when an item of type MenuItem or CheckboxMenuItem is selected.
- They do not generate events when a menu bar is accessed to display a drop-down menu, for example.
- Each time a menu item is selected, an ActionEvent object is generated. Each time a check box menu item is checked or unchecked, an ItemEvent object is generated.
- Thus, we must implement the ActionListener and ItemListener interfaces in order to handle these menu events.
- The getItem() method of ItemEvent returns a reference to the item that generated this event.
- The general **form/syntax** of this method is shown below:
  `Object getItem()`

11. Pop-up Menus:

- Pop-up menu represents a menu which can be dynamically popped up at a specified position within a component.

Constructors:

(i) `PopupMenu()`: Creates a new popup menu with an empty name.
(ii) `PopupMenu(String label)`: Creates a new popup menu with the specified name.

- Following is an Program 5.17 that adds a series of nested menus to a pop-up window. The item selected is displayed in the window. The state of the two check box menu items is also displayed.

Program 5.17: Program to demonstrate pop-up menu.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ExMenu
{
    Frame frame;
    MenuBar menuBar;
    Menu menu1, menu2;
    MenuItem mItem1, mItem2, mItem3, mItem4, mItem5, mItem6, mItem7;
```

```
    ExMenu()
    {
        frame = new Frame("MenuBar, Menu and MenuItems");

        //Creating a menu bar
        menuBar= new MenuBar();

        //Creating first menu
        menu1 = new Menu("File");
        mItem1 = new MenuItem("New");
        mItem2 = new MenuItem("Open");
        mItem3 = new MenuItem("Save");

        //Adding menu items to the  menu
        menu1.add(mItem1);
        menu1.add(mItem2);
        menu1.add(mItem3);

        //Creating a second sub-menu
        menu2 = new Menu("Save As");
        mItem5 = new MenuItem(".ppt");
        mItem6 = new MenuItem(".doc");
        mItem7 = new MenuItem(".pdf");

        //Adding menu items to the sub-menu
        menu2.add(mItem5);
        menu2.add(mItem6);
        menu2.add(mItem7);

        //Adding the sub-menu to the first menu
        menu1.add(menu2);

        menuBar.add(menu1);
        frame.setMenuBar(menuBar);
        frame.setSize(330,250);
        frame.setVisible(true);
    }
    public static void main(String arg[])
    {
        new ExMenu();
    }
}
```

Output:

5.5 SWING (CONTAINERS AND COMPONENTS)

- The swing components are defined in the package javax.swing. This package provides more powerful and flexible components.
- Swing is a Java foundation classes, library and it is an extension to do Abstract Window Toolkit (AWT).
- Swing is an extension to the AWT components which provides feature of pluggable look and feel for the components. It provides classes to design lightweight components.
- A component is an independent visual control and Java Swing Framework contains a large set of these components which provide rich functionalities and allow high level of customization.
- They all are derived from JComponent class. All these components are lightweight components. This class provides some common functionality like pluggable look and feel, support for accessibility, drag and drop, layout, etc.
- A container holds a group of components. It provides a space where a component can be managed and displayed.
- Various features of swing are listed below: **[Oct. 17, April 18]**
  1. **Light Weight**: Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls. **[April 18]**
  2. **Rich Controls**: Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, color picker, table controls and so on.
  3. **Borders:** We can draw borders in many different styles around components using the setborder( ) method.
  4. **Easy mouseless Operation:** It is easy to connect keystrokes to components.
  5. **Tooltips:** We can use the setToolTipText method of JComponent to give components a tooltip, one of those small windows that appear when the mouse hovers over a component and gives explanatory text.
  6. **Easy Scrolling:** We can connect scrolling to various components-something that was impossible in AWT.

7. **Pluggable Look and Feel:** We can set the appearance of applets and applications to one of three standard looks, i.e., Windows, Motif (Unix) or Metal (Standard swing look).
8. **Highly Customizable**: Swing controls can be customized in very easy way as visual appearance is independent of internal representation.
9. **New Layout Managers:** Swing introduces the BoxLayout and Overlay Layout layout managers.

Working with Swing:

- While designing any GUI, we need to have a main window on which we can place or position the different visual components.
- Every top level container has one intermediate container called content pane. This content pane contains all the visible components in the GUI window.
- In swing, the main window, also called a top level container is the root of a hierarchy, which contains all the other swing components that appear inside the window. All swing applications have at least one top level container.
- The content pane is the base pane upon which all other components or container objects are placed.
- One exception to this rule is, if there is a menu bar in the top level container it will be placed outside the content pane.
- Fig. 5.8 illustrates the relationship between the components discussed above.

Fig. 5.8

- The general procedure to create top-level container is iven below:
  1. Create the container,
  2. Set its size (in the case of frames and panels), and
  3. Set in visibility (in the case of frames and panels).
- All swing components names start with J. For instance the swing button class is named as JButton commonly used top-level containers are:

- Just like AWT application, a Swing application requires a top-level container. There are three top-level containers in Swing:
  1. **JFrame:** Used for the application's main window (with an icon, a title, minimize/maximize/close buttons, an optional menu-bar, and a content-pane), as illustrated.
  2. **JDialog:** Used for secondary pop-up window (with a title, a close button, and a content-pane).
  3. **JApplet:** Used for the applet's display-area (content-pane) inside a browser's window.
- Similarly to AWT, there are secondary containers (such as JPanel) which can be used to group and layout relevant components.
- Panels are an example of intermediate containers.

Advantages of Swing:

1. Swing provides paint debugging support for when you build your own component.
2. Swing components are lightweight than AWT.
3. Swing components follow the Model-View-Controller (MVC) paradigm and thus can provide a much more flexible UI.
4. Swing provides both additional components like JTable, JTree etc and added functionality to replacement of AWT components.
5. Swing provides built-in double buffering.

Disadvantages of Swing:

1. It can be slower than AWT (all components are drawn), when you are not careful about programming.
2. Swing components might not behave exactly like native components which look like native components.
3. It requires Java 2 or a separate JAR file.

5.5.1 JComponent Class

- The class JComponent is the base class for all Swing components except top-level containers.
- To use a component that inherits from JComponent, you must place the component in a containment hierarchy whose root is a top-level Swing container.
- Fig. 5.9 shows the class hierarchy of the swing GUI classes. Every swing controls inherits properties from following Component class hierarchy.
- A Container is the abstract base class for the non menu user-interface controls of swing.
- Component represents an object with graphical representation. A Container is a component that can contain other SWING components.
- A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.

Fig. 5.9

- Fig. 5.9 shows following classes of JComponent
  1. **AbstractButton:** Abstract super-class for Swing buttons.
  2. **ButtonGroup:** Encapsulates a mutually exclusive set of buttons.
  3. **ImageIcon:** Encapsulates an icon.
  4. **JApplet:** The Swing version of Applet.
  5. **JButton:** The Swing push button class.
  6. **JCheckBox:** The Swing check box class.
  7. **JComboBox:** Encapsulates a combo box, (a combination of a dropdownlist and text field).
  8. **JLabel:** The Swing version of a label.
  9. **JRadioButton:** The Swing version of a radio button.
  10. **JScrollPane:** Encapsulates a scrollable window.
  11. **JTabbedPane:** Encapsulates a tabbed window.
  12. **JTable:** Encapsulates a table-based control.
  13. **JTextField:** The Swing version of a text field.
  14. **JTree:** Encapsulates a tree-based control.

5.5.2 Swing Components

- Swing components are not implemented by platform specific code. Instead they are written entirely in Java and therefore, are platform independent.
- Swing component is called light weight, as it does not depend on any non-java system classes. Swing components have their own view supported by java's look and feel classes.
- Swing components have pluggable look and feel so that with a single set of components you can achieve the same look and feel on any operating system platform.

- The swing related classes are contained in javax.swing and its subpackages, such as javax.swing.tree.
- To use a swing class, either use an import statement for a specific class to be imported from within a swing or import all classes in the swing package as:

```
javax.swing.*;
```

- Swing controls or components and their related methods are available in javax.swing and its subpackages.

Fig. 5.10: Component of swing

5.5.3 JFrame

- The frame is a top-level container or window, on which other swing components are placed.
- A JFrame component is used to create windows in a swing program. It extends java.awt.Frame class. Every JFrame object has a root pane.
- Some of the constructors of the JFrame class are shown below:
  1. `JFrame()`: This constructor creates a new frame without title.
  2. `JFrame(string title)`: This constructor constructs the new frame with title.
- When adding components to a JFrame window, we must add it to its content pane and not directly to the JFrame window.
- For example: To add a button for b we would write frame.getContentPane().add(b);

Methods:

1. `Container getContentPane()`: This method returns, a Content Pane for the JFrame.
2. `void setLayout(LayoutManager)`: This method sets the LayoutManager for the JFrame.
3. `void setJMenuBar(JMenuBar)`: This method sets the JMenu Bar to the JFrame.
4. `void setIconImage(Image image)`: Prints icon image on JFrame.

Program 5.18: Program to demonstrate JFrame.

```
import java.awt.*;
import javax.swing.*;
public class frameDemo extends JFrame
   {
      public frameDemo(String title)
         {
         super(title);
         }
         public static void main(String args[])
         {
            frameDemo fd = new frameDemo("First frame");
            fd.setVisible(true);
            fd.setSize(300, 300);
         }
   }
```

Output:

- Here, we have imported the relevant packages necessary to create the frame window. Then in the constructor we invoked super() which in turn calls the constructor of JFrame and creates the frame.
- We then sets the size of the frame and changed its visibility to true in order to display it when it is executed.

5.5.4 JPanel

- The JPanel component which is an intermediate container, is used to group smaller lightweight components together. By default panels are opaque which means that they work similar to content panes.
- JPanel objects have FlowLayout as their default layout. Some of the constructors of JPanel are:
  `JPanel();`
  `JPanel(LayoutManager LM)`
- Most methods of the JPanel API are derived from its super classes, container JComponent and component.
- **Methods:**
  1. `Container getContentPane()`: This method returns, a Content Pane for the JPanel.
  2. `void setLayout(LayoutManager)`: This method sets the LayoutManager for the JPanel.

3. `void setJMenuBar(JMenuBar)`: This method sets the JMenuBar to the JFrame.
4. `void setIconImage(Image image)`: Prints icon image on JFrame.

Program 5.19: Program to demonstrate JPanel.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class JPanelExample extends JFrame
{
   JPanel panel1,panel2 ;
   JTextField txtData;
   JButton[] btnData = new JButton[12];
   JPanelExample()
   {
      panel1=new JPanel();
      txtData = new JTextField(20);
      panel1.add(txtData);
      add(panel1,"North");
      panel2=new JPanel(new GridLayout(3,4));
      for(int i=0;i<=9;i++)
      {
         btnData[i]=new JButton(""+i);
         panel2.add(btnData[i]);
      }
      add(panel2,"Center");
   }
}
class JPanelJavaExample
{
   public static void main(String[] args)
   {
      JPanelExample frame = new JPanelExample();
      frame.setTitle("JPanel Java Example");
      frame.setBounds(100,200,220,200);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.pack();
      frame.setVisible(true);
   }
}
```

Output:

5.5.5 JButton [April 17]

- The JButton class is used to create a push buttons.

  Package: `javax.swing`

  Constructors:

  1. `JButton()` : Create a empty button having no title and icon.
  2. `JButton(String str)` : Create a button having label.
  3. `JButton(Icon i)` : Creates a button having icon image.
  4. `JButton(String str, Icon i)` : Creates a button having string label and icon image.

- This class is subclass of Abstract Button class. So it uses following important methods of abstract button class.

  1. `void addActionListenerActionListener obj)`: This method is used to register the push button to throw an event.
  2. `Siring getText()`:This method returns the text associated with the button.
  3. `void setText(Siring s)`: This method is used to set new Label of the button
  4. `void setHorizontalTextPosition(int pos)`: Sets the horizontal text position relative to the graphics.
  5. `void setVerticalTextPosition(int pos)`: Sets the vertical text position relative to the graphics.
  6. `void set Icon( Icon i)`: This method sets the Icon to the button.
  7. `void setMnemonic(char c)`: Sets the mnemonic character to the button.

Program 5.20: Program of demonstrate JButton.

```
import javax.swing.* ;
import java.awt.*;
class JButtonExample extends JFrame
{
   JButtonExample()
   {
      setLayout(new FlowLayout());
      JButton btnOk = new JButton("OK");
      ImageIcon icon = new ImageIcon("check.png");
      JButton btnIcon = new JButton(icon);
      JButton btnTxtIcon = new JButton("OK",icon);
      add(btnOk);
      add(btnIcon);
      add(btnTxtIcon);
   }
}
```

```
class JButtonJavaExample
{
   public static void main(String args [])
   {
      JButtonExample frame = new JButtonExample();
      frame.setTitle("JButton in Java Swing Example");
      frame.setBounds(200,250,250,100);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Output:

5.5.6 JToggleButton

- A toggle button is two states button that allows user to switch on and off. To create a toggle button in Swing we use JToggleButton class.
- JToggle button, when goes to the inward push state as long as the user has pressed the left mouse key. When he frees the left mouse key, the button comes to its normal state.
- Here, are the most common used constructors of the JToggleButton class:
  1. `public JToggleButton()`: Creates a toggle button without text and icon. The state of toggle button is not selected.
  2. `public JToggleButton(Icon icon)`: Creates a toggle button with icon.
  3. `public JToggleButton(Icon icon, boolean selected)`: Creates a toggle button with icon and initialize the state of toggle button by the boolean parameter selected.
  4. `public JToggleButton(String text)`: Creates a toggle button with text.
  5. `public JToggleButton(String text, boolean selected)`: Creates a toggle button with text and initialize the state of the toggle button.
  6. `public JToggleButton(String text, Icon icon)`: Creates a toggle button which displays both text and icon.
  7. `public JToggleButton(String text, Icon icon, boolean selected)`: Creates a toggle button which displays both text and icon. The state of toggle button can be initialized.

Program 5.21: Program to demonstrate JToggleButton.

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JToggleButton;
```

```
public class ToggleButtonSample
{
   public static void main(String args[])
   {
      JFrame f = new JFrame("JToggleButton Sample");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container content = f.getContentPane();
      content.add(new JToggleButton("North"), BorderLayout.NORTH);
      content.add(new JToggleButton("East"), BorderLayout.EAST);
      content.add(new JToggleButton("West"), BorderLayout.WEST);
      content.add(new JToggleButton("Center"), BorderLayout.CENTER);
      content.add(new JToggleButton("South"), BorderLayout.SOUTH);
      f.setSize(300, 200);
      f.setVisible(true);
   }
}
```

Output:

5.5.7 JLabel

- One of the simplest Swing components is JLabel. A JLabel object is a component for placing text in a container.
- This class is used to create single line read only text which describes the other component.
- Labels can display text as well as Images.

Package: `javax.swing`

Constructors:

1. `JLabel()` : Create a empty label having no text and icon.
2. `JLabel(String str)` : Create a label having some text.
3. `JLabel(ImageIcon i)` : Creates a label having icon image.
4. `JLabel(String str,ImageIcon i)` : Creates a label having string text and icon image.

Methods:

1. `void setHorizontalAlignment(int a)`: This method sets the alignment of the text.
2. `String getText()`: This method returns the text associated with the Label.
3. `void setText( String s)`: This method is used to set the text to the Label.
4. `void set icon(Icon i)`: This method sets the Icon to icon.

Program 5.22: Program to demonstrate JLabel.

```
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.Rectangle;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextArea;
public class Frame1 extends JFrame
{
   private JLabel jLabel1 = new JLabel();
   private JLabel jLabel2 = new JLabel();
   public Frame1()
   {
      try
      {
         jbInit();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }
   }
   private void jbInit() throws Exception
   {
      this.getContentPane().setLayout( null );
      this.setSize( new Dimension(400, 300) );
      jLabel1.setText("UserName");
      jLabel1.setBounds(new Rectangle(40, 55, 80, 25));
      jLabel2.setText("Password");
      jLabel2.setBounds(new Rectangle(40, 95, 60, 25));
      this.getContentPane().add(jLabel2, null);
      this.getContentPane().add(jLabel1, null);
   }
   public static void main(String args[])
   {
      Frame1 frame=new Frame1();
      frame.setVisible(true);
   }
}
```

Output:

JIcons:

- Icons are encapsulated by the ImageIcon class, which prints an icon from an image.
- Constructors are given below:
  1. `ImageIcon(String filename)`: This constructor creates object with image which is specified by filename.
  2. `ImageIcon(URL url)`: This constructor creates object with image in the resource identified by url.
- **Methods:**
  1. `int getIconHeight()`: Returns the height of the icon in pixels.
  2. `int getIconWidth()`: Returns the width of the icon in pixels.
  3. `void paintIcon(Component comp, Graphics g, int x, int y)`: Paints the icon at position x, y on the graphics context g. Additional information about the paint operation can be provided in component.
- Following Program of demonstrate icons.

```
import java.awt.*;
import javax.swing.*;
/*
<applet code="JLabelDemo" width=250 height=150>
</applet>
*/
public class JLabelDemo extends JApplet
{
   public void init()
   {
      // Get content pane
      Container contentPane = getContentPane();
      // Create an icon
      ImageIcon ii = new ImageIcon("Lotus.gif");
      // Create a label
      JLabel jl = new JLabel("Lotus", ii, JLabel.CENTER);
      // Add label to the content pane
      contentPane.add(jl);
   }
}
```

5.5.8 JCheckBox

- A checkbox is a control that may be turned ON or OFF by the user to indicate some option.
- The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.
- The JCheckBox class is used to create CheckBox in Swing. CheckBox has the following constructors:
  1. `JCheckBox()`: Creates an initially unselected check box button with no text, no icon.

2. `JCheckBox(Action a)`: Creates a check box where properties are taken from the Action supplied.
3. `JCheckBox(Icon icon)`: Creates an initially unselected check box with an icon.
4. `JCheckBox(Icon icon, boolean selected)`: Creates a check box with an icon and specifies whether or not it is initially selected.
5. `JCheckBox(String text)`: Creates an initially unselected check box with text.
6. `JCheckBox(String text, boolean selected)`: Creates a check box with text and specifies whether or not it is initially selected.
7. `JCheckBox(String text, Icon icon)`: Creates an initially unselected check box with the specified text and icon.
8. `JCheckBox(String text, Icon icon, boolean selected)`: Creates a check box with text and icon, and specifies whether or not it is initially selected.

Program 5.23: Program to demonstrate JCheckBox.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class HobbyOfStudent extends JPanel implements ActionListener,ItemListener{
JCheckBox ch1 = new JCheckBox("Playing", false);
JCheckBox ch2 = new JCheckBox("Music", false);
JCheckBox ch3 = new JCheckBox("Painting", false);
JCheckBox ch4 = new JCheckBox("Reading", false);
JLabel j1 = new JLabel("What's student hobby?");
JButton exitbtn = new JButton("Exit");
public HobbyOfStudent()
{
   setLayout(new GridLayout(7,1));
   ch1.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
   ch2.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
   ch3.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
   ch4.setFont(new Font("Arial", Font.BOLD | Font.ITALIC, 14));
   ch1.addItemListener(this);
   ch2.addItemListener(this);
   ch3.addItemListener(this);
   ch4.addItemListener(this);
   add(j1);
   add(ch1);
   add(ch2);
   add(ch3);
   add(ch4);
   add(exitbtn);
   exitbtn.addActionListener(this);
}
```

```
public void actionPerformed(ActionEvent e)
{
   if(e.getSource().equals(exitbtn) )
   {
      System.exit(0);
   }
}
public void itemStateChanged(ItemEvent e) {
   String selected= ((JCheckBox)e.getSource()).getText();
      System.out.println(selected);
   }
}
class HobbyTest extends JFrame
{
   HobbyTest()
   {
      super();
      getContentPane().add(new HobbyOfStudent());
      setSize(600,600);
      setVisible(true);
   }
   public static void main(String args[]) {
   new HobbyTest();
   }
}
```

Output:

5.5.9 JTextField

- The JTextField component allows us to enter/edit a single line of text.
- Following are the important constructors of the subclasses of the JTextField class.
  1. `JTextField()`: Constructs a new TextField.
  2. `JTextField(Document doc, String text, int columns)`: Constructs a new JTextField that uses the given text storage model and the given number of columns.

3. `JTextField(int columns)`: Constructs a new empty TextField with the specified number of columns.
4. `JTextField(String text)`: Constructs a new TextField initialized with the specified text.
5. `JTextField(String text, int columns)`: Constructs a new TextField initialized with the specified text and columns.

Program 5.24: Program to demonstrate JTextField.

```
import java.awt.*;
import javax.swing.*;
public class TextFieldDemo extends JFrame
{
   public TextFieldDemo()
   {
      Container con = getContentPane();
      con.setLayout(new FlowLayout());
      JLabel j1 = new JLabel ("TextField");
      con.add(j1);
      JTextField tf1 = new JTextField(40);
      con.add(tf1);
      setSize(600, 600);
      setVisible(true);
   }
   public static void main(String args[])
   {
      new TextFieldDemo();
   }
}
```

Output:

- The above Program 5.25, create a frame window in Swing. We then set its existing layout to FlowLayout. We then create a Swing label along with a text field component and add them to the content pane.

5.5.10 JTextArea [Oct. 16]

- The JTextArea is used to accept several lines of text from the user and it has capabilities not found in the AWT class.
- JTextArea class itself does not handle scrolling but implements the Scrollable interface which allows it to be placed inside a JScrollPane and thus implement scrolling.

- Text wrapping is possible through the use of a bound property of JTextArea class. A JTextArea can be created using one of the following constructors:
  1. `JTextArea()`: Constructs a new TextArea.
  2. `JTextArea(Document doc)`: Constructs a new JTextArea with the given document model and defaults for all of the other arguments (null, 0, 0).
  3. `JTextArea(Document doc, String text, int rows, int columns)`: Constructs a new JTextArea with the specified number of rows and columns and the given model.
  4. `JTextArea(int rows, int columns)`: Constructs a new empty TextArea with the specified number of rows and columns.
  5. `JTextArea(String text)`: Constructs a new TextArea with the specified text displayed.
  6. `JTextArea(String text, int rows, int columns)`: Constructs a new TextArea with the specified text and number of rows and columns.

Program 5.25: Program to demonstrate JTextArea.

```
import java.awt.*;
import javax.swing.*;
public class TextAreaapplextends JFrame
{
   public TextAreaappl()
   {
      Container con = getContentPane();
      con.setLayout(new FlowLayout());
      JLabel j11 = new JLabel ("TextArea");
      con.add(j11);
      JTextArea ta1 = new JTextArea(10, 20);
      con.add(ta1);
      setSize(600, 600);
      setVisible(true);
   }
   public static void main(String args[])
   {
      new TextAreaappl();
   }
}
```

Output:

- The above program creates a standalone Swing application having a frame window with a text area and a label. To create the text area that is ten rows and twenty columns wide we use the constructor:

```
JTextArea ta1= new JTextArea(10, 20);
```

5.5.11 JRadioButton

- This class is used to create radio button with a text or icon. Radio buttons are supported by the JRadioButton class, which is aconcrete implementation of AbstractButton.
- Once, a radio button is created they must be kept in one group, which is created by using the ButtonGroup class Radio buttons must be configured into a group.
- Only one of the buttons in that group can be selected at any time.

Package: `javax.swing`

Constructors:

1. `JRadioButton()`: Create a empty radio button having no title and icon.
2. `JRadioButton (String l)`: Create a radio button having label.
3. `JRadioButton (Icon i)`: Creates a radio button having icon image.
4. `JRadioButton (String l, Icon i)`: Creates a radio button having string label and icon image.
5. `JRadioButton (Siring l, Icon i,boolean selected)`: Creates a radio button having string label and icon image and default selected if true is passed.
6. `JRadioButton (String l, boolean selected)`: Creates a radio button having string label and option for default selected policy.

Button Group:

- This class is useful to group the component such as radio buttons.
- If radio button is not grouped, then they work as checkbox. And multiple selection is possible.
- To avoid that we group the Radio button object. This class provides the default constructor ButtonGroup by using this constructor object of button group class is created.
- Then different component are added to the button group object by using add() method.

Program 5.26: Program to demonstrate JRadioButton.

```
import javax.swing.*;
public class JRadioButtonExample
{
    JRadioButtonExample()
    {
        JFrame jframe = new JFrame();
        JRadioButton jredioButton1 = new JRadioButton("Covid + ");
        JRadioButton jredioButton2 = new JRadioButton("Covid - ",true);
```

```
        jredioButton1.setBounds(75,50,100,30);
        jredioButton2.setBounds(75,100,100,30);
        ButtonGroup buttonGroup = new ButtonGroup();
        buttonGroup.add(jredioButton1);
        buttonGroup.add(jredioButton2);
        jframe.add(jredioButton1);
        jframe.add(jredioButton2);
        jframe.setSize(500,500);
        jframe.setLayout(null);
        jframe.setVisible(true);
    }
    public static void main(String[] args)
    {
       new JRadioButtonExample();
    }
}
```

Output:

5.5.12 JList

- In Swing, the JList component is the implementation of the AWT List class.
- JList consists of a range of elements arranged one after another, which can be selected individually or in a group.
- JList class, unlike its AWT counterpart, is capable of displaying not just the strings, but also icons.
- Some of the constructors used for creating JList are explained below:
  1. `public JList()`: Constructs a JList with an empty model.
  2. `public JList(ListModel dataModel)`: Constructs a JList() that displays the elements in the specified, non-null list model.
  3. `public JList(Object [] listData)`: Constructs a JList() that displays the elements of the specified array "listData".

- JList() does not support scrolling. To create a scrolling list the JList() is implemented as the viewport view of a JScrollPane.

```
JScrollPane myScrollPane = new JScrollPane();
myScrollPane.getViewport().setView(dataList);
```

OR

```
JScrollPane myScrollPane = new JScrollPane(dataList);
```

- JList does not provide any special support for handling double or triple mouse clicks. However, using the MouseListener makes it easy to handle these events. The locationToIndex() method is used to determine the cell that was checked.

Program 5.27: Program to demonstrate JList.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
class Employees extends JFrame implements ListSelectionListener
{
   String Employee[] = {"Amar Salunkhe", "Akbar Shaikh",
   "Amol Mahabal", "Kiran Velankar"};
   JList departments = new JList(Employee);
   JLabel lTE = new
   JLabel("Who is your favourite Technical Editor?");
   JTextField jt = new JTextField(40);
   public Employees(String s) {
      super(s);
      JPanel p = (JPanel)getContentPane();
      p.setLayout(new BorderLayout());
      p.add("North", lTE);
      departments.setSelectionMode(
         ListSelectionModel.SINGLE_SELECTION);
      departments.setSelectedIndex(0);
      departments.addListSelectionListener(this);
      departments.setBackground(Color.red);
      departments.setForeground(Color.black);
      p.setBackground(Color.white);
      p.setForeground(Color.black);
      p.add("Center", new JScrollPane(departments));
      p.add("South", jt);
   }
```

```
    public static void main(String args[])
    {
    Employees e1 = new Employees ("Editor of Engineering Books");
          e1.setSize(500,500);
          e1.show();
    }
    public void valueChanged(ListSelectionEvent e)
    {
       if(e.getValueIsAdjusting() == false)
       { if(departments.getSelectedIndex()!=-1)
          {
          jt.setText((String)departments.getSelectedValue());
          }
       }
    }
}
```

Output:

- In JList, events are handled by implementing the ListSelectionListener interface.
  1. `public void addListSelectionListener(ListSelectionListener listener)`: This method adds a listener to the list.
  2. `public void valueChanged(ListSelectionEvent e)`: This method is called whenever the value of the selection changes.

5.5.13 JComboBox [April 17]

- Swing provides a combo box (a combination of a text field and a dropdown list) through the JComboBox class, which extends JComponent. A combobox normally displays one entry.
- However, it can also display a drop-down list that allows a user to select a different entry. We can also type our selection into the text field.
- The JComboBox component, similar to Choice component in AWT, is a combination of textfield and drop down list of items. User can make selection by clicking at an item from the list or by typing into the box.

- A combo box is a combination of a list component and text field component. It can consist of more than one item, however, displays only one item at any point of time. It also allows user to type their selection.
- Unlike list component, combo box allows user to select only one item at a time. A combo box is an object of JComboBox class.
- Three of JComboBox's constructors are shown here:
  1. `JComboBox()`: This constructor creates an empty JComboBox instance.
  2. `public JComboBox (ComboBoxModel asModel)`: Creates a JComboBox that takes its items from an existing ComboBoxModel. asModel is the ComboBoxModel that provides the displayed list of items.
  3. `public JComboBox (Object[] items)`: Creates a JComboBox that contains the elements of the specified array
- Items are added to the list of choices via the addItem( ) method, whose signature/syntax is shown here:

  ```
  void addItem(Object obj)
  ```

  Here, obj is the object to be added to the combo box.

Methods:

1. `public void setEditable(boolean aFlag)`: It determines whether the JComboBox field is editable or not?
2. `public boolean isEditable()`: It returns true if the JComboBox is editable. By default, a combo box is not editable.
3. `public void setMaximumRowCount(int count)`: It sets the maximum number of rows the JComboBox displays. If the number of objects in the model is greater than 'count', the combo box uses a scrollbar.
4. `public void setSelectedItem(Object anObject)`: It sets the selected item in the combo box display area to the object in the argument. If an Object is in the list, the display area shows an Object selected.
5. `public void insertItemAt(Object anObject, int index)`: It inserts an item 'anObject' into the item list at a given 'index'.
6. `public void removeItem(Object anObject)`: It removes an item 'anObject' from the item list.
7. `public void removeItemAt(int anIndex)`: It removes the item at 'anIndex'.

- The following example contains a combo box and a label. The label displays an icon. The combo box contains entries for colors Green, Red, Yellow and Black.
- When a country is selected, the label is updated to display the color for that particular color. Color jpeg images are already stored in the current directory.

Program 5.28: Program to demonstrate JComboBox.

```
import javax.swing.*;
import java.awt.event.*;
```

```
public class JComboBoxExample
{
   JFrame jframe;
   JComboBoxExample()
   {
      jframe=new JFrame("Example of JComboBox");
      final JLabel jlabel = new JLabel();
      jlabel.setHorizontalAlignment(JLabel.CENTER);
      jlabel.setSize(400,100);
      JButton jbutton=new JButton("Click Me ");
      jbutton.setBounds(200,100,75,20);
      String str[]={"C","C++","Data Science","Java","PHP"};
      final JComboBox jcombox=new JComboBox(str);
      jcombox.setBounds(50, 100,90,20);
      jframe.add(jcombox);
      jframe.add(jlabel);
      jframe.add(jbutton);
      jframe.setLayout(null);
      jframe.setSize(350,350);
      jframe.setVisible(true);
      jbutton.addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
         {
            String str1 = "Programming language Selected: "
            + jcombox.getItemAt(jcombox.getSelectedIndex());
            jlabel.setText(str1);
         }
      });
   }
   public static void main(String[] args)
   {
      new JComboBoxExample();
   }
}
```

Output:

5.5.14 Menu [April 19]

- Menu components in Swing are subclasses of JComponent so they have all functionality of a normal Swing component.
- Fig. 5.11 shows hierarchy of menu.

Menu Hierarchy:

- As we know that every top-level window has a menu bar associated with it. This menu bar consists of various menu choices available to the end user.
- Further each choice contains list of options which is called drop down menus. Menu and MenuItem controls are subclass of MenuComponent class.

Fig. 5.11: Hierarchy of menu

Menu Controls:

1. **JMenuBar:** The JMenuBar object is associated with the top-level window.
2. **JMenuItem:** The items in the menu must belong to the JMenuItem or any of its subclass.
3. **JMenu:** The JMenu object is a pull-down menu component which is displayed from the menu bar.
4. **JCheckboxMenuItem:** JCheckboxMenuItem is subclass of JMenuItem.
5. **JRadioButtonMenuItem:** JRadioButtonMenuItem is subclass of JMenuItem.
6. **JPopupMenu:** JPopupMenu can be dynamically popped up at a specified position within a component.

5.5.14.1 JMenu

- A menu provides a space-saving way to let the user choose one of several options. The Menu class represents pull-down menu component which is deployed from a menu bar.
- In order to create a menu you need to use JMenu class. JMenu class represents the menu which can attach to a menu bar or another menu.
- Menu directly attached to a menu bar is known as top-level menu. If the menu is attached to a menu, it is called sub-menu.
- The constructors of JMenu class are given below:
  1. `JMenu()`: This constructor creates an instance of JMenu without text.
  2. `JMenu(String s)`: This constructor creates an instance of JMenu a given text.
  3. `JMenu(String s, boolean tearOffMenu)`: This constructor creates an instance of JMenu a given text and specify the menu as a tear-off menu or not.
  4. `JMenu(Action a)`: This constructor creates an instance of JMenu whose properties are taken from the specified Action.

Program 5.29: Program for JMenu.

```
import java.awt.event.*;
import javax.swing.*;
public class JMenuExample extends JFrame implements ActionListener
{
   public static void main(String[] s)
   {
      new JMenuExample();
   }
   public JMenuExample()
   {
      super("JMenu Example");
      addWindowListener(new WindowAdapter()
      {
      public void windowClosing(WindowEvent e)
      {
   System.exit(0);
   }
};
```

```
        // Name the JMenu & Add Items
        JMenu menu = new JMenu("File");
        menu.add(makeMenuItem("Open"));
        menu.add(makeMenuItem("Save"));
        menu.add(makeMenuItem("Quit"));
        // Add JMenu bar
        JMenuBar menuBar = new JMenuBar();
        menuBar.add(menu);
        setJMenuBar(menuBar);
        setSize(300, 300);
        setLocation(200, 200);
        setVisible(true);
    }
        public void actionPerformed(ActionEvent e)
        {
            // Menu item actions
            String command = e.getActionCommand();
            if (command.equals("Quit")) {
            System.exit(0);
            } else if (command.equals("Open")) {
            // Open menu item action
            System.out.println("Open menu item clicked");
            } else if (command.equals("Save")) {
            // Save menu item action
            System.out.println("Save menu item clicked");
        }
    }
    private JMenuItem makeMenuItem(String name)
    {
        JMenuItem m = new JMenuItem(name);
        m.addActionListener(this);
        return m;
        }
    }
```

Output:

Menus and Submenus:

- Each menu can also have a submenu. This way we can put similar commands into groups.

Program 5.30: Program for menu and submenus.

```
import javax.swing.*;
public class JMenuTest
{
   JFrame myFrame = null;
   public static void main(String[] a)
   {
      (new JMenuTest()).test();
   }
   private void test()
   {
      myFrame = new JFrame("Menu Test");
      myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myFrame.setBounds(50,50,250,150);
      myFrame.setContentPane(new JDesktopPane());
      JMenuBar myMenuBar = new JMenuBar();
      JMenu myMenu = getFileMenu();
      myMenuBar.add(myMenu);
      myMenu = getColorMenu();
      myMenuBar.add(myMenu);
      myFrame.setJMenuBar(myMenuBar);
      myFrame.setVisible(true);
   }
   private JMenu getFileMenu()
   {
      JMenu myMenu = new JMenu("File");
      JMenu mySubMenu = getOpenMenu();
      myMenu.add(mySubMenu);
      JMenuItem myItem = new JMenuItem("Close");
      myMenu.add(myItem);
      myMenu.addSeparator();
      myItem = new JMenuItem("Exit");
      myMenu.add(myItem);
      return myMenu;
   }
```

```
    private JMenu getColorMenu()
    {
        JMenu myMenu = new JMenu("Color");
        JMenuItem myItem = new JMenuItem("Red");
        myMenu.add(myItem);
        myItem = new JMenuItem("Green");
        myMenu.add(myItem);
        myItem = new JMenuItem("Blue");
        myMenu.add(myItem);
        return myMenu;
    }
    private JMenu getOpenMenu()
    {
        JMenu myMenu = new JMenu("Open");
        JMenuItem myItem = new JMenuItem("Java");
        myMenu.add(myItem);
        myItem = new JMenuItem("HTML");
        myMenu.add(myItem);
        myItem = new JMenuItem("GIF");
        myMenu.add(myItem);
        return myMenu;
    }
}
```

Output:

5.5.14.2 JMenuBar

- The JMenuBar class provides an implementation of a menu bar.
- JMenuBar, which is a drop down menu bar at the top application, and JPopupMenu, a menu you get when you press the button right mouse on a particular area.
- The components of a menu can be constructed using the JMenuBar, JMenu, JMenuItem, JCheckBoxMenuItem and JRadioButtonMenuItem classes in a way similar to the MenuBar, Menu and MenuItem classes of AWT.
- When anyone of the menu items in Menu object "BackGround" is selected, the background colour of the frame is changed to the corresponding colour selected in the menu item.

- In the same way, when any of the menu items in the menu for Foreground is selected, the foreground colour of the frame will change to corresponding colour selected in the menu, which is displayed in the text field. Similarly if the Exit menu item is selected from exit menu, the frame is closed.
- Following are the Constructors and methods of the JMenuBar class:

Name of Methods	Description
`JMenuBar()`	This constuctor constructs a new menu bar.
`JMenu add(Menu menu)`	This method adds the menu specified by a parameter, to the end of the menu bar.
`Component getComponentAtIndex (int index)`	This method returns the component at the specified index passed as a parameter.
`int getComponentIndex(Component c)`	This method returns the index of the component specified by the parameter.
`JMenugetMenu(int index)`	This method returns the menu at the specified index that is passed as a parameter to the function.
`int getMenuCount()`	This method returns a number of menus in the menu bar.

Program 5.31: Program for JMenuBar.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JMenuBarJavaExample extends JFrame implementsActionListener
{
   private JMenuBar MnuBar = new JMenuBar();
   private JMenu MnuOne = new JMenu("File");
   private JMenu MnuTwo = new JMenu("Colors");
   private JMenuItem Exit = new JMenuItem("Exit");
   private JMenu bright = new JMenu("Bright");
   private JMenuItem dark = new JMenuItem("Dark");
   private JMenuItem white = new JMenuItem("White");
   private JMenuItem pink = new JMenuItem("Pink");
   private JMenuItem yellow = new JMenuItem("Yellow");
   private JLabel Lbl = new JLabel("Hello");
   public JMenuBarJavaExample()
   {
      setTitle("Menu Bar In java Swing");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        setLayout(new FlowLayout());
        setJMenuBar(MnuBar);
        MnuBar.add(MnuOne);
        MnuBar.add(MnuTwo);
        MnuOne.add(xit);
        MnuTwo.add(bright);
        MnuTwo.add(dark);
        MnuTwo.add(white);
        bright.add(pink);
        bright.add(yellow);
        xit.addActionListener(this);
        dark.addActionListener(this);
        white.addActionListener(this);
        pink.addActionListener(this);
        yellow.addActionListener(this);
        add(Lbl);
        Lbl.setFont(new Font("Times New Roman", Font.BOLD,22));
    }
    public void actionPerformed(ActionEvent e)
    {
        Object src = e.getSource();
        Container cntnr = getContentPane();
        if(src == xit)
        System.exit(0);
        else if(src == dark)
        cntnr.setBackground(Color.BLACK);
        else if(src == white)
        cntnr.setBackground(Color.WHITE);
        else if(src == pink)
        cntnr.setBackground(Color.PINK);
        else cntnr.setBackground(Color.YELLOW);
        repaint();
    }
    public static void main(String[] de)
    {
        JMenuBarJavaExample frm = newJMenuBarJavaExample();
        final int WIDTH = 270;
        final int HEIGHT = 220;
        frm.setSize(500,500);
        frm.setVisible(true);
    }
}
```

Output:

5.5.14.3 JMenuItem

- The JMenuItem class represents the actual item in a menu. All items in a menu should derive from class JMenuItem, or one of its subclasses. By default, it embodies a simple labeled menu item.
- In order to create menu items in Swing, you need to create new instances of JMenuItem and set different properties for them. You can create menu item with both text and icon.
- **Constructors of JMenuItem:**
  1. `JMenuItem()`: This constructor creates a JMenuItem instance without icon or text.
  2. `JMenuItem(Icon icon)`: This constructor creates a JMenuItem instance with a given icon.
  3. `JMenuItem(String text)`: This constructor creates a JMenuItem instance with a given text.
  4. `JMenuItem(String text, Icon icon)`: This constructor creates a JMenuItem instance with a given text and icon
  5. `JMenuItem(String text, int mnemonic)`: This constructor creates a JMenuItem instance with the given text and keyboard mnemonic.
  6. `JMenuItem(Action a)`: This constructor creates a JMenuItem instance whose properties are taken from the a given Action.

Program 5.32: Program for JMenuItem.

```
import java.awt.event.KeyEvent;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
public class MenuCreation
{
   public static void main(final String args[])
   {
      JFrame frame = new JFrame("MenuSample Example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JMenuBar menuBar = new JMenuBar();
```

```
        // File Menu, F - Mnemonic
        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic(KeyEvent.VK_F);
        menuBar.add(fileMenu);
        // File->New, N - Mnemonic
        JMenuItem newMenuItem = new JMenuItem("New", KeyEvent.VK_N);
        fileMenu.add(newMenuItem);
        frame.setJMenuBar(menuBar);
        frame.setSize(350, 250);
        frame.setVisible(true);
    }
}
```

Output:

5.5.14.4 JPopupMenu

- Another type of a menu is a popup menu. Java Swing has a JPopupMenu class for this functionality.
- It is also called a context menu and usually shown when we right click on a component.
- The idea is to provide only the commands that are relevant in the current context. Say we have an image. By right clicking on the image, we get a popup window with commands to save, scale, or move the image.
- Popup menu represents a menu which can be dynamically popped up at a specified position within a component.
- popup menu is a free-floating menu which associates with an underlying component. This component is called the invoker.
- Most of the time, popup menu is linked to a specific component to display context-sensitive choices.
- In order to create a popup menu, you use the class JPopupMenu. You then can add menu itemsJMenuItem to popup menu like normal menu.
- To display the popup menu, we call method show(). Normally popup menu is called in response to a mouse event.

Constructors:

1. `JPopupMenu()`: This constructor constructs a JPopupMenu without an "invoker".
2. `JPopupMenu(String label)`: This constructor constructs a JPopupMenu with the specified title.

Program 5.33: Program for JPopMenu.

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;
public class PopupSample
{
   public static void main(final String args[])
   {
      JFrame frame = new JFrame("PopupSample Example");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // Create popup menu, attach popup menu listener
      JPopupMenu popupMenu = new JPopupMenu("Title");
      // Cut
      JMenuItem cutMenuItem = new JMenuItem("Cut");
      popupMenu.add(cutMenuItem);
      // Copy
      JMenuItem copyMenuItem = new JMenuItem("Copy");
      popupMenu.add(copyMenuItem);
      // Paste
      JMenuItem pasteMenuItem = new JMenuItem("Paste");
      pasteMenuItem.setEnabled(false);
      popupMenu.add(pasteMenuItem);
      // Separator
      popupMenu.addSeparator();
      // Find
      JMenuItem findMenuItem = new JMenuItem("Find");
      popupMenu.add(findMenuItem);
      JButton label = new JButton();
      frame.add(label);
      label.setComponentPopupMenu(popupMenu);
      frame.setSize(350, 250);
      frame.setVisible(true);
   }
}
```

Output:

5.5.14.5 JCheckBoxMenuItem and JRadioButtonMenuItem

- In this section we will study JCheckboxMenuItem and JRadioButtonMenuItem.

JCheckboxMenuItem:

- The JCheckboxMenuItem class represents a check box which can be included in a menu.
- Selecting the check box in the menu changes control's state from on to off or from off to on.

Constructors:

1. `JCheckboxMenuItem()`: This constructor creates an initially unselected check box menu item with no set text or icon.
2. `JCheckboxMenuItem(Action a)`: This constructor creates a menu item whose properties are taken from the Action supplied.
3. `JCheckboxMenuItem(Icon icon)`: This constructor creates an initially unselected check box menu item with an icon.
4. `JCheckboxMenuItem(String text)`: This constructor creates an initially unselected check box menu item with text.
5. `JCheckboxMenuItem(String text, boolean b)`: This constructor creates a check box menu item with the specified text and selection state.
6. `JCheckboxMenuItem(String text, Icon icon)`: This constructor creates an initially unselected check box menu item with the specified text and icon.
7. `JCheckboxMenuItem(String text, Icon icon, boolean b)`: This constructor creates a check box menu item with the specified text, icon, and selection state.

JRadioButtonMenuItem:

- The JRadioButtonMenuItem class represents a check box which can be included in a menu.
- Selecting the check box in the menu changes control's state from on to off or from off to on.

Constructors:

1. `JRadioButtonMenuItem()`: This constructor creates a JRadioButtonMenuItem with no set text or icon.
2. `JRadioButtonMenuItem(Action a)`: This constructor creates a radio button menu item whose properties are taken from the Action supplied.
3. `JRadioButtonMenuItem(Icon icon)`: This constructor creates a JRadioButtonMenuItem with an icon.
4. `JRadioButtonMenuItem(Icon icon, boolean selected)`: This constructor creates a radio button menu item with the specified image and selection state, but no text.
5. `JRadioButtonMenuItem(String text)`: This constructor creates a JRadioButtonMenuItem with text.
6. `JRadioButtonMenuItem(String text, boolean selected)`: This constructor creates a radio button menu item with the specified text and selection state.
7. `JRadioButtonMenuItem(String text, Icon icon)`: This constructor creates a radio button menu item with the specified text and Icon.

8. `JRadioButtonMenuItem(String text, Icon icon, boolean selected)`: This constructor creates a radio button menu item that has the specified text, image, and selection state.

Program 5.34: Program for JCheckBoxMenuItem and JRadioButtonMenuItem.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class JMenuBar2JavaExample extends JFrame
{
   private JMenuBar MnuBar = new JMenuBar();
   private JMenu MnuOne = new JMenu("File");
   private JCheckBoxMenuItem chkOne =
                                 new JCheckBoxMenuItem("Check Box First");
   private JCheckBoxMenuItem chkTwo =
                                 newJCheckBoxMenuItem("Check Box Second");
   private JRadioButtonMenuItem RdoOne =
                              newJRadioButtonMenuItem("Radio Button One");
   private JRadioButtonMenuItem RdoTwo =
                              newJRadioButtonMenuItem("Radio Button Two");
   private JRadioButtonMenuItem RdoThree =
                            newJRadioButtonMenuItem("Radio Button Three");
   private ButtonGroup BtnGrp = new ButtonGroup();
   public JMenuBar2JavaExample()
   {
      setTitle("Menu Bar2 In java Swing");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new FlowLayout());
      setJMenuBar(MnuBar);
      MnuBar.add(MnuOne);
      MnuOne.add(chkOne);
      MnuOne.add(chkTwo);
      MnuOne.addSeparator();
      MnuOne.add(RdoOne);
      MnuOne.add(RdoTwo);
      MnuOne.add(RdoThree);
      BtnGrp.add(RdoOne);
      BtnGrp.add(RdoTwo);
      BtnGrp.add(RdoThree);
   }
   public static void main(String[] ds)
   {
      JMenuBar2JavaExample frm = newJMenuBar2JavaExample();
      final int WIDTH = 170;
      final int HEIGHT = 220;
      frm.setSize(500,500);
      frm.setVisible(true);
   }
 }
```

Output:

5.5.15 JScrollPane

- A scrolling pane is a container that can be used to hold any component that can be scrolled.
- By default, the list and textarea component do not scroll automatically when number of items in the list or text area component go beyond the displayed area. So to make these components scroll, you must insert them into the scrollpane.
- A scroll pane is a component that presents a rectangular area in which a component may be viewed.
- Horizontal and/or vertical scroll bars may be provided if necessary. Scroll panes are implemented in Swing by the JScrollPane class.

Constructors:

1. `JScrollPane(Component comp)`: This creates a scroll pane with the component specified with comp.
2. `JScrollPane(int vsb, int hsb)`: This creates a scroll panel with the vsb and hsb are int constants that define when vertical and horizontal scroll bars for this scroll panel are shown.
3. `JScrollPane(Component comp, int vsb, int hsb)`: This constructor scroll panel creates a combination of above both constructor.

- Constants of scroll pane are given below:
  1. HORIZONTAL_SCROLLBAR_ALWAYS: Always provide horizontal scroll bar.
  2. HORIZONTAL_SCROLLBAR_AS_NEEDED: Provide horizontal scroll bar, if needed.
  3. VERTICAL_SCROLLBAR_ALWAYS: Always provide vertical scroll bar.
  4. VERTICAL_SCROLLBAR_AS_NEEDED: Provide vertical scroll bar, if needed.
- Here, are the steps that you should follow to use a scroll pane in an applet:
  1. Create a JComponent object.
  2. Create a JScrollPane object, (the arguments to the constructor specify the component and the policies for vertical and horizontal scroll bars.)
  3. Add the scroll pane to the content pane of the applet.

Program 5.35: Program to demonstrate JScrollPane.

```
import javax.swing.*;
import java.awt.*;
class JScrollPaneExample extends JFrame
{
    JScrollPaneExample()
    {
    setLayout(new FlowLayout());
```

```
    String[] language ={"Cobol","Fortran","Pascal","C","C++","java","C#"};
    JLabel lblLanguage = new JLabel("Choose the Language");
    JList LstMonth = new JList(language);
    LstMonth.setVisibleRowCount (4);
    JScrollPane s = newJScrollPane(LstMonth,JScrollPane.VERTICAL_SCROLLBAR_A
                              WAYS,JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
       add(lblLanguage);
       add(s);
    }
}
class JScrollPaneJavaExample
{
    public static void main(String[] args)
    {
       JScrollPaneExample frame = new JScrollPaneExample();
       frame.setTitle("JScrollPane Java Example");
       frame.setBounds(200,250,180,150);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);
     }
 }
```

Output:

5.6 DIALOGS [Oct. 17, April 19]

- Dialog windows or dialogs are an indispensable part of most modern GUI applications.
- A dialog is defined as a conversation between two or more persons. In a computer application a dialog is a window which is used to "talk" to the application.
- A dialog is used to input data, modify data, change the application settings etc. Dialogs are important means of communication between a user and a computer program.

5.6.1 JDialog

- A dialog can be either modal or modeless. Modal dialogs block input to other top-level windows. Modeless dialogs allow input to other windows.
- A modal dialog blocks user input to all other windows in the same application when it is visible.
- We have to close a modal dialog before other windows in the same application can get focus. A modeless one does not block user input.

- This class extends java.awt.Dialog class. Dialogs are used to accept some inputs from User. Default layout manager for JDialog class is BorderLayout.

  Package: `javax.swing`

  Constructors:

  1. `JDialog(Frame parent)`: This constructor creates a new JDialog object which appears on specified parent Frame.
  2. `JDialog (Frame parent, Boolean modal)`: This constructor creates a new JDialog object which appears on the specified parent frame.
- If we pass second parameter as 'true' then we can not work on the parent window when dialog is visible, such a Dialog boxes are called as Modal dialog boxes.
- If we pass 'false" then we can work on the parent window when dialog is visible such a dialog box is called as non-modal dialog box.
  1. `JDialog (Frame parent. String title)`: This constructor creates a dialog box which have some title.
  2. `JDialog (Frame parent. Siring title. Boolean modal)`: This constructor creales a Dialogbox which appears on specified parent frame, with title specified by the user and modal or non-modal nature is specified.
- **Methods:**
  1. `void hide()`: This method is used to hide the Dialog box.
  2. `void show()`: This method is used to show the Dialog box.
  3. `Container getContentPanel()`: This method returns, a Content Pane for the JDialog.
  4. `void setLayout(LayoutManager)`: This method sets the LayoutManager for the Dialog.
  5. `void setJMenuBar(JMenuBar)`: This method sets the JMenu Bar to the JDialog box.
  6. `boolean isModal()`: This method checks whether, the dialog is modal or non-modal.

Program 5.36: Program to demonstrate JDialog.

```
import javax.swing.*;
import java.awt.*;
class JDialogExample extends JFrame
{
   JDialog d1;
   public JDialogExample()
   {
      createAndShowGUI();
   }
   private void createAndShowGUI()
   {
      setTitle("JDialog Example");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLayout(new FlowLayout());
```

```
        // Must be called before creating JDialog for
        // the desired effect
        JDialog.setDefaultLookAndFeelDecorated(true);
        // A perfect constructor, mostly used.
        // A dialog with current frame as parent
        // a given title, and modal
        d1=new JDialog(this,"This is title",true);
        // Set size
        d1.setSize(400,400);
        // Set some layout
        d1.setLayout(new FlowLayout());
        d1.add(new JButton("Button"));
        d1.add(new JLabel("Label"));
        d1.add(new JTextField(20));
        setSize(400,400);
        setVisible(true);
        // Like JFrame, JDialog isn't visible, you'll
        // have to make it visible
        // Remember to show JDialog after its parent is
        // shown so that its parent is visible
        d1.setVisible(true);
    }
    public static void main(String args[])
    {
        new JDialogExample();
    }
}
```

Output:

5.6.2 Message Dialogs [Oct. 16]

- Message dialogs are simple dialogs that provide information to the user. The message is usually a string constant and represents the text contained in the body.
- If message is an array, it will be interpreted as a series of messages (the interpretation is recursive - each object will be interpreted according to its type).
- The class JOptionPane is a component which provides standard methods, to pop up a standard dialog box for a value or informs user of something.

- Following is the declaration for javax.swing.JOptionPane class:

```
public class JOptionPane
extends JComponent
implements Accessible
```

- Message dialogs are created with the JOptionPane.showMessageDialog() method. The messageType defines the style of message. The available options are:

Sr. No.	Symbol	Description
1.		ERROR_MESSAGE for displaying an error message.
2.		INFORMATION_MESSAGE for displaying an informational message
3.		QUESTION_MESSAGE for displaying a query message
4.		WARNING_MESSAGE for displaying a warning message.
5.		PLAIN_MESSAGE for displaying any other type of message.

- JOptionPane provides you with two useful static methods such as showMessageDialog() and showOptionDialog().
- The showMessageDialog() method shows a very simple dialog with one button while showOptionDialog() method displays a highly customized dialogs with different buttons texts.
- Besides these methods, JOptionPaneprovides showConfirmDialog() method to ask users to confirm an action and showInputDialog() to get simple input from the users.

Program 5.37: In this program, we use showMessageDialog() method of JOptionPane to show various simple dialog with one button and different icons which represent different message types such as information, warning, error and question message.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Main
{
   public static void main(String[] args)
   {
      final JFrame frame = new JFrame("JOptionPane Demo");
      // implement ItemListener interface
      class MyItemListener implements ItemListener
      {
         public void itemStateChanged(ItemEvent ev)
```

```
        {
            boolean selected = (ev.getStateChange() == ItemEvent.SELECTED);
            AbstractButton button = (AbstractButton)ev.getItemSelectable();
            String command = button.getActionCommand();
            if (selected)
            {
                int messageType = -1;
                String message = "";
                if (command.equals("INFORMATION"))
                {
                    messageType = JOptionPane.INFORMATION_MESSAGE;
                    message = "Information Message";
                }
                else if (command.equals("WARNING"))
                {
                    messageType = JOptionPane.WARNING_MESSAGE;
                    message = "Warning Message";
                }
                else if (command.equals("ERROR"))
                {
                    messageType = JOptionPane.ERROR_MESSAGE;
                    message = "Error Message";
                } else if (command.equals("QUESTION"))
                {
                    messageType = JOptionPane.QUESTION_MESSAGE;
                    message = "Question Message";
                }
                // show message
                JOptionPane.showMessageDialog(frame,
                message,
                "Message Dialog",messageType);
            }
        }
    }
        JRadioButton r1 = new JRadioButton("Information Message");
        r1.setActionCommand("INFORMATION");
        JRadioButton r2 = new JRadioButton("Warning Message");
        r2.setActionCommand("WARNING");
        JRadioButton r3 = new JRadioButton("Error Message");
        r3.setActionCommand("ERROR");
        JRadioButton r4 = new JRadioButton("Question Message");
        r4.setActionCommand("QUESTION");
        // add event handler
        MyItemListener myItemListener = new MyItemListener();
        r1.addItemListener(myItemListener);
        r2.addItemListener(myItemListener);
        r3.addItemListener(myItemListener);
```

```
            r4.addItemListener(myItemListener);
            // add radio buttons to a ButtonGroup
            final ButtonGroup group = new ButtonGroup();
            group.add(r1);
            group.add(r2);
            group.add(r3);
            group.add(r4);
            // Frame setting
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(300, 200);
            Container cont = frame.getContentPane();
            cont.setLayout(new GridLayout(0, 1));
            cont.add(new JLabel("Please choose the message type:"));
            cont.add(r1);
            cont.add(r2);
            cont.add(r3);
            cont.add(r4);
            frame.setVisible(true);
        }
    }
```

Output:

When we click on each radio button, it will display corresponding message type.

5.6.3 JFileChooser [April 18]

- File chooser dialog box is used for navigating the file system. With file chooser we can choose either file or directory from file or directory list. In order to display file chooser you use JFileChooser API.
- The class JFileChooser is a component which provides a simple mechanism for the user to choose a file.

Constructors:

1. `JFileChooser()`: This constructor constructs a JFileChooser pointing to the user's default directory.
2. `JFileChooser(File currentDirectory)`: This constructor constructs a JFileChooser using the given File as the path.
3. `JFileChooser(File currentDirectory, FileSystemView fsv)`: This constructor constructs a JFileChooser using the given current directory and FileSystemView.
4. `JFileChooser(FileSystemView fsv)`: This constructor constructs a JFileChooser using the given FileSystemView.
5. `JFileChooser(String currentDirectoryPath)`: This constructor constructs a JFileChooser using the given path.
6. `JFileChooser(String currentDirectoryPath, FileSystemView fsv)`: This constructor constructs a JFileChooser using the given current directory path and FileSystemView.

- A JFileChooser is a dialog to select a file or files. The return value of the three methods is one of the following:
  1. JFileChooser.CANCEL_OPTION, if the user clicks Cancel.
  2. JFileChooser.APPROVE_OPTION, if the user click an OK/Open/Save button.
  3. JFileChooser.ERROR_OPTION, if the user closes the dialog.

Program 5.38: Program for JFileChooser.

```
package jfilechooserdemo;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
public class Main
{
   public static void main(String[] args)
   {
      final JFrame frame = new JFrame("JFileChooser Demo");
      final JFileChooser fc = new JFileChooser();
      fc.setMultiSelectionEnabled(true);
      fc.setCurrentDirectory(new File("tmp"));
      JButton btn1 = new JButton("Show Dialog");
      btn1.addActionListener(new ActionListener()
      {
         public void actionPerformed(ActionEvent e)
         {
            fc.showDialog(frame, "Choose");
         }
       });
       JButton btn2 = new JButton("Show Open Dialog");
```

```
        btn2.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                int retVal = fc.showOpenDialog(frame);
                if (retVal == JFileChooser.APPROVE_OPTION)
                {
                    File[] selectedfiles = fc.getSelectedFiles();
                    StringBuilder sb = new StringBuilder();
                    for (int i = 0; i < selectedfiles.length; i++)
                    {
                        sb.append(selectedfiles[i].getName() + "\n");
                    }
                    JOptionPane.showMessageDialog(frame, sb.toString());
                }
            }
        });
        JButton btn3 = new JButton("Show Save Dialog");
        btn3.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                fc.showSaveDialog(frame);
            }
        });
        Container pane = frame.getContentPane();
        pane.setLayout(new GridLayout(3, 1, 10, 10));
        pane.add(btn1);
        pane.add(btn2);
        pane.add(btn3);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

Output:

5.6.4 JColorChooser [April 16]

- The class JColorChooser provides a pane of controls designed to allow a user to manipulate and select a color.

Constructors:

1. `JColorChooser()`: This constructor creates a color chooser pane with an initial color of white.
2. `JColorChooser(Color initialColor)`: This constructor creates a color chooser pane with the specified initial color.
3. `JColorChooser(ColorSelectionModel model)`: This constructor creates a color chooser pane with the specified ColorSelectionModel.

Class Methods:

1. `void addChooserPanel(AbstractColorChooserPanel panel)`: This method adds a color chooser panel to the color chooser.
2. `static JDialog createDialog(Component c, String title, boolean modal, JColorChooser chooserPane, ActionListener okListener, ActionListener cancelListener)`: This method creates and returns a new dialog containing the specified ColorChooser pane along with "OK", "Cancel", and "Reset" buttons.
3. `AccessibleContext getAccessibleContext()`: This method gets the AccessibleContext associated with this JColorChooser.
4. `AbstractColorChooserPanel[] getChooserPanels()`: This method returns the specified color panels.
5. `Color getColor()`: This method gets the current color value from the color chooser.
6. `Boolean getDragEnabled()`: This method gets the value of the dragEnabled property.
7. `JComponent getPreviewPanel()`: This method returns the preview panel that shows a chosen color.
8. `ColorSelectionModel getSelectionModel()`: This method returns the data model that handles color selections.
9. `ColorChooserUI getUI()`: This method returns the L&F object that renders this component.
10. `String getUIClassID()`: This method returns the name of the L&F class that renders this component.
11. `protected String paramString()`: This method returns a string representation of this JColorChooser.
12. `AbstractColorChooserPanel removeChooserPanel(AbstractColorChooserPanel panel)`: This method removes the Color Panel specified.
13. `void setChooserPanels(AbstractColorChooserPanel[] panels)`: This method specifies the Color Panels used to choose a color value.
14. `void setColor(Color color)`: This method sets the current color of the color chooser to the specified color.
15. `void setColor(int c)`: This method sets the current color of the color chooser to the specified color.

16. `void setColor(int r, int g, int b)`: This method sets the current color of the color chooser to the specified RGB color.
17. `void setDragEnabled(boolean b)`: This method sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
18. `void setPreviewPanel(JComponent preview)`: This method sets the current preview panel.
19. `void setSelectionModel(ColorSelectionModel newModel)`: This method sets the model containing the selected color.
20. `void setUI(ColorChooserUI ui)`: This method sets the L&F object that renders this component.
21. `static Color showDialog(Component component, String title, Color initialColor)`: This method shows a modal color-chooser dialog and blocks until the dialog is hidden.
22. `void updateUI()`: This method shows notification from the UIManager that the L&F has changed.

Program 5.39: Program for JColorChooser.

```
package jcolorchooserdemo;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Main
   {
      public static void main(String[] args)
      {
         final JFrame frame = new JFrame("JColorChooser Demo");
         JButton btn1 = new JButton("Choose Color");
         btn1.addActionListener(new ActionListener()
         {
            public void actionPerformed(ActionEvent e)
            {
               Color newColor = JColorChooser.showDialog(
               frame, "Choose Background Color", frame.getBackground());
               if(newColor != null)
               {
                  frame.getContentPane().setBackground(newColor);
               }
            }
         });
         Container pane = frame.getContentPane();
         pane.setLayout(new FlowLayout());
         pane.add(btn1);
         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         frame.setSize(300, 200);
         frame.setVisible(true);
      }
   }
```

Output:

5.7 EVENT HANDLING [April 16, 17, Oct. 16, 17]

- Changing the state of an object is known as an event. For example, click on button, moving the mouse, dragging mouse etc.
- The java.awt.event package provides many event classes and Listener interfaces for event handling. An event is a signal to the program that something has happened.
- Each event must return a Boolean value (true or false), indicating whether the event should be made available to other event handlers.
- An Event is an object that describes a state change in a source. It can be generated as a consequence of a user interacting with the elements in a GUI.
- Any object capable of rising an event is an event source. Event handlers are procedures that are called when a corresponding event occurs.

What is an Event? [April 16]

- Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components.
- The events can be broadly classified into two categories:
  1. **Foreground Events:** Those events which require the direct interaction of user.They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page etc.
  2. **Background Events:** Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires, an operation completion are the example of background events.

What is Event Handling? [April 16, 17]

- The code that performs a task in response to an event is called event handler. Event handling is a process of responding to events that can occur at any time during execution of a program.
- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism has the code which is known as event handler that is executed when an event occurs.

- Java Uses the Delegation Event Model to handle the events. This model defines the standard mechanism to generate and handle the events.
- The Delegation Event Model has the following key participants namely:
  1. An event is a change of state of an object.
  2. The source is an object on which event occurs.
  3. Listener is also known as event handler.Listener is responsible for generating response to an event.

Steps involved in Event Handling:

Step 1 : The User clicks the button and the event is generated.

Step 2 : Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.

Step 3 : Event object is forwarded to the method of registered listener class the method is now get executed and returns.

5.7.1 Delegation Event Model

- Any program that uses GUI such as Java application written for windows is event driven.
- A program contains methods that handle the events. Such a program is called an event-driven program.

Fig. 5.12

- The modern approach to handling events is based on the delegation event model, which defines standard and consistent mechanisms to generate and process events.
- Its concept is quite simple: A source generates an event and sends it to one or more listeners.
- In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.
- Fig. 5.13 shows delegation event model. In this model a GUI component such as Button, MenuItem etc. called as a source, which generates an event.
- The event is passed to one or more listeners. A listener is an object that listens for particular events.
- Fig. 5.13 shows following key participants:

1. **Events:**

- Event is an object that describes a state of change in a source.
- Events may be generated as a consequence of a user interacting with the GUI elements like pressing a button, moving mouse etc.

2. **Event Sources:**

- "Source" is an object that generates an event. In the delegation event model source generates an event and sends it to one or more listeners.

- In this scheme, the listener simply waits until it receives an event. Once received, the listener processes the event and then returns.
- User interface logic is able to delegate the processing of an event to a separate piece of code. In the delegation model, listeners must register with the source in order to receive event notification.
- Notifications are sent only to listeners that want to receive them. Each type of event has its own registration methods.

Fig. 5.13

- The format of registering listener is as:
  `public void addTypeListener (TypeListener t1)`
  where, Type is name of event.
- For example, registration of keyboard event listener is as:
  `addKeyListener(KeyListener t1)`
- The methods that receive and process events are defined in a set of interfaces found in java.awt.event packages.

3. **Listener Interface**:

- It is an interface which contains methods that the listener must implement and the source of the event invokes when the event occurs.

Event Classes:

- When the user interacts with the application by clicking a mouse button or pressing a key an event is generated.
- The Operating System traps this event and the data associated with it. For example, info about time at which the event occurred, the event types (like keypress or mouse click). This data is then passed on to the application to which the event belongs.
- In Java, events are represented by objects, which describe the events themselves. And Java has a number of classes that describe and handle different categories of events.
- The EventObject class is at the top of the event class hierarchy. It belongs to the java.util package.

- While most of the other event classes are present in java.awt.event package. The getSource() method of the EventObject class returns the object that initiated the event. The getId () method returns the nature of the event.
- Following Table 5.1 shows some of important classes that represent events.

Table 5.1

Class	Generated at which Event
1. `WindowEvent`	Window is activated, closed, deactivated, iconified, opened etc.
2. `MouseEvent`	Mouse is moved, clicked, released, dragged, enters or exists component.
3. `MouseWheelEvent`	Mouse wheel is moved.
4. `KeyEvent`	Input is received from Keyboard.
5. `ActionEvent`	A button is pressed, a list item or menu-item is selected.
6. `ItemEvent`	A checkbox or list item is clicked. Also when choice selection is done or checkable menu item is checked or unchecked.
7. `TextEvent`	Text area or text field content is changed.
8. `AdjustmentEvent`	A scroll bar is manipulated.
9. `FocusEvent`	A component gains or loses keyboard focus.

Swing Event Classes:

- The Event classes represent the event. Java provides us various Event classes but we will discuss those which are more frequently used.

EventObject class:

- It is the root class from which all event state objects shall be derived. All Events are constructed with a reference to the object, the source that is logically deemed to be the object upon which the Event in question initially occurred upon. This class is defined in java.util package.
- Following is the **declaration** for java.util.EventObject class:

```
public class EventObject
    extends Object
        implements Serializable
```

Constructor:

1. `EventObject(Object source)`: This constructor constructs a prototypical Event.

Methods:

1. `Object getSource()`: This method is an object on which the Event initially occurred.
2. `String toString()`: This method returns a String representation of this EventObject.

- Following is the list of commonly used event classes:
  1. `AWTEvent`: It is the root event class for all SWING events. This class and its subclasses supercede the original java.awt.Event class.
  2. `ActionEvent`: The ActionEvent is generated when button is clicked or the item of a list is double clicked.
  3. `InputEvent`: The InputEvent class is root event class for all component-level input events.
  4. `KeyEvent`: On entering the character the Key event is generated.
  5. `MouseEvent`: This event indicates a mouse action occurred in a component.
  6. `WindowEvent`: The object of this class represents the change in state of a window.
  7. `AdjustmentEvent`: The object of this class represents the adjustment event emitted by Adjustable objects.
  8. `ComponentEvent`: The object of this class represents the change in state of a window.
  9. `ContainerEvent`: The object of this class represents the change in state of a window.
  10. `MouseMotionEvent`: The object of this class represents the change in state of a window.
  11. `PaintEvent`: The object of this class represents the change in state of a window.

5.7.2 Event Sources and Listeners

- The delegation event model basically has two main parts i.e., Sources and Listeners. Sources are the user interface components that generate the events.
- Such sources are Window, Button, Checkbox, Textbox, Choice, List, MenuItem, Scrollbar, applet etc.
- In the program, Listeners are created by implementing Listener interfaces defined in package java.awt.event.

Event Source:

- An event source is the object that generates an event. This occurs when the internal state of that object changes in some way.
- Sources may generate more than one type of event. A source may register listeners in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method. Example: if you click a button, an ActionEvent Object is generated. The object of the ActionEvent class contains information about the event.
- The job of the event source is to accept registrations, get events from the user and call the listener's event handling method.
- In addition to GUI elements, other components such as an Applet, can generate Events. For e.g. you receive key and mouse events from an Applet.

- Following is the table to describe some of the Event Sources:

Event Sources	Description
1. Button	Generates action events when the buttion is pressed.
2. Checkbox	Generates item events when the check box is selected or deselected.
3. List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
4. Choice	Generates items events when the choice is changed.
5. MenuItem	Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected.
6. Scrollbar	Generates adjustment events when the scrollbar is manipulated.
7. Text components	Generates text events when the user enters a character.
8. Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

Event Listener:

- It is an object that watch for (i.e. listen for) events and handles them when they occur. It is basically a consumer that receives events from the source.
- To sum up, the job of an event listener is to implement the interface, register with the source and provide the event handling.
- The Event listener represents the interfaces responsible to handle events. Java provides us various Event listener classes but we will discuss those which are more frequently used.
- Every method of an event listener method has a single argument as an object which is subclass of EventObject class. For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.
- EventListner interface is a marker interface which every listener interface has to extend.This class is defined in java.util package.
- Following is the declaration for java.util.EventListener interface:

```
public interface EventListener
```

- Following is the list of commonly used event listeners:
  1. `ActionListener`: This interface is used for receiving the action events.
  2. `ComponentListener`: This interface is used for receiving the component events.
  3. `ItemListener`: This interface is used for receiving the item events.
  4. `KeyListener`: This interface is used for receiving the key events.
  5. `MouseListener`: This interface is used for receiving the mouse events.
  6. `WindowListener`: This interface is used for receiving the window events. **[Oct. 18]**
  7. `AdjustmentListener`: This interface is used for receiving the adjusmtent events.

8. `ContainerListener:` This interface is used for receiving the container events.
9. `MouseMotionListener:` This interface is used for receiving the mouse motion events.
10. `FocusListener:` This interface is used for receiving the focus events.

- Some of commonly used listener interfaces are given below:

1. ActionListener Interface:

- The class which processes the ActionEvent should implement this interface.
- The object of that class must be registered with a component. The object can be registered using the addActionListener() method.
- When the action event occurs, that object's actionPerformed method is invoked.

Method:

(i) `void actionPerformed(ActionEvent e):` This method invoked when an action occurs.

2. WindowListener:

- The class which processes the WindowEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addWindowListener() method.

Methods:

(i) `void windowActivated(WindowEvent e):` This method invoked when the Window is set to be the active Window.

(ii) `void windowClosed(WindowEvent e):` This method invoked when a window has been closed as the result of calling dispose on the window.

(iii) `void windowClosing(WindowEvent e):` This method invoked when the user attempts to close the window from the window's system menu.

(iv) `void windowDeactivated(WindowEvent e):` This method invoked when a Window is no longer the active Window.

(v) `void windowDeiconified(WindowEvent e):` This method invoked when a window is changed from a minimized to a normal state.

(vi) `void windowIconified(WindowEvent e):` This method invoked when a window is changed from a normal to a minimized state.

(vii) `void windowOpened(WindowEvent e):` This method invoked the first time a window is made visible.

3. ComponentListener Interface:

- The class which processes the ComponentEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addComponentListener() method. Component event are raised for information only.

Methods:

(i) `void componentHidden(ComponentEvent e):` This method invoked when the component has been made invisible.

(ii) `void componentMoved(ComponentEvent e)`: This method invoked when the component's position changes.

(iii) `void componentResized(ComponentEvent e)`: This method invoked when the component's size changes.

(iv) `void componentShown(ComponentEvent e)`: This method invoked when the component has been made visible.

4. ItemListener Interface:

- The class which processes the ItemEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addItemListener() method.
- When the action event occurs, that object's itemStateChanged method is invoked.

(i) `void itemStateChanged(ItemEvent e)`: This method invoked when an item has been selected or deselected by the user.

5. KeyListener Interface:

- The class which processes the KeyEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addKeyListener() method.

(i) `void keyPressed(KeyEvent e)`: This method invoked when a key has been pressed.

(ii) `void keyReleased(KeyEvent e)`: This method invoked when a key has been released.

(iii) `void keyTyped(KeyEvent e)`: This method invoked when a key has been typed.

6. MouseListener Interface:

- The class which processes the MouseEvent should implement this interface. The object of that class must be registered with a component.
- The object can be registered using the addMouseListener() method.

(i) `void mouseClicked(MouseEvent e)`: This method invoked when the mouse button has been clicked (pressed and released) on a component.

(ii) `void mouseEntered(MouseEvent e)`: This method invoked when the mouse enters a component.

(iii) `void mouseExited(MouseEvent e)`: This method invoked when the mouse exits a component.

(iv) `void mousePressed(MouseEvent e)`: This method invoked when a mouse button has been pressed on a component.

(v) `void mouseReleased(MouseEvent e)`: This method invoked when a mouse button has been released on a component.

7. AdjustmentListener Interface:

- The interfaceAdjustmentListener is used for receiving adjustment events. The class that process adjustment events needs to implements this interface.

Method:

(i) `void adjustmentValueChanged(AdjustmentEvent e)`: This method invoked when the value of the adjustable has changed.

8. ContainerListener Interface:

- The interfaceContainerListener is used for receiving container events.
- The class that process container events needs to implements this interface.
  (i) `void componentAdded(ContainerEvent e):` This method invoked when a component has been added to the container.
  (ii) `void componentRemoved(ContainerEvent e):` This method invoked when a component has been removed from the container.

9. MouseMotionListener Interface:

- The interface MouseMotionListener is used for receiving mouse motion events on a component.
- The class that process mouse motion events needs to implements this interface.
  (i) `void mouseDragged(MouseEvent e):` This method invoked when a mouse button is pressed on a component and then dragged.
  (ii) `void mouseMoved(MouseEvent e):` This method invoked when the mouse cursor has been moved onto a component but no buttons have been pushed.

10. FocusListener Interface:

- The interface FocusListener is used for receiving keyboard focus events.
- The class that process focus events needs to implements this interface.
  (i) `void focusGained(FocusEvent e):` This method invoked when a component gains the keyboard focus.
  (ii) `void focusLost(FocusEvent e):` This method invoked when a component loses the keyboard focus.

5.8 ADAPTERS [April 17, Oct. 17]

- Adapter pattern is frequently used in modern Java frameworks. It comes into place when we want to use an existing class, and its interface does not match the one we need, or we want to create a reusable class that cooperates with unrelated classes with incompatible interfaces.
- Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.
- This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.
- A real life example could be a case of card reader which acts as an adapter between memory card and a laptop.
- We plugin the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.
- The Adapter idea can be demonstrated with the following simple example. The purpose of the sample problem is to adapt an orange as an apple.
- From the lower diagram, the adapter contains an instance of Orange, and extends Apple. It seems to be that after an Orange object gets a adapter skin, it acts like an Apple object now.
- Adapters are abstract classes for receiving various events. The methods in these classes are empty. This class exists as convenience for creating listener objects.

Fig. 5.14

Adapter Class:

- An adapter class can simplify the creation of event handlers in certain situations.
- An adapter class provides an empty implementation of all methods in an event listener interface i.e this class itself write definition for methods which are present in particular event listener interface. However these definitions do not affect program flow or meaning at all.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- We can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which we are interested.
- For example, suppose we want to use MouseClicked Event or method from MouseListener, if we do not use adapter class then unnecessarily we have to define all other methods from MouseListener such as MouseReleased, MousePressed etc.
- But if we use adapter class then we can only define MouseClicked method and don't worry about other method definition because class provides an empty implementation of all methods in an event listener interface.
- Below table indicates Listener interface with their respective adapter class:

Listener Interface	Adapter Class
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdapter
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

Swing Adapters:

- Following is the list of commonly used adapters while listening GUI events in Swing:
  1. **FocusAdapter:** An abstract adapter class for receiving focus events.
  2. **KeyAdapter:** An abstract adapter class for receiving key events.
  3. **MouseAdapter:** An abstract adapter class for receiving mouse events.
  4. **MouseMotionAdapter:** An abstract adapter class for receiving mouse motion events.
  5. **WindowAdapter:** An abstract adapter class for receiving window events.
- Program 5.42 shows use of adapter class which handles empty implementation of event methods which are not provided.

Program 5.40: Use of MouseAdapter class.

```
import java.awt.*;
import java.awt.event.*;
public class JMouseAdapterExample extends MouseAdapter
{
    Frame frame;
    JMouseAdapterExample()
    {
        frame = new Frame("My Mouse Adapter");
        frame.addMouseListener(this);
        frame.setSize(300,300);
        frame.setLayout(null);
        frame.setVisible(true);
    }
    public void mouseClicked(MouseEvent me)
    {
        Graphics g = frame.getGraphics();
        g.setColor(Color.ORANGE);
        g.fillOval(me.getX(),me.getY(),30,30);
    }
    public static void main(String[] args)
    {
        new JMouseAdapterExample();
    }
}
```

Output:

5.9 ANONYMOUS INNER CLASSES [April 16, 19]

- Anonymous classes of Java are called anonymous because they have no name.
- A class that have no name is known as anonymous inner class in Java. An anonymous class is a local class without a name.
- An anonymous class is defined and instantiated in a single succinct expression using the new operator.
- While a local class definition is a statement in a block of Java code, an anonymous class definition is an expression, which means that it can be included as part of a larger expression, such as a method call.
- When a local class is used only once, consider using anonymous class syntax, which places the definition and use of the class in exactly the same place.

When to Use an Anonymous Class?

- In general, we should consider using an anonymous class instead of a local class if:
  - The class has a very short body.
  - Only one instance of the class is needed.
  - The class is used right after it is defined.
  - The name of the class does not make your code any easier to understand.
- Anonymous classes in Java are more accurately known as anonymous inner classes. The fact that they are anonymous inner classes means that they are defined insider another class.
- An anonymous inner class also has unusual syntax. Anonymous classes enable us to make the code more concise. They enable us to declare and instantiate a class at the same time.
- They are like local classes except that they do not have a name. Use them if we need to use a local class only once.

Program 5.41: Program for anonymous class.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BoxLayout;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class AnonymousClassAppUseDemo
{
   public static void main(String[] args)
   {
      GreetingFrame hwf = new GreetingFrame();
      hwf.CreateFrame();
   }
}
class GreetingFrame
{
   private JFrame frame;
   private JPanel messagePanel;
   private JPanel buttonPanel;
   private JPanel container;
   private JLabel messageLabel;
   private JButton greetButton;
   private JButton clearButton;
   public GreetingFrame ()
   {
      //beautify the frame
      JFrame.setDefaultLookAndFeelDecorated(true);
      frame = new JFrame("Hello World");
      messagePanel = new JPanel(); //used to place greeting message
      buttonPanel = new JPanel(); //used to place command button
      container = new JPanel(); //used to place message and button Panels
      messageLabel = new JLabel(" "); // contain greeting message
      greetButton = new JButton("Greeting"); //display greeting message
      when clicked
      clearButton = new JButton("Clear"); //clears greeting message when
      clicked
   }
   public void CreateFrame ()
   {
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
      messagePanel.add(messageLabel);
      buttonPanel.add(greetButton);
      buttonPanel.add(clearButton);
      /* brand new class of super type ActionListener
      * will be created anonymously and passed to addActionListener method
```

```
        * of greetButton object
        */
        greetButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent e)
           {
               messageLabel.setText("Hello User! Greetings of the day!");
           }
        });
        /* brand new class of super type ActionListener
        * will be created anonymously and passed to addActionListener method
        * of clearButton object
        */
        clearButton.addActionListener(new ActionListener()
        {
           public void actionPerformed(ActionEvent e)
           {
              messageLabel.setText(" ");
           }
        });
        container.add(messagePanel);
        container.add(buttonPanel);
        frame.getContentPane().add(container);
        //Display the window.
        frame.setSize(300, 200);
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
     }
  }
```

Output:

Benefits:

1. The key benefit of an anonymous class is encapsulation (or clutter reduction).
2. An anonymous class is, in a sense, the ultimate in private object oriented encapsulation.
3. It is defined exactly where it is needed, it can never be used anywhere else, and it has totally local scope.
4. Anonymous classes have access to all data and methods of their containing classes, including private members; meaning that for small highly localized tasks, they may require less initialization.

Drawbacks:

1. The big drawback with anonymous classes is they can't have explicit constructors.
2. We can't pass them any parameters when they are instantiated.

PROGRAMS

Program 1: Program for JFrame.

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class JFrameExample
{
   public static void main(String s[])
   {
      JFrame jframe = new JFrame("Example of JFrame");
      JPanel jpanel = new JPanel();
      jpanel.setLayout(new FlowLayout());
      JLabel jlabel = new JLabel("First JFrame Example");
      JButton jbutton = new JButton();
      jbutton.setText(" Click Me ");
      jpanel.add(jlabel);
      jpanel.add(jbutton);
      jframe.add(jpanel);
      jframe.setSize(400, 300);
      jframe.setLocationRelativeTo(null);
      jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jframe.setVisible(true);
   }
}
```

Output:

Program 2: Program for JLabel.

```
import javax.swing.*;
class JLabelExample
{
   public static void main(String args[])
```

```
    {
        JFrame jframe= new JFrame("JLabel Example");
        JLabel jlable1,jlable2;
        jlable1=new JLabel("My First Label.");
        jlable1.setBounds(50,100, 100,40);
        jlable2=new JLabel("My Second Label.");
        jlable2.setBounds(50,150, 100,40);
        jframe.add(jlable1);
        jframe.add(jlable2);
        jframe.setSize(400,300);
        jframe.setLayout(null);
        jframe.setVisible(true);
     }
}
```

Output:

Program 3: Program for JTextField.

```
import javax.swing.*;
class JTextFieldExample
{
    public static void main(String args[])
    {
        JFrame jframe = new JFrame("JTextField Example");
        JTextField jtextfield1,jtextfield2;
        jtextfield1 = new JTextField("Welcome to JAVA.");
        jtextfield1.setBounds(50,100, 200,30);
        jtextfield2 = new JTextField("Welcome to Swing.");
        jtextfield2.setBounds(50,150, 200,30);
        jframe.add(jtextfield1);
```

```
        jframe.add(jtextfield2);
        jframe.setSize(400,400);
        jframe.setLayout(null);
        jframe.setVisible(true);
     }
}
```

Output:

Program 4: Program to display the language name using JCheckBox.

```
import javax.swing.*;
public class JCheckBoxExample
{
   JCheckBoxExample()  //constructor
   {
      JFrame jframe= new JFrame("My CheckBox Example");
      JCheckBox jcheckBox1 = new JCheckBox("C");
      jcheckBox1.setBounds(100,100, 150,50);
      JCheckBox jcheckBox2 = new JCheckBox("OOP");
      jcheckBox2.setBounds(100,150, 150,50);
      JCheckBox jcheckBox3 = new JCheckBox("Java", true);
      jcheckBox3.setBounds(100,200, 150,50);
      jframe.add(jcheckBox1);
      jframe.add(jcheckBox2);
      jframe.add(jcheckBox3);
      jframe.setSize(400,400);
      jframe.setLayout(null);
      jframe.setVisible(true);
   }
   public static void main(String args[])
   {
      new JCheckBoxExample();
   }
}
```

Output:

Program 5: Program to display the city name using JList.

```
import javax.swing.*;
public class JListExample
{
   JListExample()
   {
      JFrame jframe= new JFrame();
      JLabel jlabel = new JLabel();
       String data = "Popular Cities  ";
       DefaultListModel<String> list = new DefaultListModel<>();
       list.addElement("Delhi");
       list.addElement("Pune");
       list.addElement("Mumbai");
       list.addElement("Goa");
       JList<String> jlist = new JList<>(list);
       jlist.setBounds(100,100, 75,75);
       jlabel.setText(data);
       jlabel.setSize(500,100);
       jframe.add(jlist);
       jframe.add(jlabel);
      jframe.setSize(400,400);
       jframe.setLayout(null);
       jframe.setVisible(true);
   }
   public static void main(String args[])
   {
      new JListExample();
   }
}
```

Output:

Program 6: Program to display the menu and submenu.

```
import javax.swing.*;
class JMenuExample
{
    JMenu jmenu, jsubmenu;
    JMenuItem i1, i2, i3, i4, i5;
    JMenuExample()
    {
        JFrame jframe = new JFrame("Example of Menu ");
        JMenuBar jmenubar = new JMenuBar();

        jmenu=new JMenu("Menu");
        jsubmenu=new JMenu("Sub Menu");
        i1=new JMenuItem("File");
        i2=new JMenuItem("Edit");
        i3=new JMenuItem("Format");
        i4=new JMenuItem("New");
        i5=new JMenuItem("Open");
        jmenu.add(i1);
        jmenu.add(i2);
        jmenu.add(i3);
        jsubmenu.add(i4);
        jsubmenu.add(i5);
        jmenu.add(jsubmenu);
        jmenubar.add(jmenu);
        jframe.setJMenuBar(jmenubar);
        jframe.setSize(400,400);
        jframe.setLayout(null);
        jframe.setVisible(true);
    }
    public static void main(String args[])
    {
        new JMenuExample();
    }
}
```

Output:

Program 7: Program to create open or save dialog using JFileChooser.

```
import java.io.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.filechooser.*;

class JfilechooserExample extends JFrame implements ActionListener
{
   static JLabel jlabel;
   // a default constructor
   JfilechooserExample()
   {
   }
   public static void main(String args[])
   {
      // frame to contains GUI elements
      JFrame jframe = new JFrame("File Chooser Example");
      jframe.setSize(400, 400);
      jframe.setVisible(true);
      jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton jbutton1 = new JButton("OPEN");
      JButton jbutton2 = new JButton("SAVE");
      JfilechooserExample file1 = new JfilechooserExample();
                                 // make an object of the class filechooser
      jbutton1.addActionListener(file1);
      jbutton2.addActionListener(file1);
      JPanel jpanel = new JPanel();
         jpanel.add(jbutton1);
```

```
            jpanel.add(jbutton2);
            jlabel = new JLabel("File Selected");
            jpanel.add(jlabel);
            jframe.add(jpanel);
            jframe.show();
        }
        public void actionPerformed(ActionEvent ae)
        {
           String comm = ae.getActionCommand();
           if (comm.equals("SAVE"))
           {
              JFileChooser jfile = new JFileChooser(FileSystemView.
                                      getFileSystemView().getHomeDirectory());
              int ro = jfile.showSaveDialog(null);
              if (ro == JFileChooser.APPROVE_OPTION)
              {
                 jlabel.setText(jfile.getSelectedFile().getAbsolutePath());
              }
              else
              jlabel.setText("User Not Select");
           }
           else
           {
              JFileChooser jfile1 = newJFileChooser(FileSystemView.
                                      getFileSystemView().getHomeDirectory());
              int ro1 = jfile1.showOpenDialog(null);
              if (ro1 == JFileChooser.APPROVE_OPTION)
              {
                 jlabel.setText(jfile1.getSelectedFile().getAbsolutePath());
              }
              else
              jlabel.setText("User Not Select");
           }
        }
    }
```

Output:

Program 8: Program to display the message using JOptionPane.

```
import javax.swing.*;
public class JOptionPaneExample
{
   JFrame jframe;
   JOptionPaneExample()
   {
      jframe=new JFrame();
      JOptionPane.showMessageDialog(jframe,"Welcome JAVA World...");
   }
   public static void main(String[] args)
   {
      new JOptionPaneExample();
   }
}
```

Output:

Program 9: Program to display ten buttons with labels Ten, Twenty,....., Hundred using flow layout. Use array of Buttons.

```
import java.awt.*;
import java.awt.event.*;
class FlowLayoutDemo extends Frame
{
   Label label;
   public FlowLayoutDemo()
   {
      String[] names = {"Ten", "Twenty", "Thirty", "Forty",
      "Fifty","Sixty","Seventy","Eighty","Ninety","Hundred"};
      Button b1[] = new Button[names.length];
```

```
        setTitle("FlowLayout ");
        setSize(400,400);
        setLayout(new FlowLayout());
        setVisible(true);
        for (int i = 0; i < b1.length; i++)
        {
           b1[i] = new Button(names[i]);
        }
        for (int i = 0; i < b1.length; i++)
        {
           add(b1[i]);
        }
           add(label);
     }
     public static void main(String args[])
     {
        new FlowLayoutDemo();
     }
  }
```

Output:

Program 10: Program that displays three concentric circles where ever the user clicks the mouse on a frame. The program must exit when user clicks 'X' on the frame.

```
import java.awt.Graphics;
import javax.swing.JPanel;
import java.awt.*;
import java.applet.*;
import javax.swing.*;
import javax.swing.JFrame;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.*;
```

```
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.JPanel;
public class Ovals extends JPanel
{
   private Graphics g;
   private int prevX, prevY;
   private String drawtype;
   public Ovals()
   {
      addMouseListener(new MouseAdapter()
      {
         public void mousePressed(MouseEvent me)
         {
            selectpaint();
         }
       });
   }
   private void selectpaint()
   {
      g = getGraphics();
      Dimension d = getSize();
          int x = d.width/2;
          int y = d.height/2;
          int r1 = (int) ((d.width < d.height)? 0.4 *
                                                d.width: 0.4 * d.height);
          g.setColor(Color.red);
          g.fillOval(x-r1, y-r1, 2*r1, 2*r1);
          int r2 = (int) ((d.width < d.height)? 0.3 *
                                                d.width: 0.3 * d.height);
          g.setColor(Color.blue);
          g.fillOval(x-r2, y-r2, 2*r2, 2*r2);
          int r3 = (int) ((d.width < d.height)? 0.2 *
                                                d.width: 0.2 * d.height);
          g.setColor(Color.yellow);
          g.fillOval(x-r3, y-r3, 2*r3, 2*r3);
      // to choose polygon to draw
   }
```

```
    public static void main( String[] args)
    {
        Ovals ovalsPanel = new Ovals();
        JFrame newFrame = new JFrame();
        newFrame.getContentPane().add(new Ovals());
        newFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        // newFrame.add( ovalsPanel );
        newFrame.setSize( 550, 550 );
        newFrame.setVisible( true );
    }
}
```

Output:

Program 11: Program to design a frame having a text box and a label. As the user types characters in the text box simultaneously display these type characters in the label. Your program should exit when user presses the 'X' button of the window.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class f2
{
    JTextField txtsub = new JTextField(20);
    private JFrame mainFrame;
    private JLabel headerLabel;
    private JLabel headerLabel1;
    private JLabel statusLabel;
    private JPanel controlPanel;
    JButton quitButton = new JButton("Quit");
    public f2()
    {
        prepareGUI();
    }
```

```
public static void main(String[] args)
{
   f2  f2 = new f2();
   f2.showCheckBoxDemo();
}
private void prepareGUI()
{
   mainFrame = new JFrame("Java Swing Examples");
   mainFrame.setSize(400,400);
   mainFrame.setLayout(new GridLayout(5, 4));
   mainFrame.addWindowListener(new WindowAdapter()
   {
      public void windowClosing(WindowEvent windowEvent)
      {
         System.exit(0);
      }
   });
   headerLabel1 = new JLabel("Display Text", JLabel.CENTER);
   headerLabel = new JLabel("", JLabel.CENTER);
   statusLabel = new JLabel("",JLabel.CENTER);
   controlPanel = new JPanel();
   controlPanel.setLayout(new FlowLayout());
   mainFrame.add(headerLabel1);
   mainFrame.add(controlPanel);
   mainFrame.add(statusLabel);
   mainFrame.add(headerLabel);
   mainFrame.add(quitButton);
   mainFrame.setVisible(true);
}
private void showCheckBoxDemo()
{
   headerLabel.setText("Display Text");
   controlPanel.add(txtsub);
   mainFrame.setVisible(true);
   KeyHandler keyH = new KeyHandler();
    txtsub.addKeyListener(keyH);
```

```
        quitButton.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                System.exit(0);
            }
        });
    }
    private class KeyHandler implements KeyListener
    {
        public void keyPressed(KeyEvent e)
        {
        }
        public void keyReleased(KeyEvent e)
        {
        }
        public void keyTyped(KeyEvent e)
        {
            if(e.getSource() == txtsub)
            {
                String passwordTyped = txtsub.getText();
                headerLabel.setText(passwordTyped);
            }
        }
    }
}
```

Output:

Program 12: Program to design a following GUI (use Swing).

```
import java.awt.*;
import javax.swing.*;
class Sample extends JFrame
{
   JLabel l1,l2,l3,l4,l5,l6;
   JTextField t1,t2,t3;
   JTextArea t;
   JPanel p,p1,p2,p3;
   ButtonGroup bg;
   JRadioButton m,f;
   JCheckBox c1,c2,c3;
   JButton b1,b2;
   Sample()
   {
      p =new JPanel();
      p1=new JPanel();
      l1=new JLabel("First Name: ");
      l2=new JLabel("last Name: ");
      l3=new JLabel("Address: ");
      l4=new JLabel("Mobile No.: ");
      t1=new JTextField(10);
      t2=new JTextField(10);
      t3=new JTextField(10);
      t=new JTextArea(2,10);
      p.add(l1);  p.add(t1);
      p.add(l2);  p.add(t2);
      p.add(l3);  p.add(t);
      p.add(l4);  p.add(t3);
      p.setLayout(new GridLayout(4,2));
```

```
        add(p);
        l5=new JLabel("Gender ");
        m = new JRadioButton("male");
        f = new JRadioButton("female");
        bg = new ButtonGroup();
        bg.add(m);
        bg.add(f);
        p1.add(l5);
        p1.add(m);
        p1.add(f);
        p1.setLayout(new GridLayout(1,3));
        p2=new JPanel();
        l6=new JLabel("Your Interests  ");
        c1=new JCheckBox("Computer");
        c2=new JCheckBox("Sports");
        c3=new JCheckBox("Music");
        p2.add(l6);
        p2.add(c1);
        p2.add(c2);
        p2.add(c3);
        p2.setLayout(new GridLayout(1,4));
        p3=new JPanel();
        b1=new JButton("submit");
        b2=new JButton("clear");
        p3.add(b1);
        p3.add(b2);
        add(p);
        add(p1);
        add(p2);
        add(p3);
        setSize(300,400);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String a[])
    {
        new Sample();
    }
}
```

Output:

Program 13: Program for gcd.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
class Demogcd extends JFrame implements ActionListener
{
    JMenu m1,m2;
    JMenuBar mb;
    JMenuItem m[];
    StringBuffer ss=new StringBuffer();
    int n;
    int arr[]=new int [20];
    int n1,n2,gcd,pow;
    public Demogcd()
    {
        mb=new JMenuBar();
        m1=new JMenu("Operation");
        m2=new JMenu("Compute");
        String str[]={"Accept","Exit","GCD","Power"};
        m=new JMenuItem[str.length];
        for(int i=0;i<str.length;i++)
        {
            m[i]=new JMenuItem(str[i]);
            m[i].addActionListener(this);
        }
        mb.add(m1);
        mb.add(m2);
        m1.add(m[0]);
```

```
        m1.add(m[1]);
        m2.add(m[2]);
        m2.add(m[3]);
        setLayout(new BorderLayout());
        add(mb,BorderLayout.NORTH);
        setSize(300,150);
        setVisible(true);
        setLocation(500,200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    } // constructor
    public void actionPerformed(ActionEvent e)
    {
        String s=e.getActionCommand();    //return the name of menu
        if(s.equals("Exit"))
        System.exit(0);
        if(s.equals("Accept"))
        {
            n1=Integer.parseInt(JOptionPane.showInputDialog("Enter 1st no for
                                                                Number"));
            n2=Integer.parseInt(JOptionPane.showInputDialog("Enter 2nd no for
                                                                Number"));
        }
        if(s.equals("GCD"))
        {
            int min;
            if(n1>=n2)
            min=n1;
            else
            min=n2;
            for(int i=1;i<=min;i++)
            {
                if(n1%i==0 && n2%i==0)
                gcd=i;
            }
            JOptionPane.showMessageDialog(null,"GCD = "+gcd);
        }
```

```
            if(s.equals("Power"))
            {
                pow=1;

                for(int i=1;i<=n2;i++)
                {
                    pow=pow*n1;
                }
                JOptionPane.showMessageDialog(null,"Power = "+pow);
            }
        }
        public static void main(String a[])
        {
            new Demogcd();
        }
    }
```

Output:

PRACTICE QUESTIONS

Q.I Multiple Choice Questions:

1. Which object can be constructed to show any number of choices in the visible Window?
   (a) Labels (b) Choice
   (c) List (d) Checkbox
2. Which is a component in AWT that can contain other components like buttons, textfields, labels etc.?
   (a) Window (b) Container
   (c) Panel (d) Frame
3. Which of these events will be notified if scroll bar is manipulated?
   (a) ActionEvent (b) ComponentEvent
   (c) AdjustmentEvent (d) WindowEvent
4. How many types of controls does AWT support?
   (a) 7 (b) 6
   (c) 5 (d) 8

5. What will be the output of line 5

```
1. Choice c1 = new Choice();
2. c1.add("First");
3. c1.addItem("Second");
4. c1.add("Third");
5. System.out.println(c1.getItemCount());
```

(a) 1 (b) 2
(c) 3 (d) 4

6. To use the ActionListener interface it must be implemented by a class there are several ways to do that find in the following?
   (a) Creating a new class
   (b) Using the class the graphical component
   (c) An anonymous inner class
   (d) All mentioned above
7. What is an event in delegation event model used by Java programming language?
   (a) An event is an object that describes a state change in a source.
   (b) An event is an object that describes a state change in processing.
   (c) An event is an object that describes any change by the user and system.
   (d) An event is a class used for defining object, to create events.
8. Which of these methods are used to register a mouse motion listener?
   (a) addMouse() (b) addMouseListener()
   (c) eventMouseMotionListener() (d) addMouseMotionListner()
9. Which of these methods is used to obtain the object that generated a WindowEvent?
   (a) getMethod() (b) getWindow()
   (c) getWindowEvent() (d) getWindowObject()
10. Which class can be used to represent a checkbox with a textual label that can appear?
    (a) Menu (b) MenuBar
    (c) MenuItem (d) CheckboxMenuItem
11. What does AWT stands for?
    (a) All Writing Tools (b) Abstract Window Toolkit
    (c) Abstract Writing Toolkit (d) All Window Tools
12. What is the super class of all components of Java?
    (a) java.all.Component (b) all.awt.Component
    (c) java.awt.Component (d) awt.Component
13. Which of the following methods can be used to change the size of a java.awt.Component object?
    (i) dimension() (ii) setSize()
    (iii) area() (iv) size()
    (v) resize()
    (a) (i), (ii), (iii) & (v) (b) (ii) & (v)
    (c) (i), (ii) & (v) (d) (ii), (iv) & (v)

14. Which of these events is generated by scroll bar?
    (a) ActionEvent (b) KeyEvent
    (c) WindowEvent (d) AdjustmentEvent
15. Which of these methods is used to obtain the object that generated a WindowEvent?
    (a) getMethod() (b) getWindow()
    (c) getWindowEvent() (d) getWindowObject()
16. Which of these are constants defined in WindowEvent class?
    (a) WINDOW_ACTIVATED (b) WINDOW_CLOSED
    (c) WINDOW_DEICONIFIED (d) All of the mentioned
17. Which of these is superclass of WindowEvent class?
    (a) WindowEvent (b) ComponentEvent
    (c) ItemEvent (d) InputEvent
18. Which of these events is generated when the window is closed?
    (a) TextEvent (b) MouseEvent
    (c) FocusEvent (d) WindowEvent
19. Which of these interfaces define a method itemStateChanged()?
    (a) ComponentListener (b) ContainerListener
    (c) ActionListener (d) ItemListener
20. Which of these interfaces define four methods?
    (a) ComponentListener (b) ContainerListener
    (c) ActionListener (d) InputListener
21. Positions the components into five regions: east, west, north, south, center,
    (a) BorderLayout (b) CardLayout
    (c) GridLayout (d) FlowLayout
22. Arranges the components as a deck of cards such that only one component is visible at a time,
    (a) BorderLayout (b) CardLayout
    (c) GridLayout (d) FlowLayo
23. Which of these events generated when a button is pressed,
    (a) ActionEvent (b) WindowEvent
    (c) ItemEvent (d) KeyEvent
24. Which is an API that contains large number of classes and methods to create and manage GUI applications?
    (a) AWT (b) Window
    (c) Panel (d) Frame
25. Which class is at the top of AWT hierarchy?
    (a) Container (b) Component
    (c) Window (d) Frame

26. Which is a component that contains other components like button, textfield, label, etc.?
    (a) Window (b) Component
    (c) Container (d) Button
27. Which class creates a top level window?
    (a) Frame (b) Panel
    (c) Button (d) Window
28. Which is used for placing text in a container?
    (a) Button (b) Frame
    (c) Label (d) Panel
29. Which is used for displaying single line text?
    (a) Label (b) TextField
    (c) Frame (d) Window
30. Which is used for creating a drop-down menu of choices?
    (a) Choice (b) Chech box
    (c) TextArea (d) TextField
31. MVC architecture contains,
    (a) Model represents component's data.
    (b) View represents visual representation of the component's data.
    (c) Controller takes the input from the user on the view and reflects the changes in Component's data.
    (d) All the Mentioned
32. Which refers to the arrangement of components within the container?
    (a) Layouts (b) Borders
    (c) Menu bars (d) None of mentioned
33. The task of laying out the controls is done automatically by the,
    (a) AWT Manager (b) Swing Manager
    (c) Layout Manager (d) All of mentioned
34. Which class is the base class for all Swing components except top-level containers?
    (a) Component (b) Container
    (c) JComponent (d) All of the mentioned
35. Which is is a platform-independent, "model–view–controller" GUI framework for Java?
    (a) AWT (b) Swing
    (c) Botn (a) and (b) (d) None of mentioned
36. Which is an interface which implements the classes of the layout manager?
    (a) SwingLayoutMananger (b) AWTLayoutMananger
    (c) LayoutMananger (d) All of mentioned
37. Which is the default layout that ayouts the components in a directional flow?
    (a) BorderLayout (b) CardLayout
    (c) FlowLayout (d) GridLayout

38. Which AWT layout arranges the components in grid cells as rows and columns?
    (a) Border (b) Card
    (c) Flow (d) Grid
39. Which is the top level class for all menu related controls?
    (a) MenuComponent (b) Menut
    (c) MenuContainer (d) All of mentioned
40. Change in the state of an object is known as,
    (a) source (b) listener
    (c) event (d) None of mentioned
41. Which is the mechanism that controls the event and decides what should happen if an event occurs?
    (a) Event handling (b) Source handling
    (c) Listener handling (d) All of mentioned
42. Which model defines the standard mechanism to generate and handle the events?
    (a) MVC (b) Delegation Event
    (c) Both (a) and (b) (d) None of mentioned
43. Which class is the root class from which all events in AWT?
    (a) EventObject (b) AWTObject
    (c) SwingOBject (d) None of mentioned
44. Which represent the interfaces responsible to handle events?
    (a) Event Listener (b) Event Source
    (c) Event Handler (d) None of mentioned
45. Which are abstract classes for receiving various events?
    (a) Sources (b) Listeners
    (c) Adapters (d) All of mentioned
46. Which interface is used for receiving the mouse events?
    (a) TextListener (b) KeyListener
    (c) ComponentListener (d) MouseListener
47. Which abstract adapter class for receiving focus events?
    (a) KeyAdapter (b) MouseAdapter
    (c) FocusAdapter (d) WindowAdapter
48. Which class has no name?
    (a) inner (b) wrapper
    (c) string (d) anonymous
49. In swing which are the subclass of MenuComponent class.
    (a) Menu (b) MenuItem
    (c) Both (a) and (b) (d) None of mentioned
50. Which AWT control represents a top-level window with a title and a border used to take some form of input from the user?
    (a) Scrollbar (b) Dialog
    (c) Button (d) Label

51. Which is Swing graphical component that can be in either an on (true) or off (false) state?
    (a) JComboBox (b) JCheckBox
    (c) JButton (d) JRadioButton
52. Which is AWT control component that has a label and generates an event when pressed?
    (a) List (b) Check box
    (c) Button (d) Choice
53. Which control represents a dialog window from which the user can select a file?
    (a) MessageDialog (b) FileDialog
    (c) Dialog (d) All of mentioned
54. Which Swing component presents the user with a scrolling list of text items?
    (a) JTextField (b) JList
    (c) JComboBox (d) JCheckBox
55. Which is is a pull-down menu component which is displayed from the menu bar?
    (a) JMenu (b) JMenuItem
    (c) JMenuBar (d) JPopupMenu
56. Which menu represents a menu which can be dynamically popped up at a specified position within a component?
    (a) MenuBar (b) PopupMenu
    (c) MenuItem (d) Menu
57. Which is a top-level window in AWT with a title and a border?
    (a) Panel (b) Window
    (c) Frame (d) All of mentioned
58. Which Swing control provides a pane of controls designed to allow a user to manipulate and select a color?
    (a) JFileChooser (b) JColorChooser
    (c) JDialogChooser (d) None of mentioned

Answers

1. (c)	2. (b)	3. (c)	4. (a)	5. (c)	6. (d)	7. (a)	8. (d)	9. (b)	10. (c)
11. (b)	12. (c)	13. (b)	14. (d)	15. (b)	16. (d)	17. (b)	18. (d)	19. (d)	20. (a)
21. (a)	22. (c)	23. (a)	24. (a)	25. (b)	26. (c)	27. (d)	28. (c)	29. (b)	30. (a)
31. (d)	32. (a)	33. (c)	34. (c)	35. (b)	36. (c)	37. (c)	38. (d)	39. (a)	40. (c)
41. (a)	42. (b)	43. (a)	44. (a)	45. (c)	46. (d)	47. (c)	48. (d)	49. (c)	50. (b)
51. (d)	52. (c)	53. (b)	54. (b)	55. (a)	56. (b)	57. (c)	58. (b)		

Q.II Fill in the Blanks:

1. Fill in the blanks so that this program displays a Frame:

```
import java.awt.*;
public class microGUI
{
    public static void main ( String[] args )
    {
        Frame frm = new frame();
        frm._______( 350, 300 );
        frm._______( true );
    }
}
```

2. The classes and interfaces defined in AWT are contained within the _____ package.
3. In Java an event is an _____ which specifies the change of state in the source.
4. Java packages such as _____ support the Event handling mechanism.
5. Panel is used for _____ components.
6. _____ is a superclass of TextField and TextArea classes that is used to create single-line or multiline textfields respectively
7. A label is a simple control which is used to display _____ on the window.
8. _____ generates action events when an item is double-clicked, generate action events when an item is selected or deselected.
9. _____ arranges the componemnts into grid
10. _____ classes are derived from the Component class.
11. A label is a simple AWT control which is used to display _____ (non-editable) on the window.
12. _____ method we can set or change the text in a Label.
13. The classes and interfaces defined in AWT are contained within the _____ package.
14. A class that have no name is known as _____ class.
15. Swing components that does not rely on Native GUI are referred to as _____ component.
16. Swing is a component based _____, whose components are all ultimately derived from the javax.swing.JComponent class.
17. The base class for all Swing components except top-level containers is _____.
18. _____ in AWT does not have borders and menubar.
19. _____ define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).
20. The layout _____ is associated with every Container object.
21. A _____ object is a top-level window with no borders and no menubar.
22. AWT layout is defined as the order of placement of the _____ in a container and this arrangement can be done by LayoutManager.
23. The class Component is the _____ base class for the non menu user-interface controls of AWT.
24. Java provides various layout managers to _____ the controls. Properties like size, shape, and arrangement varies from one layout manager to the other.
25. _____ is a GUI widget toolkit for Java.
26. The interface between the _____ and application program is known as an user-interface.
27. Java _____ is an API to develop GUI or window based applications in java.

28. The screen area on the desktop displaying the data and performing the operations specified is called as a _____.
29. AWT _____ are nothing but AWT components that allows the user to communicate or interact with the user in different ways.
30. An object change from one state to another state when used in an application is known as _____.
31. The layout manager automatically positions all the components within the _____.
32. The standard process of giving directions when an event occurs is called as _____.
33. _____ in AWT is the user-defined text, which cannot be modified by the end-user.
34. _____ in AWT contains a Label and creates an event when clicked on it.
35. ScrollBar in AWT allows the end-user to navigate up and down when there are _____ options in a field.
36. _____ control in AWT decides the state of an event by true or false.
37. _____ control in AWT allows a single selection and List allows multiple selection of choices.
38. A _____JFrame is a top-level window with a title and a border.
39. _____ is considered as a window without having a menu bar.
40. FlowLayout is the _____ layout manager.
41. Swing API architecture based _____ (Model-View-Controller) architecture.
42. A new LayoutManager can be set by using the method _____.
43. A JComponent is a _____ class for all Swing UI components.
44. _____ arranges the components from top to bottom and left to right with the preferred sizes on the screen.
45. _____ arranges the components as per the given borders (North, South, East, West) and Center.
46. GridBagLayout is a _____ AWT Layout.
47. The _____ object treats each component in the container as a card (outputs only a single card at a time).
48. Textfield control in AWT is used to accept a _____ input from the end-user.

Answers

1. setSize, setVisible	2. java.awt.*	3. object	4. java.awt.event
5. Grouping	6. TextComponent	7. Text	8. Check box
9. GridLayout	10. Container	11. Text	12. setText()
13. java.awt.*;	14. anonymous	15. Ligthweight	16. framework
17. JComponent	18. Window	19. Layouts	20. manager
21. JWindow	22. components	23. abstract	24. position
25. Swing	26. user	27. AWT	28. Window
29. controls	30. event	31. container	32. event handling
33. Label	34. Button	35. multiple	36. Checkbox
37. Choice	38. Dialog	39. Dialog	40. default
41. MVC	42. setLayout()	43. base	44. FlowLayout
45. BorderLayout	46. resizable	47. CardLayout	48. single-line

Q.III State True or False:

1. Implement the Listener interface and overrides its methods is required to perform in event handling.
2. The ActionListener interface is not used for handling action events.
3. The Java Foundation Classes (JFC) is a set of GUI components which simplify the development of desktop applications.
4. Swing is not a part of JFC (Java Foundation Classes) that is used to create GUI application
5. AWT has more powerful components like tables, lists, scroll panes, color chooser, tabbed pane etc.
6. The ActionListener interface is not used for handling action events?
7. AWT is used for GUI programming in java?
8. The getLabel() method used to retrieve the label of a button.
9. These two ways are used to create a Frame: By creating the object of Frame class (association) By extending Frame class (inheritance)
10. The getWindowObject() methods is used to obtain the object that generated a WindowEvent.
11. The getPoint() method can not be used to obtain the coordinates of a mouse
12. Swing not provides platform-independent and lightweight components.
13. The mouseDragged() methods is defined in MouseMotionAdapter class
14. AWT more powerful components like tables, lists, scroll panes, color chooser, tabbed pane etc.?
15. Implement the Listener interface and overrides its methods is required to perform in event handling?
16. The class Container is the super class for the containers of AWT.
17. Swing is part of Oracle's Java Foundation Classes (JFC) - an API for providing a GUI for Java programs.
18. The AWT is Java's original platform dependent windowing, graphics, and user-interface widget toolkit, preceding Swing.
19. The AWT is part of the Java Foundation Classes (JFC) - the standard API for providing a graphical user interface (GUI) for a Java program.
20. Dialog accepts the user input and does not have menu bar.
21. TextArea Control in AWT supports single-line input from the end-user.
22. The Delegation Event Model describes how to generate and control the events.
23. A Swing component is an object having a graphical representation that can be displayed on the screen and that can interact with the user.
24. JComboBox is used to select single element from multiple, here user can select an element from the drop down list.

25. Swing Dialogues is used to provide the communication between the user and the system and most of the dialogues are used to display warning messages and alert messages.
26. JMenuItem is used to display menubar on the window or frame.
27. JFileChooser represents a dialog window from which the user can select file.
28 AWT scrolling is done vertically or horizontally the ScrollBar scrolls vertically and horizontally from its default value by setting the minimum and maximum values.
29. A JFileChooser control represents a dialog window from which the user can select a color.
30. Events are generated as result of user interaction with the graphical user interface components such as clicking on a button, moving the mouse and so on.
31. The source is an object on which event occurs.
32. Listener is also known as event handler which are responsible for generating response to an event.
33. Java uses the Delegation Event Model to handle the events.
34. Event handling mechanism have the code which is known as event handler that is executed when an event occurs.
35. A JButton object is a component for placing text in a container.
36. The ActionEvent is generated when button is clicked or the item of a list is double clicked.
37. A JTextField object is a text component that allows editing of a single line of text.
38. FocusListener is used for receiving the focus events.
39. WindowAdapter is an abstract adapter class for receiving window events.
40. The Menu object is a pull-down menu component which is displayed from the menu bar.
41. Swing based GUI Application look and feel can be changed at run-time.
42. A JList component presents the user with a scrolling list of text items.

Answers

1. (T)	2. (F)	3. (T)	4. (F)	5. (T)	6. (F)	7. (T)	8. (T)	9. (T)	10. (F)
11. (F)	12. (F)	13. (T)	14. (T)	15. (T)	16. (T)	17. (T)	18. (T)	19. (T)	20. (T)
21. (F)	22. (T)	23. (T)	24. (T)	25. (T)	26. (F)	27. (T)	28. (T)	29. (F)	30. (T)
31. (T)	32. (T)	33. (T)	34. (T)	35. (F)	36. (T)	37. (T)	38. (T)	39. (T)	40. (T)
41. (T)	42. (T)								

Q.IV Answer the following Questions:

(A) Short Answer Questions:

1. What is AWT?
2. What is Swing?
3. Define layout and layout manager.
4. What is MVC architecture?
5. What is frame?
6. What is JPopupMenu?
7. Define containers and components in Swing.
8. Name the class which is a base class for all Swing components.
9. What is a panel?
10. What is the purpose of JMenu?
11. Define event?
12. What is a container?
13. What is the purpose of FlowLayout?
14. Define event source and listener?
15. What is menu?
16. Give the purpose of JLabel.
17. What is the use of JButton?
18. What is anonymous inner class?
19. What is Dialog?
20. Give function of JColorChooser?
21. What is the purpose of JList?

(B) Long Answer Questions:

1. With the help of diagram describe AWT hierarchy.
2. Differentiate between AWT and Swing.
3. What is layout? List types of layout mangers in AWT.
4. With the help of example explain following layout managers:
   (i) FlowLayout (ii) BorderLayout
   (iii) CardLayout (iv) GridLayout.
5. Explain MVC architecture digrammatically.
6. Define event, source and listeners. Also explain how to handle events with Delegation Event model.
7. Which Swing classes are used to create menu?
8. Write a short note on: Adapters.
9. Which are the various sources of event in AWT and Swing?
10. Explain in brief the fundamental idea behind MVC architecture?
11. Which are the features of Swing?
12. How to create menus in AWT? Epain popup menus with example.
13. With the help of example describe following AWT components:
    (i) Button (ii) Checkbox
    (iii) TextArea (iv) ScrollBar

14. With the help of example describe following Swing components:
    (i) JButton (ii) JFileChooser
    (iii) JTextField (iv) JRadioButton
    (v) JComboBox (vi) JColorChooser.
15. Describe anonymous inner class in detail.

UNIVERSITY QUESTIONS AND ANSWERS

April 2016

1. List any four layout managers. **[1 M]**
Ans. Refer to Section 5.3.
2. Define AWT and swing. **[1 M]**
Ans. Refer to Section 5.1.3.
3. Write a Java program to set background colour of dialog box using JColorChooser. **[5 M]**
Ans. Refer to Section 5.6.4.
4. Define anonymous class. **[2 M]**
Ans. Refer to Section 5.9.
5. What is an event? What is an event handling? **[2 M]**
Ans. Refer to Section 5.7.

October 2016

1. List any four layout managers in Java. **[1 M]**
Ans. Refer to Section 5.3.
2. List constructors of JTextArea class (any two). **[1 M]**
Ans. Refer to Section 5.5.10.
3. Explain JOptionPane class with their static methods. **[5 M]**
Ans. Refer to Section 5.6.2.
4. Design a screen in Java using AWT to handle the mouse events and display the positions of mouse click in textbox. **[1 M]**
Ans. Refer to Section 5.7.

April 2017

1. What is an adapter class? **[1 M]**
Ans. Refer to Section 5.8.
2. List any four methods of JButton class. **[1 M]**
Ans. Refer to Section 5.5.7.
3. Explain JComboBox class with their constructor and methods. **[5 M]**
Ans. Refer to Section 5.5.13.
4. Write a Java program using AWT to print "Welcome to T.Y. B.Sc. Computer Science" with Times New Roman font face and size is 16 in Red Color. When we click on a button text color should change to Blue. **[5 M]**
Ans. Refer to Section 5.4.

October 2017

1. What is the use of adapter class? **[1 M]**

Ans. Refer to Section 5.8.

2. Give two types of Dialogs. **[1 M]**

Ans. Refer to Section 5.6.

3. Explain in brief the event handling mechanism in Java with the help of suitable example. **[5 M]**

Ans. Refer to Section 5.7.

4. What are the features of Swing? **[2 M]**

Ans. Refer to Section 5.5.

April 2018

1. Write a syntax of JFileChooser class. **[1 M]**

Ans. Refer to Section 5.6.3.

2. Why Swing objects are called as light weight components? **[1 M]**

Ans. Refer to Section 5.5 Point (1).

3. Write a program in Java to create a screen which contains three checkboxes (.net, php, java) and displays the selected items in a textbox. **[5 M]**

Ans. Refer to Section 5.4.

October 2018

1. State any four methods of WindowListener. **[1 M]**

Ans. Refer to Section 5.7.2.

2. How is menu created in Java ? Explain with suitable example. **[5 M]**

Ans. Refer to Section 5.4 Point (10).

3. What is the use of Checkboxes and RadioButtons ? Explain with suitable example. **[4 M]**

Ans. Refer to Section 5.4.

4. Explain layout managers used in AWT. **[4 M]**

Ans. Refer to Section 5.3.

April 2019

1. What is anonymous classes? **[1 M]**

Ans. Refer to Section 5.9.

2. Write a Java program which will create a frame if we try to close it. It should change it's color Red and it display closing message on the screen.(Use swing) **[5 M]**

Ans. Refer to Section 5.4.

3. What are the different types of dialogs in Java? Write any one in detail. **[3 M]**

Ans. Refer to Section 5.6.

4. Which Swing classes are used to create menu? **[2 M]**

Ans. Refer to Section 5.5.14.

■■■