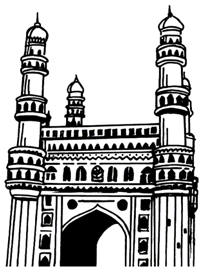
Rahul's ✔

Topper's Voice

AS PER CBCS SYLLABUS



Latest 2022 Edition

B.Sc.

III Year V Sem

PROGRAMMING IN JAVA

Paper - V

- **Study Manual**
- **Lab Practicals**
- **FAQ's & Important Questions**
- **Choose the Correct Answer**
- **Fill in the blanks**
- **One Mark Answers**
- **Solved Previous Question Papers**
- **Solved Model Papers**

Price **`. 189/-**

- by -

WELL EXPERIENCED LECTURER

Rahul Publications™

Hyderabad. Ph : 66550071, 9391018098

B.Sc.

III Year V Sem

PROGRAMMING IN JAVA

Paper - V

Price ` . 189-00

Sole Distributors : **☎ : 665500071, Cell : 9391018098**

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.
Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

CONTENTS

PROGRAMMING IN JAVA

SYLLABUS

UNIT - I

Introduction: Java Essentials, JVM, Java Features, Creation and Execution of Programs, Data Types, Structure of Java Program, Type Casting, Conditional Statements, Loops, Classes, Objects, Class Declaration, Creating Objects.

UNIT - II

Method Declaration and Invocation, Method Overloading, Constructors – Parameterized Constructors, Constructor Overloading, Cleaning-up unused Objects. Class Variables &Method-static Keyword, this Keyword, One-Dimensional Arrays, Two-Dimensional Arrays, Command-Line Arguments, Inner Class.

Inheritance: Introduction, Types of Inheritance, extends Keyword, Examples, Method Overriding, super, final Keyword, Abstract classes, Interfaces, Abstract Classes Verses Interfaces.

Packages: Creating and Using Packages, Access Protection, Wrapper Classes, String Class, StringBuffer Class.

UNIT - III

Exception: Introduction, Types, Exception Handling Techniques, User-Defined Exception.

Multithreading: Introduction, Main Thread and Creation of New Threads –By Inheriting the Thread Class or Implementing the Runnable Interface, Thread Lifecycle, Thread Priority and Synchronization.

Input/Output: Introduction, java.io Package, File Streams, FileInputStream Class, FileOutputStream Class, Scanner Class, BufferedInputStream Class, BufferedOutputStream Class, RandomAccessFile Class.

UNIT - IV

Applets: Introduction, Example, Life Cycle, Applet Class, Common Methods Used in Displaying the Output (Graphics Class).

Event Handling: Introduction, Types of Events, Example.

AWT: Introduction, Components, Containers, Button, Label, Checkbox, Radio Buttons, Container Class, Layouts.

Swings: Introduction, Differences between Swing and AWT, JFrame, JApplet, JPanel, Components in Swings, Layout Managers, JTable.

Contents

Frequently Asked & Important Questions

UNIT - I

1. Write about different features of java.

Ans : **(Dec.-19)**

Refer Unit-I, Q.No. 5

---

2. Describe the Structure of a Java Program.

Ans : **(Dec.-19)**

Refer Unit-I, Q.No. 8

---

3. Explain about the type casting and type conversion in java.

Ans : **(July-21)**

Refer Unit-I, Q.No. 9

---

4. What are the various types of Control Statements in Java with a suitable examples.

Ans : **(Dec.-19)**

Refer Unit-I, Q.No. 10

---

5. Explain the Conditional statements in java with suitable examples.

Ans : **(Dec.-19, June-19(MGU)**

Refer Unit-I, Q.No. 11

---

6. What are the different looping statements in java? Explain.

Ans : **(Dec.-19, Dec.-18(MGU), Dec.-18(KU)**

Refer Unit-I, Q.No. 12

---

7. Explain the process of creating objects in java?

Ans : **(Dec.-19)**

Refer Unit-I, Q.No. 15

UNIT - II

1. Compare and contrast method overloading and method overriding.

Ans : **(July-21, Dec.-18)**

Refer Unit-II, Q.No. 5

---

2. Define Constructors. Explain different types of constructors in java.

Ans : **(Dec.-18, KU)**

Refer Unit-II, Q.No. 6

3. Explain about Command-Line Arguments. How they are useful.

Ans : **(June-19, MGU)**

Refer Unit-II, Q.No. 15

4. What are the different types of Inheritance?

Ans : **(July-21, July-19, Dec.-18-MGU)**

Refer Unit-II, Q.No. 19

5. Differentiate the use of Abstract class and Interface?

Ans : **(June-19-MGU, Dec.-18-KU)**

Refer Unit-II, Q.No. 32

6. What is a package in java?

Ans : **(July-19)**

Refer Unit-II, Q.No. 33

7. What are wrapper class? What is its use in JAVA?

Ans : **(Dec.-19, Dec.-18)**

Refer Unit-II, Q.No. 36

8. What is a String Buffer class?

Ans : **(Dec.-18)**

Refer Unit-II, Q.No. 39

UNIT - III

1. Explain the handling of exception in Java.

Ans : **(June-19-(MGU), Dec.-18(MGU)**

Refer Unit-III, Q.No. 3

2. Explain the Multithreading in Java.

Ans : **(Dec.-19)**

Refer Unit-III, Q.No. 9

3. Define Threads. List different ways of creating threads. Explain with examples.

Ans : **(July-21, June-19(MGU), Dec.-18(MGU)**

Refer Unit-III, Q.No. 11

4. What is java.io Package? Explain the different ways of input output methods.

Ans : **(July-21)**

Refer Unit-III, Q.No. 15

5. Explain in detail about FileInputStream Class with an example.

Ans : **(Dec.-19, July-19, Dec.-18(KU)**

Refer Unit-III, Q.No. 17

6. Explain in detail about FileOutputStream Class with an example.

Ans : **(Dec.-19, July-19, Dec.-18(KU)**

Refer Unit-III, Q.No. 18

7. What is a RandomAccessFile Class? Write a program for reading / writing using random access file.

Ans : **(July-19)**

Refer Unit-III, Q.No. 23

UNIT - IV

1. Explain applet life cycle.

Ans : **(July-21, Dec.-18, Dec.-18(KU)**

Refer Unit-IV, Q.No. 2

2. Explain various steps involved in even handling.

Ans : **(June-19(MGU)**

Refer Unit-IV, Q.No. 8

3. What are the Components of AWT?

Ans : **(Dec.-19)**

Refer Unit-IV, Q.No. 13

4. Discuss about various containers of AWT.

Ans : **(Dec.-18(MGU)**

Refer Unit-IV, Q.No. 14

5. Explain in detail about Checkbox control with example?

Ans : **(Dec.-19, Dec.-18(MGU)**

Refer Unit-IV, Q.No. 17

6. Explain types of layout managers with an example.

Ans : **(June-19(MGU), Dec.-18, Dec.-18(MGU)**

Refer Unit-IV, Q.No. 20

7. Compare and contrast AWT and Swing.

Ans : **(Dec.-19)**

Refer Unit-IV, Q.No. 22

8. Explain about JPanel by giving an example.

Ans : **(July-19)**

Refer Unit-IV, Q.No. 25

9. Discuss about Layout Managers in swing.

Ans : **(Imp.)**

Refer Unit-IV, Q.No. 27

UNIT I

INTRODUCTION:
Java Essentials, JVM, Java Features, Creation and Execution of Programs, Data Types, Structure of Java Program, Type Casting, Conditional Statements, Loops, Classes, Objects, Class Declaration, Creating Objects.

1.1 Introduction to Java

Q1. What is Java?

(OR)

Define Java.

Ans :

Introduction

Java is a high-level programming language that follows object-oriented programming principle. The main motto of Java programming language is "write once run anywhere." The syntax and semantics of Java are very much similar to C and C++ programming language.

Java is considered a programming language and a platform. As a programming language, it is a general-purpose, object-oriented high-level programming language that has its own syntax and style. As a platform, it provides an environment in which Java applications run.

Java has four different platforms:-

(i) Java Standard Edition (Java SE): In general, most of the programmers use Java SE to do programming in Java. It provides the core functionality of the Java language. APIs that are used for database access, GUI development, networking, etc. are present in this platform.

(ii) Java Enterprise Edition (Java EE): This platform is used to develop and run reliable, secure and large-scale network applications. It is built on the top of Java SE.

(iii) Java Micro Edition (Java ME): This platform is suitable for small devices like smartphones. It is a subset of Java SE and provides API applicable for application development on small appliances. It has small footprint virtual machine that enables Java applications to run smoothly on less memory constrained devices.

(iv) JavaFX: It is used to create rich internet applications. It helps programmers to create applications with modern look-and-feel and utilize hardware-accelerated graphics.

1.1.1 History of Java

Q2. Explain the Evolution of Java.

Ans

The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts from Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions etc. But, it was suited for internet programming. Later, Java technology was incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describe the history of java.

1. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
2. Originally designed for small, embedded systems in electronic appliances like set-top boxes.
3. Firstly, it was called "Greentalk" by James Gosling and file extension was .gt.
4. After that, it was called Oak and was developed as a part of the Green project.
5. Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.
6. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

Java Version History

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)
11. Java SE 9 (21st Sep, 2017).

1.1.2 Java Essentials

Q3. Write about the Java Essentials.

Ans :

The essentials of Java programming language are as follows,

(i) High-level Language

Java is known to be high-level language that supports unique features. It is quite similar to C and C++.

(ii) Bytecode of Java

A bytecode is nothing but intermediate code that the compiler generates. This code is executed by JVM.

(iii) Java Virtual Machine (JVM)

A bytecode consists of optimized set of instructions which are usually executed by Java runtime system on a special machine called Java Virtual Machine. A JVM is a bytecode interpreter. It performs line by line execution of bytecode.

Java has a neutral-architecture since it allows programs to run on different platforms. The java compiler generates the bytecode that are neutral-architecture in order to achieve the capabilities of cross-architecture. These instructions can be interpreted easily on any platform translated into machine code. Java Runtime Environment (JRE) contains JVM, class libraries and different supporting files.

There are many versions of JDK and differ in case of platforms like Windows and Linux. JVM belongs to JRE, that differs incase of OS and computer architecture. Java cannot be run if JVM is not available for the particular environment.

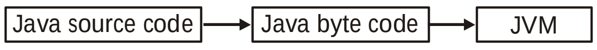


Fig.: java Runtime Environment

Java source code contains the simple code that is easy to develop and understand. The bytecode can be run on any platform, therefore, it is Write Once Run Anywhere (WORA). JVM allows to convert the bytecode into machine code. Java Development Kit (JDK) contains the tools like javac java etc. They are of various versions used on different platforms.

Java Virtual Machine (JVM) is a Java interpreter that executes bytecode. The interpreter reads and translates the entire code into machine code. This translated code is then executed by the machine. This machine perform all the functions of a computer. The type of Java interpreter to be ported on the machine must be compatible with the machine and its operating system. The virtual machine code is different for different computers and acts as an intermediate between the virtual machine and the actual machine SUN provides virtual machine implementations for Microsoft Windows 95, 98, NT or Solaris operating system. The process of translation of a Java program into bytecode simplifies the execution of program in various platforms. Due to this, Java is said to be platform independent. Execution of a Java program by JVM makes the program more secure and portable. JVM also solves the issues related with web-based programs.

However, sometimes interpreter becomes inefficient due to the hotspots and due to the consumption of large amount of time for interpreting. A compilation technology called JIT (Just-in-time) has been newly introduced for virtual machine implementations. This technology provides JIT machine executable codes. Another technology called Hot spot technology also improved the performance of JVM.

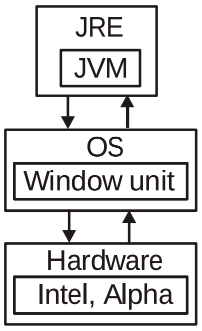


Fig.: JVM Handles Transiations

1.1.3 JVM

Q4. Explain the architecture of JVM.

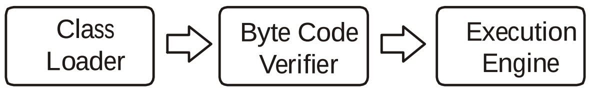
(OR)

Explain the architecture of JVM with a neat sketch.

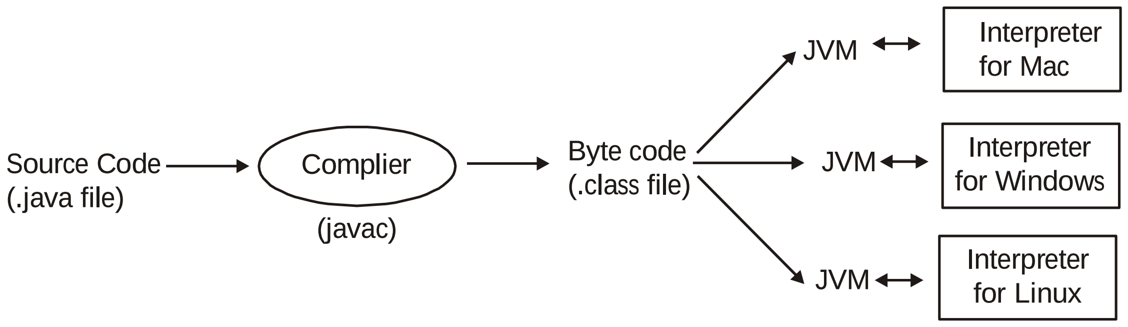
Ans :

Java is a high level programming language. A program written in high level language cannot be run on any machine directly. First, it needs to be translated into that particular machine language. The javac compiler does this thing, it takes java program (.java file containing source code) and translates it into machine code (referred as byte code or .class file).

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the machine language for JVM is byte code. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. JVM is the one that makes java platform independent.



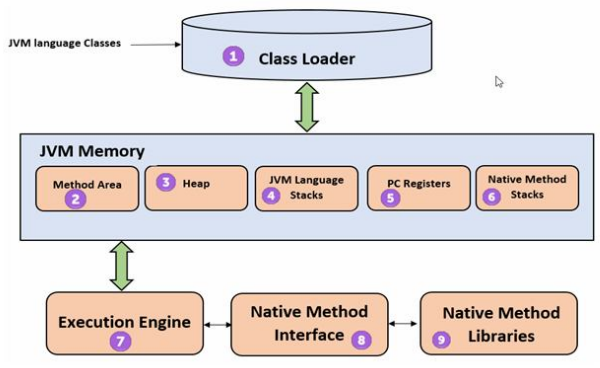
So, now we understood that the primary function of JVM is to execute the byte code produced by compiler. Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems. Which means that the byte code generated on Windows can be run on Mac OS and vice versa. That is why we call java as platform independent language. The same thing can be seen in the diagram below:



The Java Virtual machine (JVM) is the virtual machine that runs on actual machine (your computer) and executes Java byte code. The JVM doesn't understand Java source code, that's why we need to have javac compiler that compiles *.java files to obtain *.class files that contain the byte codes understood by the JVM. JVM makes java portable (write once, run anywhere). Each operating system has different JVM, however the output they produce after execution of byte code is same across all operating systems.

JVM Architecture

1. **Class Loader:** The class loader reads the .class file and save the byte code in the method area.
2. **Method Area**: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.
3. **Heap**: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.
4. **JVM language Stacks:** Java language Stacks store local variables, and it's partial results. Each thread has its own JVM stack, created simultaneously as the thread is created. A new frame is created whenever a method is invoked, and it is deleted when method invocation process is complete.
   - **Stack**: Stack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.
5. **PC Registers**: This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.
6. **Native Method stack:** A native method can access the runtime data areas of the virtual machine.
7. **Native Method interface**: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.
8. **Garbage collection**: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.



1.1.4 Java Features

Q5. Explain the features of Java.

(OR)

Write about different features of java.

Ans : **(Dec.-19)**

As the languages like Objective C, C++ fulfills the above four characteristics yet they are not fully object oriented languages because they are structured as well as object oriented languages.

In java everything is an Object. Java can be easily extended since it is based on the Object model

- **Secure:** Java is Secure Language because of its many features it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption. Java does not support pointer explicitly for the memory. All Program Run under the sandbox.
- **Robust:** Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.

  Memory management has been simplified java in two ways. First Java does not support direct pointer manipulation or arithmetic. This make it possible for a java program to overwrite memory or corrupt data.

  Second, Java uses runtime garbage collection instead of instead of freeing of memory. In languages like C++, it Is necessary to delete or free memory once the program has finished with it.
- **Platform-independent:** Java Language is platform-independent due to its hardware and software environment. Java code can be run on multiple platforms e.g. windows, Linux, sun Solaris, Mac/Os etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).
- **Architecture neutral:** It is not easy to write an application that can be used on Windows, UNIX and a Macintosh. And its getting more complicated with the move of windows to nonIntel CPU architectures.

  Java takes a different approach. Because the Java compiler creates byte code instructions that are subsequently interpreted by the java interpreter, architecture neutrality is achieved in the implementation of the java interpreter for each new architecture.
- **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.

  In java, all primitive types (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run. This is in direct contrast to languages like C and C++ that leave the sized of primitive types up to the compiler and developer.

  Additionally, Java is portable because the compiler itself is written in Java.
- **Dynamic:** Because it 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.

  At runtime, the java interpreter performs name resolution while linking in the necessary classes. The Java interpreter is also responsible for determining the placement of object in memory. These two features of the Java interpreter solve the problem of changing the definition of a class used by other classes.
- **Interpreted:** We all know that Java is an interpreted language as well. With an interpreted language such as Java, programs run directly from the source code.

  The interpreter program reads the source code and translates it on the fly into computations. Thus, Java as an interpreted language depends on an interpreter program.

The versatility of being platform independent makes Java to outshine from other languages. The source code to be written and distributed is platform independent.

Another advantage of Java as an interpreted language is its error debugging quality. Due to this any error occurring in the program gets traced. This is how it is different to work with Java.

- **High performance: For** all but the simplest or most infrequently used applications, performance is always a consideration for most applications, including graphics-intensive ones such as are commonly found on the world wide web, the performance of java is more than adequate.
- **Multithreaded: Writing** a computer program that only does a single thing at a time is an artificial constraint that we've lived with in most programming languages. With java, we no longer have to live with this limitation. Support for multiple, synchronized threads is built directly into the Java language and runtime environment.

  Synchronized threads are extremely useful in creating distributed, network-aware applications. Such as application may be communicating with a remote server in one thread while interacting with a user in a different thread.
- **Distributed: Java** facilitates the building of distributed application by a collection of classes for use in networked applications. By using java?s URL (Uniform Resource Locator) class, an application can easily access a remote server. Classes also are provided for establishing socket-level connections.

1.1.5 Creation and Execution of Programs

Q6. Explain the process of Creating and Executing a Jaga Program.

Ans :

A Java program can be successfully created and i in three steps. They are as follows,

1. Writing the source code
2. Saving the code
3. Compiling the code.

1. Writing the Source Code

The java code can be written in text editor like notepad.

2. Saving the Code

The program can be saved using class name followed by 'java' extension i.e., Simple.Java.

3. Compiling the Code

To compile the program, run the Java compiler javac with the name of source file on the command line

as,

```
javac Simple.java
```

If the syntax is correct and no errors have been detected then Java compiler creates a file called Simple, java containing byte code of the program.

Process of Executing a Java Program

Stand alone applications are run using interpreter. For this, type the following commands at command prompt,

```
java Simple
```

After this, the interpreter searches the main method in the program and begins the execution. After performing the execution it displays the desired result as output.

1.2 Data Types

Q7. Discuss in brief about various data types.

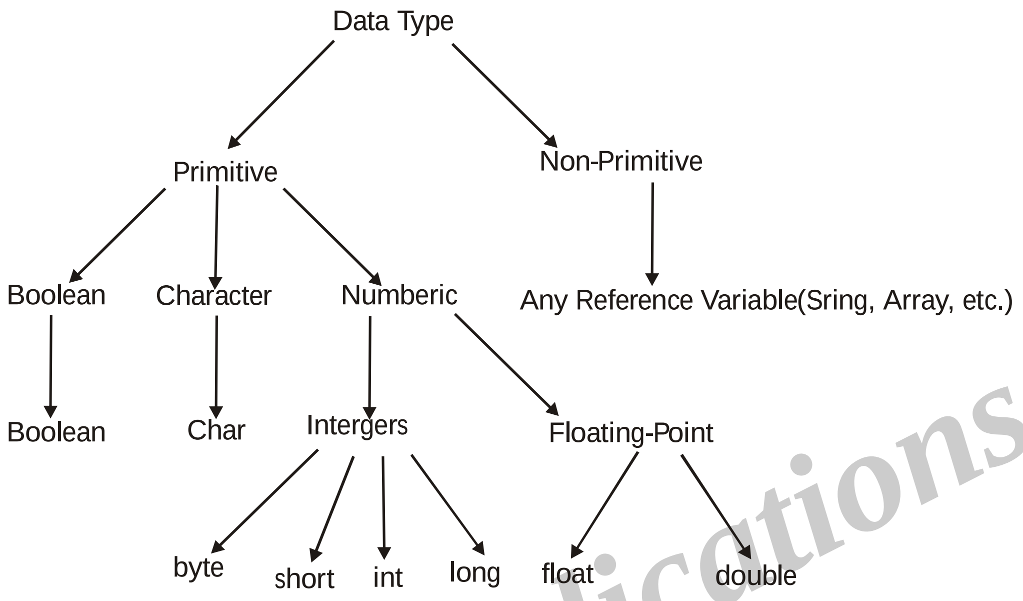
Ans :

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

There are two data types available in Java:

- Primitive Data Types
- Reference/Object Data Types



I. Primitive Data Types

There are eight primitive data types supported by Java. Primitive data types are predefined by the language and named by a key word. Let us now look into detail about the eight primitive data types.

(i) byte

- Byte data type is a 8-bit signed two's complement integer.
- Minimum value is –128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 –1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an int.
- Example : byte a = 100 , byte b = -50

(ii) short

- Short data type is a 16-bit signed two's complement integer.
- Minimum value is -32,768 (-2^{15})
- Maximum value is 32,767(inclusive) (2^{15} -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an int
- Default value is 0.
- Example : short s= 10000 , short r = -20000

(iii) int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648.(-2^{31})
- Maximum value is 2,147,483,647 (inclusive). (2^{31} -1)
- Int is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0.
- **Example:** int a = 100000, int b = -200000

(iv) long

- Long data type is a 64-bit signed two's complement integer.
- Minimum value is -9,223, 372, 036, 854, 775, 808. (-2^{63})
- Maximum value is 9, 223, 372, 036, 854, 775, 807 (inclusive). (2^{63} -1)
- This type is used when a wider range than int is needed.
- Default value is 0L.
- Example : long a = 100000L, int b = -200000L

(v) float

- Float data type is a single-precision 32-bit IEEE 754 floating point.
- Float is mainly used to save memory in large arrays of floating point numbers.
- Default value is 0.0f.
- Float data type is never used for precise values such as currency.
- Example : float f1 = 234.5f

(vi) double

- double data type is a double-precision 64-bit IEEE 754 floating point.
- This data type is generally used as the default data type for decimal values. Generally the default choice.
- Double data type should never be used for precise values such as currency.
- Default value is 0.0d.
- Example : double d1 = 123.4

(vii) boolean

- boolean data type represents one bit of information.
- There are only two possible values : true and false.
- This data type is used for simple flags that track true/false conditions.
- Default value is false.
- Example : boolean one = true

(viii) char

- char data type is a single 16-bit Unicode character.
- Minimum value is '\u0000' (or 0).
- Maximum value is '\uffff' (or 65,535 inclusive).
- Char data type is used to store any character.
- Example: char letterA = 'A'

II. Reference Data Types

- Reference variables are created using defined constructors of the classes. They are used to access objects. These variables are declared to be of a specific type that cannot be changed. For example, Employee, Puppy etc.
- Class objects, and various type of array variables come under reference data type.
- Default value of any reference variable is null.
- A reference variable can be used to refer to any object of the declared type or any compatible type.
- Example: Animal animal = new Animal ("giraffe");

Data Type	Size	Range	Default Value
boolean	1 bit	true or false	false
byte	1 byte	-128 to 127	0
short	2 byte	-32768 to 32767	0
char	2 byte	0 to 65535	'\u0000'
int	4 byte	-2^{31} to $2^{31} - 1$	0
long	8byte	-2^{63} to $2^{63} - 1$	0L
float	4 byte	-2^{31} to $2^{31} - 1$	0.0f
double	8 byte	-2^{63} to $2^{63} - 1$	0.0d

1.2.1 Structure of Java Program

Q8. Describe the Structure of a Java Program.

Ans : **(Dec.-19)**

The general structure of a Java program is shown below :

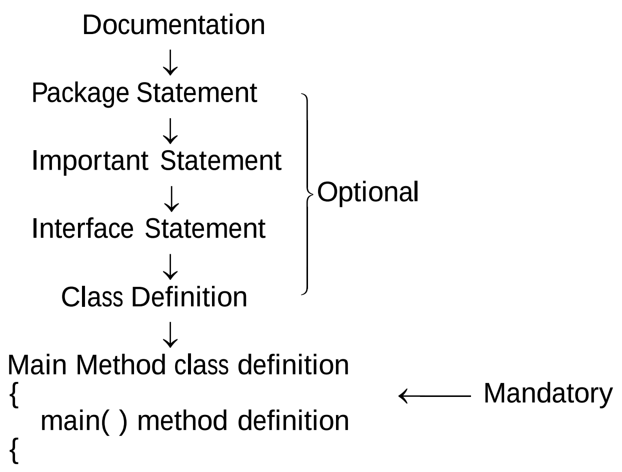


Figure: General Structure of a Java Program

1. Documentation

This section consists of set of comments about the program including the name of the program. This section basically helps in understanding the program.

2. Package Statement

This is the first statement in every Java program. It tells the compiler that the classes defined here belongs to this package. This statement is optional.

3. Import Statement

The import statement is optional and it tells the interpreter to include the classes from the package defined. This is the next statement after the package declaration but should be written before defining a class. There may be a number of import statements.

4. Interface Statement

The interface statement defines method declarations without body for the subclasses to provide implementation for them. This is optional, it is used only if multiple inheritance is required in the program.

5. Class Definition

This section consists of number of class definitions where each class consists of data members and methods. The number of classes required are based on the complexity of the problem.

6. Main Method Class

This is an essential section of a Java program. Every Java program must have a class definition that defines the main() method. This is essential because mainf) is the starting point for running Java stand-alone programs. The main() method instantiates the objects of various classes and establishes the communication between them. The program terminates on reaching the end of the main() method.

1.3 Type Casting

Q9. Explain about the type casting and type conversion in java.

Ans : **(July-21)**

Type casting is an explicit conversion of a value of one type into another type. And simply, the data type is stated using parenthesis before the value. Type casting in Java must follow the given rules,

1. Type casting cannot be performed on Boolean variables.
2. Type casting of integer datatype into any other datatype is possible. But. if the casting into smaller type is performed, it results in loss of data.
3. Type casting of floating point types into other float types or integer type is possible, but with loss of data.
4. Type casting of char type into integer types is possible. But, this also results in loss of data, since char holds 16-bits the casting of it into byte that results in loss of data or mixup characters.

Syntax

(target-type) expression;

Here, target-type is the specification to convert the expression into required type.

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Byte –> Short –> Int –> Long –> Float –> Double

Widening or Automatic Conversion

```
class Test
{
    public static void main(String[] args)
    {
            int i = 100;
                // automatic type conversion
            long l = i;
                // automatic type conversion
            float f = l;
            System.out.println("Int value "+i);
            System.out.println("Long value "+l);
            System.out.println("Float value "+f);
    }
}
```

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explict Conversion

char and number are not compatible with each other. Let's see when we try to convert one into other.

```
public class Test
{
        public static void main (String[] args)
{
                char ch = 'c';
                int num = 88;
                ch = num;
        }
}
```

Type promotion in Expressions

While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

1. Java automatically promotes each byte, short, or char operand to int when evaluating an expression.
2. If one operand is a long, float or double the whole expression is promoted to long, float or double respectively.

//Java program to illustrate Type promotion in Expressions

```
classTest
{
        public static void main(String args[])
        {
                byte b = 42;
                char c = 'a';
                shor t s = 1024;
                in ti = 50000;
                floatf = 5.67f;
                double d = .1234;
                // The Expression
                double result = (f * b) + (i / c) - (d * s);
                //Result after all the promotions are done
                System.out.println("result = "+ result);
        }
}
```

Explicit type casting in Expressions

While evaluating expressions, the result is automatically updated to larger data type of the operand. But if we store that result in any smaller data type it generates compile time error, due to which we need to type cast the result.

Example

```
//Java program to illustrate type casting int to byte
class Test
{
	public static void main(String args[])
	{
		byte b = 50;
		//type casting int to byte
		b = (byte)(b * 2);
		System.out.println(b);
	}
}
```

Your one stop destination for all data type conversions

byte TO	short	int	long	float	double	char	boolean
short TO	byte	int	long	float	double	char	boolean
int TO	byte	short	long	float	double	char	boolean
float TO	byte	short	Int	long	double	char	boolean
double TO	byte	short	Int	long	float	char	boolean
char TO	byte	short	Int	long	float	double	boolean
boolean TO	byte	short	Int	long	float	double	char

String and data type conversions

String TO	byte	short	int	long	float	double	char	boolean
byte	short	int	long	float	double	char	boolean	**TO String**

1.4 Conditional Statements, Loops

Q10. What are the various types of Control Statements in Java with a suitable examples.

Ans : **(Dec.-19)**

The various types of control statements in java are as follows :

1. Conditional statement
2. Loops statements
3. Branching statements

Q11. Explain the Conditional statements in java with suitable examples.

Ans : **(Dec.-19, June-19(MGU)**

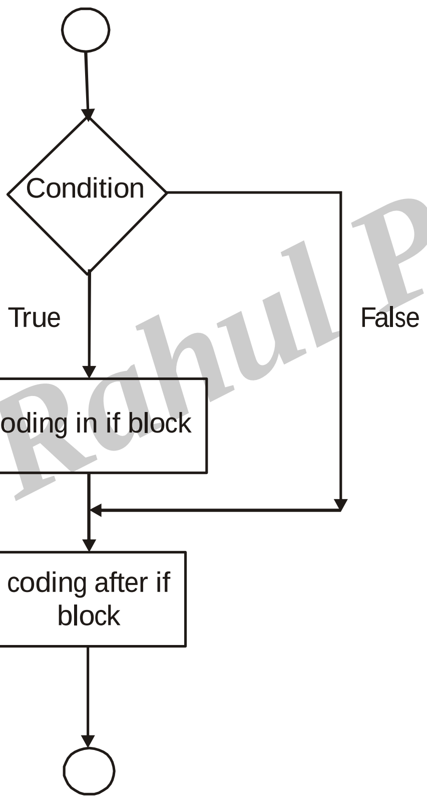
1. If Statement

A set of statements inside *if block* is executed when the condition evaluates to true. Otherwise, *if block* is skipped. If there is only one line in the *if block* then there is no need of braces.

Syntax

```
if(condition)
{
        //code to be executed
}
```

Flowchart of if statement



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

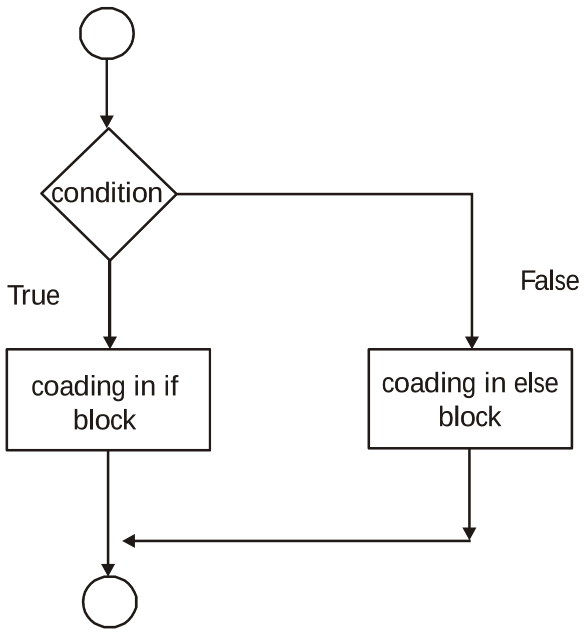
2. if-else statement in Java

If a condition evaluates to true then *if block* will be executed. Otherwise, *else block* will be executed.

Syntax

```
if(condition)
{
        //code to be executed when condition is true
}
else
{
        //code to be executed when condition is false
}
```

Flowchart of if-else statement



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

3. switch statement in Java

A switch statement enables a programmer to select one of the execution paths based on the value of an expression. It is very much similar to if-else if statement that we studied previously.

- A switch statement has one or more case labels and a single default label.
- A default label is executed when no matching value is found in the case labels.
- The argument of switch() must be one of the following types- byte, short, char, int or enum.
- You cannot have duplicate case labels, that is the same value cannot be used twice.
- The default label does not have to be placed at the end. It can appear anywhere in the switch block.

Working of switch statement

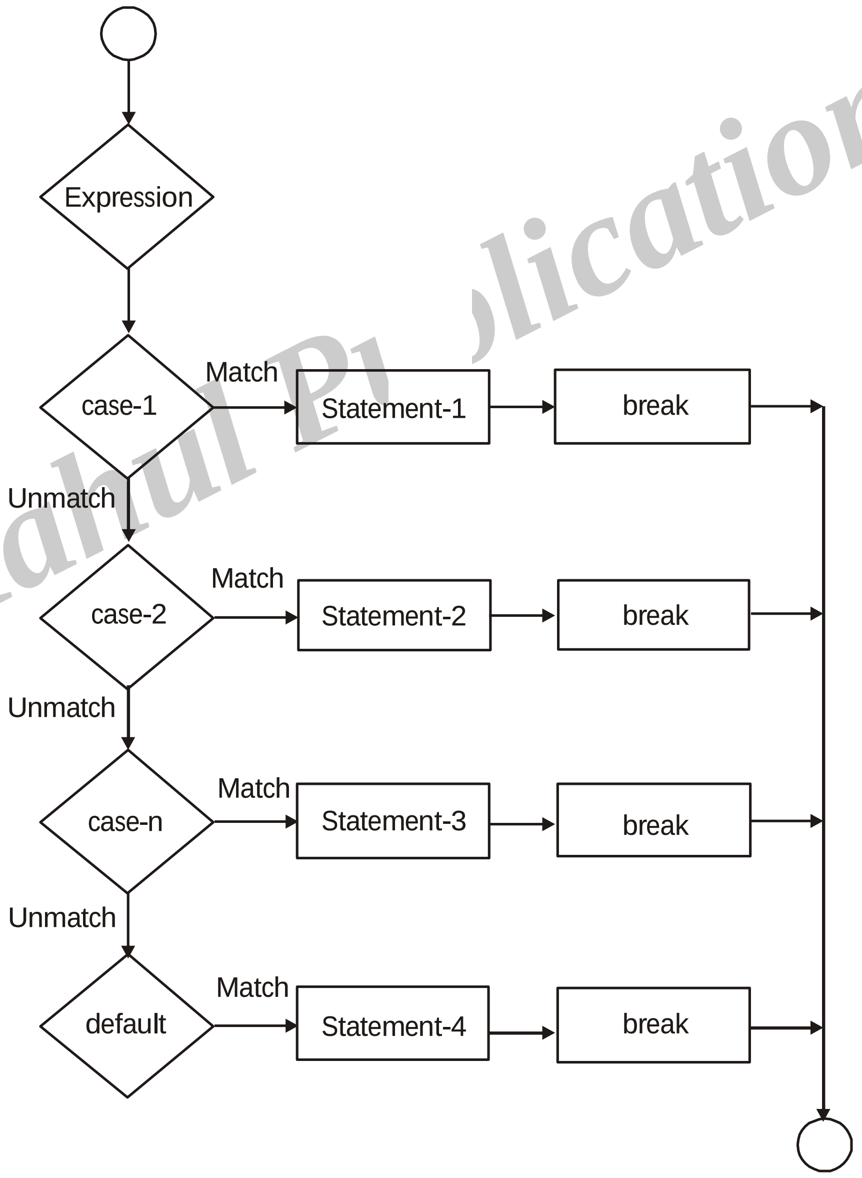
Let's start from the top of the switch block, each non-default case is checked. If the case gives true, the statements in that case and the following cases are executed until a break statement is encountered. If none of the non-default cases is true, and the execution control meets with the default case, the statements in the default case and all the following cases are executed until a break statement is encountered.

Syntax

```
switch(variable)
{
        case value1:
                //code to be executed
                break;
        case value2:
```

```
        //code to be executed
        break;
    ...
    case valuen:
        //code to be executed
        break;
    default:
        //code to be executed
        break;
}
```

Flowchart of switch statement



```
public class SwitchCaseStatementDemo
{
        public static void main(String[] args)
        {
              int a = 10, b = 20, c = 30;
              int status = -1;
              if (a > b && a > c)
              {
                     status = 1;
              }
              else if (b > c)
              {
                     status = 2;
              }
              else
              {
                     status = 3;
              }
              switch (status)
              {
                     case 1:
                            System.out.println(“a is the greatest”);
                            break;
                     case 2:
                            System.out.println(“b is the greatest”);
                            break;
                     case 3:
                            System.out.println(“c is the greatest”);
                            break;
                     default:
                            System.out.println(“Cannot be determined”);
              }
        }
}
```

Q12. What are the different looping statements in java? Explain.

Ans : **(Dec.-19, Dec.-18(MGU), Dec.-18(KU)**

Loop is very common control flow statement in programming languages such as java. We are going to describe the basics of "java loop". In this post, we will learn various ways to use loop in day-to-day programming habits.

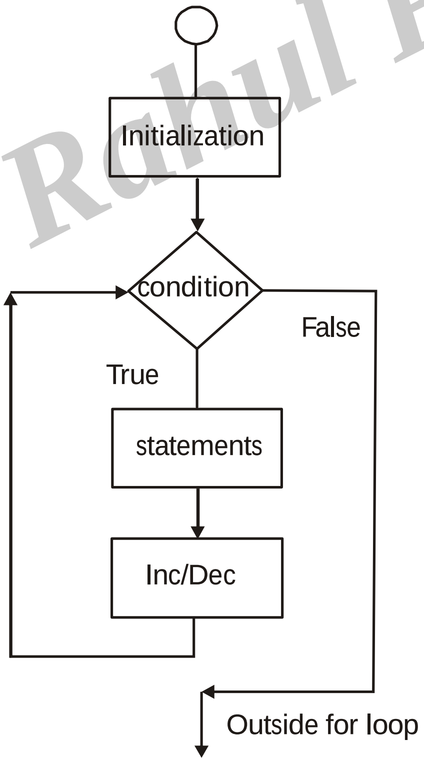
There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop

There are four types of loops:

1. For loop
2. For each loop
3. While loop
4. Do.While loop

1. For Loop

It is structured around a finite set of repetitions of code. So if you have a particular block of code that you want to have run over and over again a specific number of times the For Loop is helpful.



Syntax

```
for(initilization; conditional expression; increment expression)
{
        //repetition code here
}
```

Example

```
public class ForLoopDemo
{
        public static void main(String[] args)
        {
                System.out.println("Printing Numbers from 1 to 10"); for (int count = 1; count <= 10; count++)
                {
                        System.out.println(count);
                }
        }
}
```

2. For each Loop

This loop is supported from Java 5.

For each loop is mainly used for iterate the Array, List etc.

Syntax

```
for(declaration : expression)
{
        //Code Here
}
```

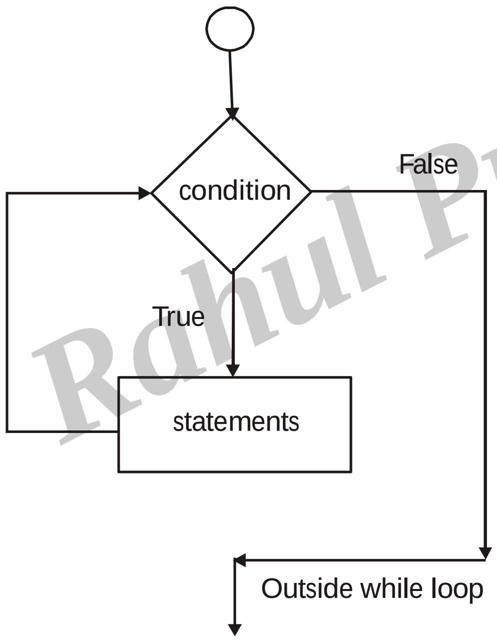
Example

```
import java.util.ArrayList;
class Test
{
        public static void main(String args[])
        {
                ArrayListlang = new ArrayList();
                lang.add("Java");
```

```
            lang.add("Python");
            lang.add("MongoDB");
            for(String i : lang)
            {
                  System.out.println(i);
            }
      }
}
```

3. While Loop

Another looping strategy is known as the While Loop. The While Loop is good when you don't want to repeat your code a specific number of times, rather, you want to keep looping through your code until a certain condition is met.



Syntax

```
while(Boolean_expression)
{
        //Repetition Code Here
}
```

Example

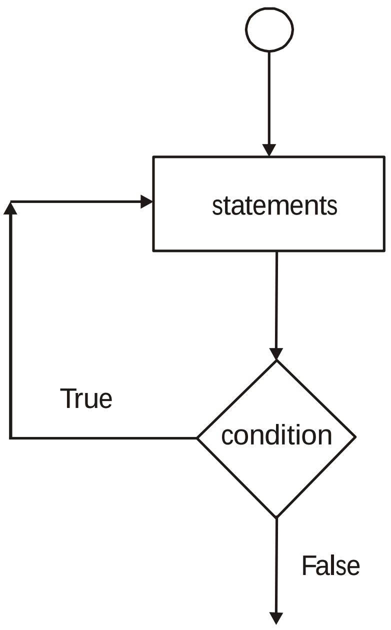
```
      public class Example
{
      public static void main(String args[])
      {
            int i = 1
            while(i<=10)
            {
                  System.out.println(2*i);
                  i++;
            }
      }
}
```

4. Do..While Loop

This type of loop is used in very rare cases because it does the same thing as a while loop does, except that a do..while loop is guaranteed to execute at least on time.

Syntax:

```
do
{
        //Code Here
}while(Boolean_expression);
```



Example

```
public class DoWhileLoopDemo
{
        public static void main(String[] args)
        {
                int count = 1;
                System.out.println("Printing Numbers
                from 1 to 10");
                do
                {
                        System.out.println(count++);
                } while (count <= 10);
        }
}
```

Q13. Explain about branching mechanism in java.

Ans : **(Dec.-19, Dec.-18)**

Branching statements can be used to skip over the loop by terminating a set of statements The varic types of branching statements m Ja-a are as follows,

(i) break statement

(ii) continue statement

(iii) return statement

(i) break Statement

The 'break' statement performs unconditional jump that terminates or exits the iteration or switch statement. It terminates the loop as soon as the control encounters it. And then the control is transferred to the statement that occurs after the iteration or switch statement. Therefore, it closes the smallest enclosing such as do, for, switch or while statements. It is written as,

Syntax

break;

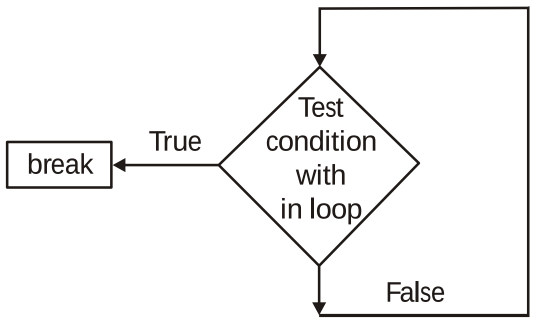


Fig.: Flowchart for break Statement

(ii) continue Statement

Continue statement is a jump statement. It tells the interpreter to continue the next iteration of the loop. It is a keyword used for continuing the next iteration of the loop. In contrast to the break statement the continue statement does not exit from the loop but transfers the control to the testing expression (while, do-while) and to the updating expression (for). The while and do- while loops can logically act as a jump statement that transfer the control to the end of the loop body. This is because both these statements transfer their control to two different positions.

Syntax

continue; x

(iii) return Statement

The last control statement is return. This is used to return from a method. When the control reaches to return statement, it causes the program control to transfer back to the caller of the method.

1.5 Classes, Objects, Class Declaration

Q14. Explain in detail about class & Objects in JAVA?

Ans :

In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

Object in Java

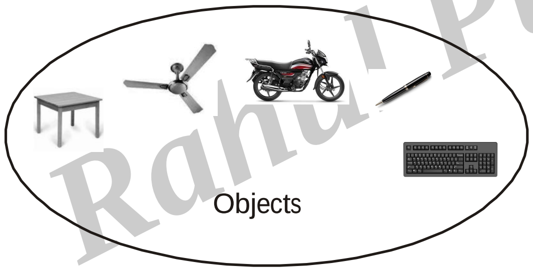
An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example

Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.



Object is an instance of a class

Class is a template or blueprint from which objects are created. So object is the instance (result) of a class.

Object Definitions

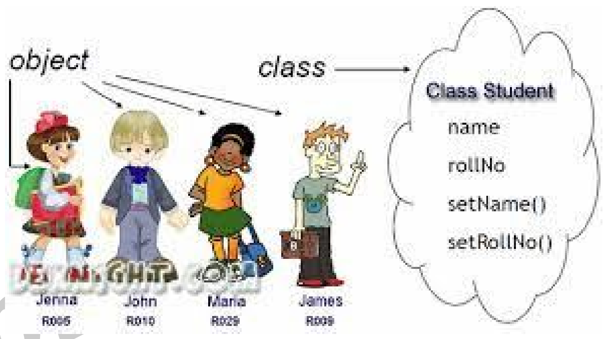
- Object is a real world entity.
- Object is a run time entity.
- Object is an entity which has state and behavior.
- Object is an instance of a class.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- fields
- methods
- constructors
- blocks
- nested class and interface



Syntax to declare a class:

```
class <class_name>
{
    field;
    method;
}
```

Instance variable in Java

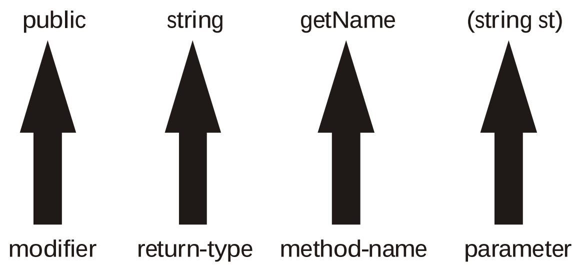
A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization



new keyword in Java

The new keyword is used to allocate memory at run time. All objects get memory in Heap memory area.

Object and Class Example: main within class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

Here, we are creating main() method inside the class.

File: Student.java

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

Rules for Java Class

- A class can have only public or default(no modifier) access specifier.
- It can be either abstract, final or concrete (normal class).
- It must have the class keyword, and class must be followed by a legal identifier.
- It may optionally extend one parent class. By default, it will extend java.lang.Object.
- It may optionally implement any number of comma-separated interfaces.
- The class's variables and methods are declared within a set of curly braces {}.
- Each .java source file may contain only one public class. A source file may contain any number of default visible classes.
- Finally, the source file name must match the public class name and it must have a .java suffix.

Important points

- A class name starts with keyword class.
- The keyword class is case sensitive. class isn't same as Class.
- The file that contains your coding is called source file, and it must have .java extension.
- A source file can have one or more classes or interfaces. Also, a source file cannot define more than one public class or interface.
- If there is one public class or interface in the file, then the filename must match the name of this public class or interface.
- When the source file is compiled , it generates one class file corresponding to each class in the source file. The name of the generated class file matches the name of the corresponding class in the source file.

Example

```
class Employee
{
    int id;
    String name;
    public void setId(int i)
    {
        Id=i;
    }
```

```
        public int getId()
        {
                return id;
        }
        public void setName(String n)
        {
                name=n;
        }
        public int getName()
        {
                return name;
        }
        public static void main(String args[])
        {
                Employee e = new Employee();
                e.setId(1);
                e.setName("Keshav");
                System.out.println("Employee ID =
                                        "+e.getId());
                System.out.println("Employee Name
                                    = "+e.getName());
        }
}
```

1.6 Creating Objects

Q15. Explain the process of creating objects in java?

Ans : **(Dec.-19)**

Fields Declaration

Field declarations are composed of three components, in order:

- Zero or more modifiers, such as public or private.
- The field'stype.
- The field's name.

In the example below fields of Rect class are named length and breadth and are all of data type integer (int). The public keyword identifies these fields as public members, accessible by any object that can access the class.

```
class Rect
{
public int length;
public int breadth;
}
```

There are several kinds of variables:

- Member variables in a class—these are calledfields.
- Variables in a method or block of code—these are called local variables.
- Variables in method declarations—these are calledparameters.

Method Declaration

A class should have methods that are necessary for manipulating the data contained in the class. Immediately after the declaration of instance variables inside the body of the class methods are declared. The general form of a method declarationis:

```
modifiers type method_name(parameter-list)
{
Method-body;
}
```

More generally, method declarations have six components, in order:

- **Modifiers:** such as public, private, and others you will learn about later
- **The return type:** the data type of the value returned by the method, or void if the method does not return a value.
- **The method name:** the rules for field names apply to method names as well, but the convention is a little different.
- **The parameter list in parenthesis**: a comma-delimited list of input parameters, preceded by their data types, enclosed by

parentheses, (). If there are no parameters, you must use empty parentheses.

- **An exception list:** to be discussed later.
- **The method body, enclosed between braces:** the method's code, including the declaration of local variables, goeshere.

Example

```
class Rect
{
        public int length;
        public int breadth;
        void getValues(int p,int q)
        {
              length=p;
              breadth=q;
        }
        int area()
        {
              int ans=length*breadth;
              return (ans);
        }
}
```

Explanation

The method getValues() has a return type of void because it does not return any values. Two integer values are passed which are then assigned to the instance variables length and breadth. This method is added to provide values to the class instance variables.

The method area() computes the area of a rectangle and returns the result with the 'return' keyword. Note that the parameter list is empty. Since the result would be an integer the return type of the method is specified asint.

Creating Objects

A typical Java program creates many objects, which as you know, interact by invoking methods. Through these object interactions, a program can carry out various tasks, such as implementing a GUI, running an animation, or sending and receiving information over a network. Once an object has completed the work for which it was created, its resources are recycled for use by other objects. A class provides the blueprint for objects; you create an object from a class. Each of the following statements creates anobject.

RectrectOne= newRect();

RectrectTwo= newRect();

Each of the above statements has three parts (discussed in detail below):

- **Declaration**: The code set in bold are all variable declarations that associate a variable name with an objecttype.
- **Instantiation**: The new keyword is a Java operator that creates the object.
- **Initialization**: The new operator is followed by a call to a constructor, which initializes the newobject.

Declaring a Variable to Refer to an Object

Previously, you learned that to declare a variable, you write: type name;

For example

int value;

This notifies the compiler that you will use **value** to refer to data whose type is **int**. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. You can also declare a reference variable on its own line.

For example

RectrectTwo;

If you declare rect Two like this, its value will be undetermined until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object. For that, you need to use the new operator. You must assign an object to rectTwo before you use it in your code. Otherwise, you will get a compilererror.

Instantiating a Class

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The phrase "instantiating a class" means the same thing as "creating an object." When you create an object, you are creating an "instance" of a class, therefore "instantiating" aclass.

The new operator returns a reference to the object it created. This reference is usually assigned to a variable of the appropriate type, like:

RectrectTwo = new Rect();

Initializing anObject

After we have created an object of the class, we can initialize the object in two ways. We can use the "." dot operator to provide values to the instance variables. Also we can call some method of the class which will help us in setting the values of the object variables.

Example

```
RectrectTwo=new Rect();
rectTwo.length=10;
rectTwo.breadth=20;
or
rectTwo.setData(10,20);
```

Using Objects

Once you've created an object, you probably want to use it for something. You may need to use the value of one of its fields, change one of its fields, or call one of its methods to perform an action.

Referencing an Object'sFields

Object fields are accessed by their name. You must use a name that is unambiguous. You may use a simple name for a field within its own class. For example, we can add a statement within the Rect class that prints the length and breadth:

System.out.println("Length and breadth are: "+ length + ", " + breadth);

In this case, length and breadth are simple names. Code that is outside the object's class must use an object reference or expression, followed by the dot (.) operator, followed by a simple field name, asin:

objectReference.fieldName

For example, the code in the Rect class, we can refer to the length and breadth fields within the Rect object named rectOne, the class must use the names rectOne.length and rectOne.breadth, respectively. The program uses two of these names to display the length and the breadth of rectOne:

System.out.println("Lenght of rectOne: " + rectOne.length);

System.out.println("Breadth of rectOne: " + rectOne.breadth);

Calling an Object's Methods

You also use an object reference to invoke an object's method. You append the method's name to the object reference, with an dot operator (.). Also, you provide, within enclosing parentheses, any arguments to the method. If the method does not require any arguments, use emptyparentheses.

objectReference.methodName(argumentList);

or

objectReference.methodName();

The Rect class has a method: area() to compute the rectangle's area. Here's the code that invokes this method:

System.out.println("Area of rectOne: " + rectOne.area());

Some methods, such as area(), return a value. For methods that return a value, you can use the method invocation in expressions. You can assign the return value to a variable, use it to make decisions, or control a loop. This code assigns the value returned by area() to the variable

areaOfRectangle:

int areaOfRectangle = rectOne.area();

The Garbage Collector

Managing memory explicitly is tedious and error-prone. The Java platform allows you to create as many objects as you want, and you don't have to worry about destroying them. The Java runtime environment deletes objects when it determines that they are no longer being used. This process is called garbage collection.

An object is eligible for garbage collection when there are no more references to that object. References that are held in a variable are usually dropped when the variable goes out of scope. Remember that a program can have multiple references to the same object; all references to an object must be dropped before the object is eligible for garbagecollection.

The Java runtime environment has a garbage collector that periodically frees the memory used by objects that are no longer referenced. The garbage collector does its job automatically when it determines that the time is right.

Short Question and Answers

1. Type conversion in java.

Ans

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Byte –> Short –> Int –> Long –> Float –> Double

Widening or Automatic Conversion

```
class Test
{
        public static void main(String[] args)
        {
                int i = 100;
                        // automatic type conversion
                long l = i;
                        // automatic type conversion
                float f = l;
                System.out.println("Int value "+i);
                System.out.println("Long value "+l);
                System.out.println("Float value "+f);
        }
}
```

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explict Conversion

char and number are not compatible with each other.

2. Explain the features of Java.

Ans :

- **Secure:** Java is Secure Language because of its many features it enables to develop virus-free, tamper-free systems. Authenti-cation techniques are based on public-key encryption. Java does not support pointer explicitly for the memory. All Program Run under the sandbox.
- **Robust:** Java was created as a strongly typed language. Data type issues and problems are resolved at compile-time, and implicit casts of a variable from one type to another are not allowed.

  Memory management has been simplified java in two ways. First Java does not support direct pointer manipulation or arithmetic. This make it possible for a java program to overwrite memory or corrupt data.

  Second, Java uses runtime garbage collection instead of instead of freeing of memory. In languages like C++, it Is necessary to delete or free memory once the program has finished with it.
- **Platform-independent:** Java Language is platform-independent due to its hardware and software environment. Java code can be run on multiple platforms e.g. windows, Linux, sun Solaris, Mac/Os etc. Java code is compiled by the compiler and converted into byte code. This byte code is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).
- **Architecture neutral:** It is not easy to write an application that can be used on Windows, UNIX and a Macintosh. And its getting more complicated with the move of windows to nonIntel CPU architectures.

  Java takes a different approach. Because the Java compiler creates byte code instructions that are subsequently interpreted by the java interpreter, architecture neutrality is achieved in the implementation of the java interpreter for each new architecture.
- **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.

  In java, all primitive types (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run. This is in direct contrast to languages like C and C++ that leave the sized of primitive types up to the compiler and developer.

  Additionally, Java is portable because the compiler itself is written in Java.
- **Dynamic:** Because it 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.

  At runtime, the java interpreter performs name resolution while linking in the necessary classes. The Java interpreter is also responsible for determining the placement of object in memory. These two features of the Java interpreter solve the problem of changing the definition of a class used by other classes.

3. Structure of a Java Program.

Ans :

The general structure of a Java program is shown below :

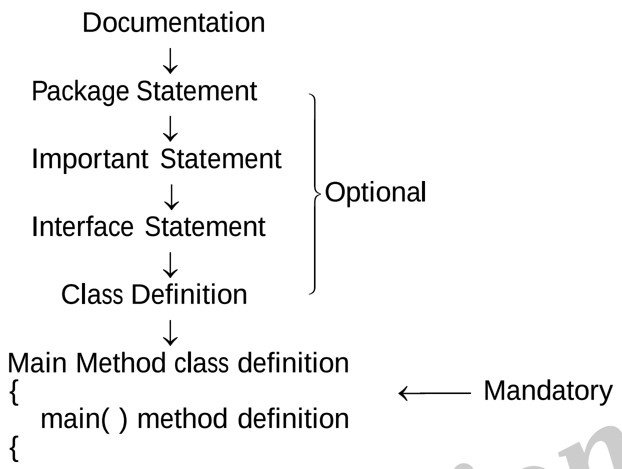


Figure: General Structure of a Java Program

1. Documentation

This section consists of set of comments about the program including the name of the program. This section basically helps in understanding the program.

2. Package Statement

This is the first statement in every Java program. It tells the compiler that the classes defined here belongs to this package. This statement is optional.

3. Import Statement

The import statement is optional and it tells the interpreter to include the classes from the package defined. This is the next statement after the package declaration but should be written before defining a class. There may be a number of import statements.

4. Interface Statement

The interface statement defines method declarations without body for the subclasses to provide implementation for them. This is optional, it is used only if multiple inheritance is required in the program.

5. Class Definition

This section consists of number of class definitions where each class consists of data members and methods. The number of classes required are based on the complexity of the problem.

6. Main Method Class

This is an essential section of a Java program. Every Java program must have a class definition that defines the main() method. This is essential because mainf) is the starting point for running Java stand-alone programs. The main() method instantiates the objects of various classes and establishes the communication between them. The program terminates on reaching the end of the main() method.

4. Define Java.

Ans :

Introduction

Java is a high-level programming language that follows object-oriented programming principle. The main motto of Java programming language is "write once run anywhere." The syntax and semantics of Java are very much similar to C and C++ programming language.

Java is considered a programming language and a platform. As a programming language, it is a general-purpose, object-oriented high-level programming language that has its own syntax and style. As a platform, it provides an environment in which Java applications run.

Java has four different platforms:-

(i) Java Standard Edition (Java SE): In general, most of the programmers use Java SE to do programming in Java. It provides the core functionality of the Java language. APIs that are used for database access, GUI development, networking, etc. are present in this platform.

(ii) Java Enterprise Edition (Java EE): This platform is used to develop and run reliable, secure and large-scale network applications. It is built on the top of Java SE.

(iii) Java Micro Edition (Java ME): This platform is suitable for small devices like smartphones. It is a subset of Java SE and provides API applicable for application development on small appliances. It has small footprint virtual machine that enables Java applications to run smoothly on less memory constrained devices.

(iv) JavaFX: It is used to create rich internet applications. You can think JavaFX as an alternate of Swing. It helps programmers to create applications with modern look-and-feel and utilize hardware-accelerated graphics.

5. Java Essentials.

Ans :

The essentials of Java programming language are as follows,

(i) High-level Language

Java is known to be high-level language that supports unique features. It is quite similar to C and C++.

(ii) Bytecode of Java

A bytecode is nothing but intermediate code that the compiler generates. This code is executed by JVM.

(iii) Java Virtual Machine (JVM)

A bytecode consists of optimized set of instructions which are usually executed by Java runtime system on a special machine called Java Virtual Machine. A JVM is a bytecode interpreter. It performs line by line execution of bytecode.

Java has a neutral-architecture since it allows programs to run on different platforms. The java compiler generates the bytecode that are neutral-architecture in order to achieve the capabilities of cross-architecture. These instructions can be interpreted easily on any platform translated into machine code. Java Runtime Environment (JRE) contains JVM, class libraries and different supporting files.

6. JVM

Ans

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the machine language for JVM is byte code. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. JVM is the one that makes java platform independent.

7. Type casting.

Ans :

Type casting' is an explicit conversion of a value of one type into another type. And simply, the data type is stated using parenthesis before the value. Type casting in Java must follow the given rules,

1. Type casting cannot be performed on Boolean variables.
2. Type casting of integer datatype into any other datatype is possible. But. if the casting into smaller type is performed, it results in loss of data.
3. Type casting of floating point types into other float types or integer type is possible, but with loss of data.
4. Type casting of char type into integer types is possible. But, this also results in loss of data, since char holds 16-bits the casting of it into byte that results in loss of data or mixup characters.

8. branching mechanism in java.

Ans :

Branching statements can be used to skip over the loop by terminating a set of statements The varic types of branching statements m Ja-a are as follows,

(i) break statement

(ii) continue statement

(iii) return statement

(i) break Statement

The 'break' statement performs unconditional jump that terminates or exits the iteration or switch statement. It terminates the loop as soon as the control encounters it. And then the control is transferred to the statement that occurs after the iteration or switch statement. Therefore, it closes the smallest enclosing such as do, for, switch or while statements. It is written as,

Syntax

break;

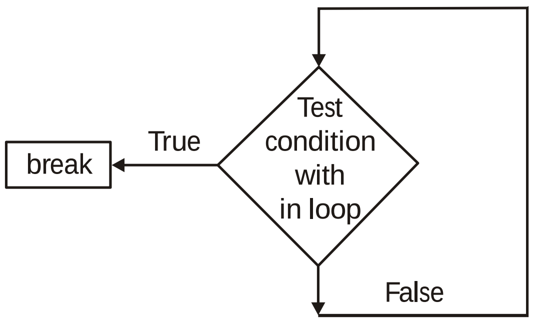


Fig.: Flowchart for break Statement

(ii) continue Statement

Continue statement is a jump statement. It tells the interpreter to continue the next iteration of the loop. It is a keyword used for continuing the next iteration of the loop. In contrast to the break statement the continue statement does not exit from the loop but transfers the control to the testing expression (while, do-while) and to the updating expression (for). The while and do- while loops can logically act as a jump statement that transfer the control to the end of the loop body. This is because both these statements transfer their control to two different positions.

Syntax

continue; x

(iii) return Statement

The last control statement is return. This is used to return from a method. When the control reaches to return statement, it causes the program control to transfer back to the caller of the method.

9. Garbage Collector.

Ans

Managing memory explicitly is tedious and error-prone. The Java platform allows you to create as many objects as you want, and you don't have to worry about destroying them. The Java runtime environment deletes objects when it determines that they are no longer being used. This process is called garbage collection.

An object is eligible for garbage collection when there are no more references to that object. References that are held in a variable are usually dropped when the variable goes out of scope. Remember that a program can have multiple references to the same object; all references to an object must be dropped before the object is eligible for garbagecollection.

The Java runtime environment has a garbage collector that periodically frees the memory used by objects that are no longer referenced. The garbage collector does its job automatically when it determines that the time is right.

10. Object in Java.

Ans

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

Choose the Correct Answers

1. Java was developed in the year _________ [b]

    (a) 1990 (b) 1991

    (c) 1993 (d) 1996

2. First name of java _________ [b]

    (a) J Language (b) Oak

    (c) OOPS (d) None

3. Which of the following is the smallest integer _________ [c]

    (a) Int (b) Long

    (c) Byte (d) short

4. Range of byte data type is _________ [c]

    (a) -128 to 255 (b) -128 to 256

    (c) -128 to 127 (d) -127 to 128

5. Which of the following is not supported in Java _________ [c]

    (a) Multi Threading (b) Reflection

    (c) Operatior Overloading (d) Garbage Collection

6. Which of the following option leads to the portability and security of Java? [a]

    (a) Bytecode is executed by JVM

    (b) The applet makes the Java code secure and portable

    (c) Use of exception handling

    (d) Dynamic binding between objects

7. Which of the following is not a Java features? [c]

    (a) Dynamic (b) Architecture Neutral

    (c) Use of pointers (d) Object-oriented

8. _________ is used to find and fix bugs in the Java programs. [d]

    (a) JVM (b) JRE

    (c) JDK (d) JDB

9. Which of the following is a valid declaration of a char? [a]

   (a) char ch = '\utea'; (b) char ca = 'tea';

   (c) char cr = \u0223; (d) char cc = '\itea';

10. What is the return type of the hashCode() method in the Object class? [b]

   (a) Object (b) int

   (c) long (d) void

Fill in the Blanks

1. __________ are the basic runtime entities in OOPS
2. __________ means the ability to take more tank one form.
3. A __________ acts as a intermediately between virtual machine and real time.
4. The __________ operator is also used for concatenating two strings.
5. The increment and decrement operator can be applied __________ or __________.
6. Java is known to be __________ that supports unique features. It is quite similar to C and C++.
7. WORA Stands for __________.
8. A __________ statement enables a programmer to select one of the execution paths based on the value of an expression.
9. __________ is very common control flow statement in programming languages.
10. The __________ statement performs unconditional jump that terminates or exits the iteration or switch statement.
11. The last control statement is __________.

Answers

1. Object
2. Polymorphism
3. Virtual Machine Code
4. Pluse(+)
5. Before,After
6. high-level language
7. Write Once and Run Anywhere
8. Switch
9. Loop
10. Break
11. Return

One Mark Answers

1. Java.

Ans

Java is a high-level programming language that follows object-oriented programming principle. The main motto of Java programming language is “write once run anywhere.”

2. JVM.

Ans

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the machine language for JVM is byte code.

3. Type casting.

Ans

Type casting is an explicit conversion of a value of one type into another type. And simply, the data type is stated using parenthesis before the value.

4. Object in java.

Ans

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc.

5. Garbage Collector.

Ans

Managing memory explicitly is tedious and error-prone. The Java platform allows you to create as many objects as you want, and you don’t have to worry about destroying them. The Java runtime environment deletes objects when it determines that they are no longer being used. This process is called garbage collection.

UNIT II

Method Declaration and Invocation, Method Overloading, Constructors – Parameterized Constructors, Constructor Overloading, Cleaning-up unused Objects. Class Variables &Method-static Keyword, this Keyword, One-Dimensional Arrays, Two-Dimensional Arrays, Command-Line Arguments, Inner Class.

Inheritance: Introduction, Types of Inheritance, extends Keyword, Examples, Method Overriding, super, final Keyword, Abstract classes, Interfaces, Abstract Classes Verses Interfaces.

Packages: Creating and Using Packages, Access Protection, Wrapper Classes, String Class, StringBuffer Class.

2.1 Method Declaration and Invocation

Q1. What are the methods in Java? Explain its creation?

Ans :

It is a collection of statement which is used to perform some operation. The main benefit to use method is that it provides code reusability. Once a method is created, then it can be reused multiple times as per requirements.

Creating Method

To create a method following syntax is used:

```
Access_modifierReturn_typeMethod_name (Parameter_list)
{
//Body of method
}
```

Where,

- **Access_modifier:** It defines the access type of methods like private, public, protected.
- **Return_type:** Specify the return type of method.
- **Method_name:** It shows the name of method.
- **Parameter _list:** Contains a list of parameters with their type. A method contains either zero parameter or more than 0 parameters.
- **Body of method:** Contains the statements which perform some operation.

Example

```
public static int sum(int i, int j)
{
        int total;
        total = i + j;
        return total;
}
```

To call the method following syntax is used:

```
        Method_name(Parameter_list);
```

Example

```
public class Intellipaat
{
        public static void main(String[] args)
        {
                int a = 10;
                int b = 20;
                int c = sum(a, b);
                System.out.println("Sum of two
                numbers = " + c);
        }
        public static int sum(int i, int j)
        {
                int total;
                total = i + j;
                return total;
        }
}
```

Q2. Explain about method declaration.

Ans :

Every method contains the name of a method and list of parameters in a class that are unique. The uniqueness of any parameter is considered according to the number of parameters and order of parameters.

The various elements involved in method declaration are as follows,

(i) modifiers

Modifiers are optional and are used in the method declaration. There are a number of modifiers that can be used with a method declaration. The following are some of the modifiers which are optional,

- **Public, Protected, Default or Private:** They are used for defining the scope.
- **Static:** The class methods and variables are declared by static. An object need not be created to invoke/call the method.
- **Abstract:** The class is declared abstract when it has to be overridden in its subclasses.
- **Final:** The method should not be overridden in a subclass.
- **Native:** The method can be implemented in different languages.
- **Synchronized:** The method needs a lock or monitor before its execution.
- **Throws:** This method is used to throw a number of exceptions.

(ii) datatype

The return type may be void or any other value. When a method is declared with any return type then it should contain return statement before it exits.

(iii) methodname

The name of the method should be a identifier which is valid.

(iv) parameters

The list of parameters can be zero or more. Comma is used to separate the parameter.

(v) Curly Braces

The curly braces are used to enclose the bod} of method. A method has a set of statements to execute them in any order. A body of method can be empty.

Q3. Write about method invocation.

Ans :

The instance methods or the class methods are not run by themselves, they should be called or invoked by their objects. A method is invoked or called by an object then the values of it are given to the method. The data which is given to a method is referred to as an argument or a parameter. The arguments which are needed for any method are defined in the list of parameters.

There are two types of arguments of parameters. They are as follows,

1. Actual Parameter

The parameters that are passed to a method call are called actual parameters. These parameters are defined in the calling method. Actual parameters can be variables, constants or expressions.

2. Formal Parameter

The parameters that are used in the method definition are called formal parameters. These parameters belong to the called method. These can be only variables but not expressions or constants.

2.1.1 Method Overloading

Q4. Write about Method Overloading.

(OR)

Explain different types of method overloading.

Ans : **(Dec.-19)**

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as method overloading.

Suppose we have to perform addition of given number but there can be any number of arguments, if we write method such as a(int, int)for two arguments, b(int, int, int) for three arguments then it is very difficult for you and other programmer to understand purpose or behaviors of method they can not identify purpose of method. So we use method overloading to easily figure out the program. For example above two methods we can write sum(int, int) and sum(int, int, int) using method overloading concept.

Syntax

```
class class_Name
{
        Returntype   method()
        {.........}
        Returntype   method(datatype1 variable1)
        {
            .........
        }
        Returntype   method(datatype1 variable1, datatype2 variable2)
        {
            .........
        }
        Returntype   method(datatype2 variable2)
        {
            .........
        }
        Returntype   method(datatype2 variable2, datatype1 variable1)
        {
            .........
        }
}
```

Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments or parameters
- By changing the data type

By changing number of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

Example

```
public class Sum
{
        // Overloaded sum(). This sum takes two int parameters
        public int sum(int x, int y)
        {
        return(x + y);
        }
        // Overloaded sum(). This sum takes three int parameters
        public int sum(int x, int y, int z)
        {
        return(x + y + z);
        }
        // Overloaded sum(). This sum takes two double parameters
        public double sum(double x, double y)
        {
        return(x + y);
        }
        // Driver code
        public static void main(String args[])
        {
              Sum s = new Sum();
              System.out.println(s.sum(10, 20));
              System.out.println(s.sum(10, 20, 30));
              System.out.println(s.sum(10.5, 20.5));
        }
}
```

Output:

```
        30
        60
        31.0
```

Q5. Compare and contrast method overloading and method overriding.

Ans : **(July-21, Dec.-18)**

S.No.	Method Overloading	Method Overriding
1	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

2.2 Constructors

2.2.1 Parameterized Constructors

Q6. Define Constructors. Explain different types of constructors in java.

Ans : **(Dec.-18, KU)**

Constructor is a special method that creates and return an object of the class in which they are defined. Constructor has the same name as that of class and has no return type not even void

Types of Constructor in Java

There are three types of constructor in Java-

(i) Default Constructor

(ii) Parameterized Constructor

(iii) Copy Constructor

(i) Default Constructor

This constructor takes no argument and is called when an object is created without any explicit initialization.

Note

The compiler provides default constructor when you do not write any constructor for a class. If you provide at least one constructor for the class, then the compiler does not provide any constructor.

Example

```
public class Car
{
        public Car()
        {
                System.out.println("Car is created.");
        }
        public static void main(String args[])
        {
                Car c = new Car();
        }
}
```

Output

```
        Car is created.
```

(ii) Parameterized Constructor

This constructor is called when an object is created and it is initialized with some values at the time of creation.

Example

```
public class Employee
{
        int id;
        String name;
        public Employee(int i, String n)
        {
                id = i;
                name = n;
        }
        void show()
        {
                System.out.println(id+" "+name);
        }
        public static void main(String args[])
        {
                Employee emp1 = new Employee(1, "Govind");
                Employee emp2 = new Employee(2, "Akash");
                emp1.show();
                emp2.show();
        }
}
```

Output

```
        1 Govind
        2 Akash
```

(iii) Copy Constructor

This constructor is called when an object is created and it is initialized with some other object of the same class at the time of creation.

Example

```
public class Employee
{
        int id;
        String name;
        public Employee(int i, String n)
        {
                id = i;
                name = n;
        }
        public Employee(Employee e)
        {
                id = e.id;
                name = e.name;
        }
        void show()
        {
                System.out.println(id+" "+name);
        }
        public static void main(String args[])
        {
        Employee emp1 = new Employee(1, "Govind");
        Employee emp2 = new Employee(emp1);
        emp1.show();
        emp2.show();
        }
}
```

Output

```
        1 Govind
        1 Govind
```

2.2.2 Constructor Overloading

Q7. Explain about Constructor Overloading in java.

Ans :

➢ When you write more than one constructor and each constructor has a different argument list, this is called constructor overloading.

➢ It helps the programmer by providing alternative ways for instantiating objects of a class.

Example

```
public class Employee
{
        int id;
        String name;
        int salary;
        public Employee(int i, String n)
        {
            id=i;
            name=n;
            salary=0;
        }
        public Employee(int i, String n, int s)
        {
            id=i;
            name=n;
            salary=s;
        }
        void show()
        {
            System.out.println(id+" "+name+" "+salary);
        }
        public static void main(String args[])
        {
            Employee emp1 = new Employee(1,
            "Govind");
            Employee emp2 = new Employee(2,
            "Akash", 10000);
        emp1.show();
        emp2.show();
    }
}
```

Output

```
1 Govind 0
2 Akash 10000
```

Rules for calling a constructor

There are two rules for calling a constructor-

➢ **Outside the class**: The constructor can only be called with the help of new operator that is when you create an object of the class. For example, new LinkedList() will call the default constructor of LinkedList class

➢ **Inside the class**: The constructor can only be invoked from within another constructor using this or super keyword and not from anywhere else.

2.3 Cleaning-up unused Objects

Q8. Explain about Cleaning-up unused Objects.

Ans :

Dynamic allocation of objects is done using new keyword in java. Some times allocation of new objects fail due to insufficient memory. In such cases memory space consumed by unused objects is deallocated and made available for reallocation. This is done manually in languages such as C and C++ using delete keyword. In java, a new concept called garbage collection is introduced for this purpose. It is a trouble-tree approach which reclaims the objects automatically. This is done without the interference of the programmer. The objects which are not used for long time and which does not have'kny references are identified and their space is deallocated. This recycled space can now be used by other objects.

Garbage Collector is a program that manages memory automatically wherein de-allocation of objects is handled by Java rather than the programmer. In the Java programming language, dynamic allocation of objects is achieved using the new operator. An object once created uses some memory and the memory remains allocated till there are references for the use of the object.

When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed. There is no explicit need to destroy an object as Java handles the de-allocation automatically.

The technique that accomplishes this is known as Garbage Collection. Programs that do not de-allocate memory can eventually crash when there is no memory left in the system to allocate. These programs are said to have memory leaks.

Garbage collection in Java happens automatically during the lifetime of the program, eliminating the need to de-allocate memory and thereby avoiding memory leaks. In C language, it is the programmer's responsibility to de-allocate memory allocated dynamically using free() function. This is where Java memory management leads.

Note : All objects are created in Heap Section of memory.

To Learn Garbage Collector Mechanism in Java

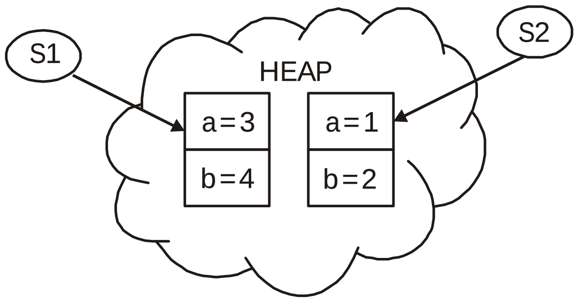
Step 1:

Copy the following code into an editor.

```
class Student
{
        int a;
        int b;
        public void setData(int c,int d)
        {
                a=c;
                b=d;
        }
        public void showData()
        {
                System.out.println("Value of a = "+a);
                System.out.println("Value of b = "+b);
        }
        public static void main(String args[])
        {
                Student s1 = new Student();
                Student s2 = new Student();
                s1.setData(1,2);
                s2.setData(3,4);
                s1.showData();
                s2.showData();
                //Student s3;
                //s3=s2;
                //s3.showData();
                //s2=null;
                //s3.showData();
                //s3=null;
                //s3.showData();
        }
}
```

Step 2:

Save, Compile and Run the code. As shown in the diagram, two objects and two reference variables are created.

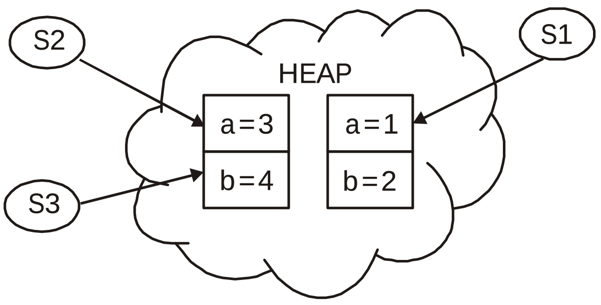


Step 3:

Uncomment line # 20,21,22. Save, compile & run the code.

Step 4:

As shown in the diagram below, two reference variables are pointing to the same object.

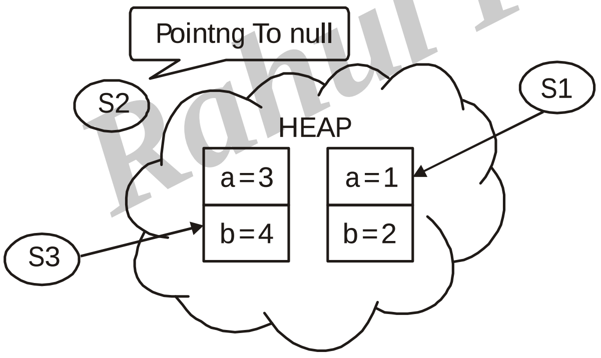


Step 5:

Uncomment line # 23 & 24. Compile, Save & Run the code

Step 6:

As show in diagram below, s2 becomes null, but s3 is still pointing to the object and is not eligible for java garbage collection.

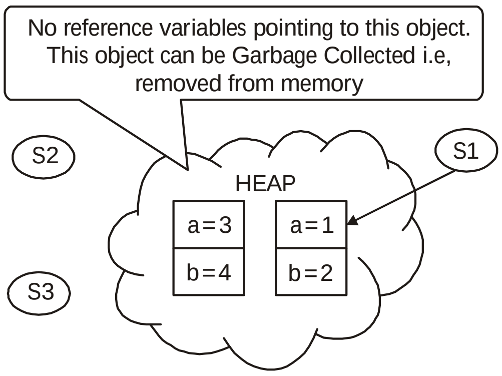


Step 7:

Uncomment line # 25 & 26. Save, Compile & Run the Code

Step 8:

At this point there are no references pointing to the object and becomes eligible for garbage collection. It will be removed from memory, and there is no way of retrieving it back.


No reference variables pointing to this object. This object can be Garbage Collected i.e, removed from memory
S2
S1
HEAP
a=3
b=4
a=1
b=2
S3

2.4 Class Variables & Method

Q9. What is Static Variable in Java?

Ans :

Static variable in Java is variable which belongs to the class and initialized only once at the start of the execution.

- It is a variable which belongs to the class and not to object(instance)
- Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
- A single copy to be shared by all instances of the class
- A static variable can be accessed directly by the class name and doesn't need any object

Syntax:

<class-name>.<variable-name>

2.4.1 Static Keyword

Q10. Explain about static keyword.

Ans :

The static keyword is used in java mainly for memory management. It is used with variables, methods, blocks and nested class. It is a keyword that are used for share the same variable or method of a given class. This is used for a constant variable or a method that is the same for every instance of a class. The main method of a class is generally labeled static.

No object needs to be created to use static variable or call static methods, just put the class name before the static variable or method to use them. Static method can not call non-static method.

In java language static keyword can be used for following

- variable (also known as class variable)
- method (also known as class method)
- block
- nested class

Static variable

If any variable we declared as static is known as static variable.

- Static variable is used for fulfill the common requirement. For Example company name of employees, college name of students etc. Name of the college is common for all students.
- The static variable allocate memory only once in class area at the time of class loading.

Advantage of static variable

Using static variable we make our program memory efficient (i.e. it saves memory).

When and why we use static variable

Suppose we want to store record of all employee of any company, in this case employee id is unique for every employee but company name is common for all. When we create a static variable as a company name then only once memory is allocated otherwise it allocate a memory space each time for every employee.

Syntax for declare static variable

```
public static variable Name;
```

Syntax for declare static method

```
public static void method Name()
{
    .......
    .......
}
```

Syntax for access static methods and static variable

```
class Name.variableName=10;
class Name.methodName();
```

Example

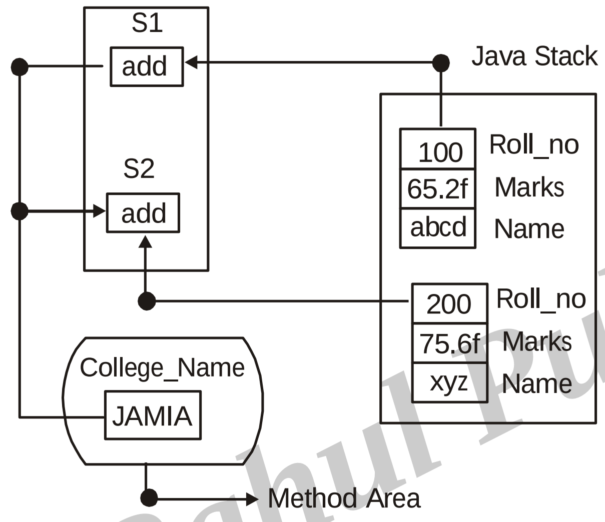
```
public static final double PI=3.1415;
public static void main(Stringargs[])
{
......
......
}
```

Example

```
class Student
{
introll_no;
String name;
static String College_Name="ITM";
}
class Static Demo
{
    public static void main(Stringargs[])
    {
        Student s1=new Student();
        s1.roll_no=100;
        s1.name="abcd";
        System.out.println(s1.roll_no);
        System.out.println(s1.name);
        System.out.println(Student.College_
        Name);
        Student s2=newStudent();
        s2.roll_no=200;
        s2.name="zyx";
        System.out.println(s2.roll_no);
        System.out.println(s2.name);
        System.out.println(Student.
        College_Name);
    }
}
```

Output

```
100
abcd
ITM
200
zyx
ITM
```



In the above image static data variable are store in method area and non static variable is store in java stack.

2.4.2 This Keyword

Q11. Write about this Keyword in java.

Ans :

Keyword THIS is a reference variable in Java that refers to the current object.

The various usages of 'THIS' keyword in Java are as follows:

- ➢ It can be used to refer instance variable of current class
- ➢ It can be used to invoke or initiate current class constructor
- ➢ It can be passed as an argument in the method call
- ➢ It can be passed as argument in the constructor call
- ➢ It can be used to return the current class instance

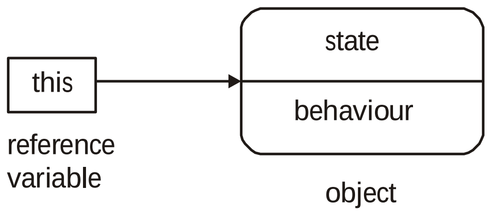
Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. This can be used to refer current class instance variable.
2. This can be used to invoke current class method (implicitly)
3. This() can be used to invoke current class constructor.
4. This can be passed as an argument in the method call.
5. This can be passed as argument in the constructor call.
6. This can be used to return the current class instance from the method.

Suggestion

If you are beginner to java, lookup only three usage of this keyword.



The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Understanding the problem without this keyword

Let's understand the problem if we don't use this keyword by the example given below:

```
class Student
{
    int rollno;
    String name;
    float fee;
```

```
        Student(int rollno,Stringname,float fee)
        {
                rollno=rollno;
                name=name;
                fee=fee;
        }
        void display()
        {
                System.out.println(rollno+"
"+name+" "+fee);
        }
}
class TestThis1
{
        public static void main(String args[])
        {
                Student                    s1=new
Student(111,"ankit",5000f);
                Student                    s2=new
Student(112,"sumit",6000f);
                s1.display();
                s2.display();
        }
}
```

Output

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable. The solution for the above program is….

Program

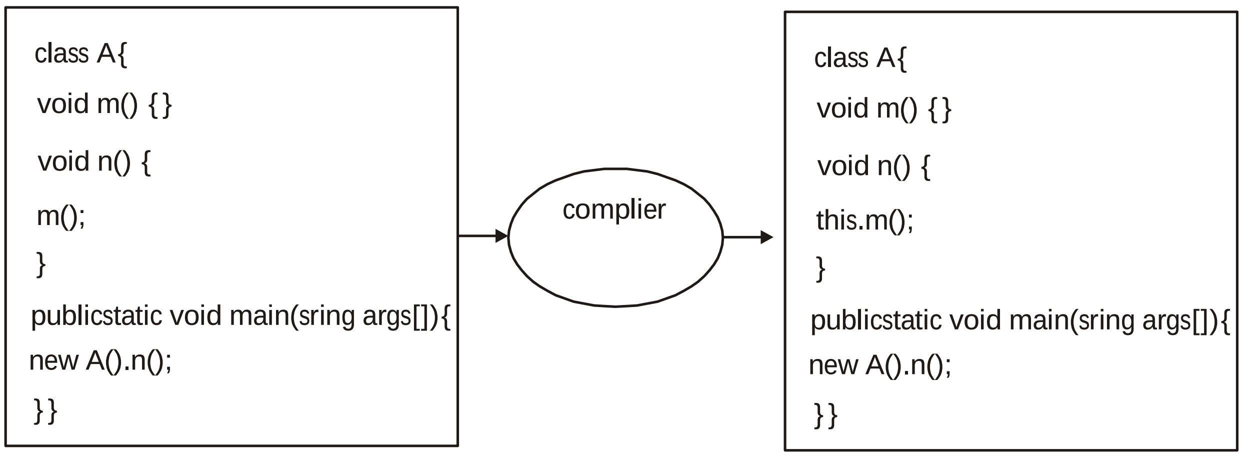
```
class Student
{
        int rollno;
        String name;
        float fee;
        Student(int rollno,Stringname,float fee)
        {
                this.rollno=rollno;
                this.name=name;
                this.fee=fee;
        }
        void display()
        {
                System.out.println(rollno+"
                "+name+" "+fee);
        }
}
class TestThis2
{
        public static void main(String args[])
        {
                Student s1=new Student(111, "ankit",
                5000f);
                Student s2=new Student(112, "sumit",
                6000f);
                s1.display();
                s2.display();
        }
}
```

Output

```
111 ankit 5000
112 sumit 6000
```

invoke current class method

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example



```
class A{
void m() {}
void n() {
m();
}
publicstatic void main(sring args[]){
new A().n();
}}
```

```
class A{
void m() {}
void n() {
this.m();
}
publicstatic void main(sring args[]){
new A().n();
}}
```

2.5 ARRAYS

Q12. What is an Array?

Ans :

Java array is an object the 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.

Array in java is index based, first element of the array is stored at 0 index.

Creating Arrays

You can create an array by using the new operator with the following syntax

Syntax

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things-

- ➢ It creates an array using new dataType [arraySize].
- ➢ It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below “

```
dataType[] arrayRefVar = new dataType [arraySize];
```

Alternatively you can create arrays as follows-

```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

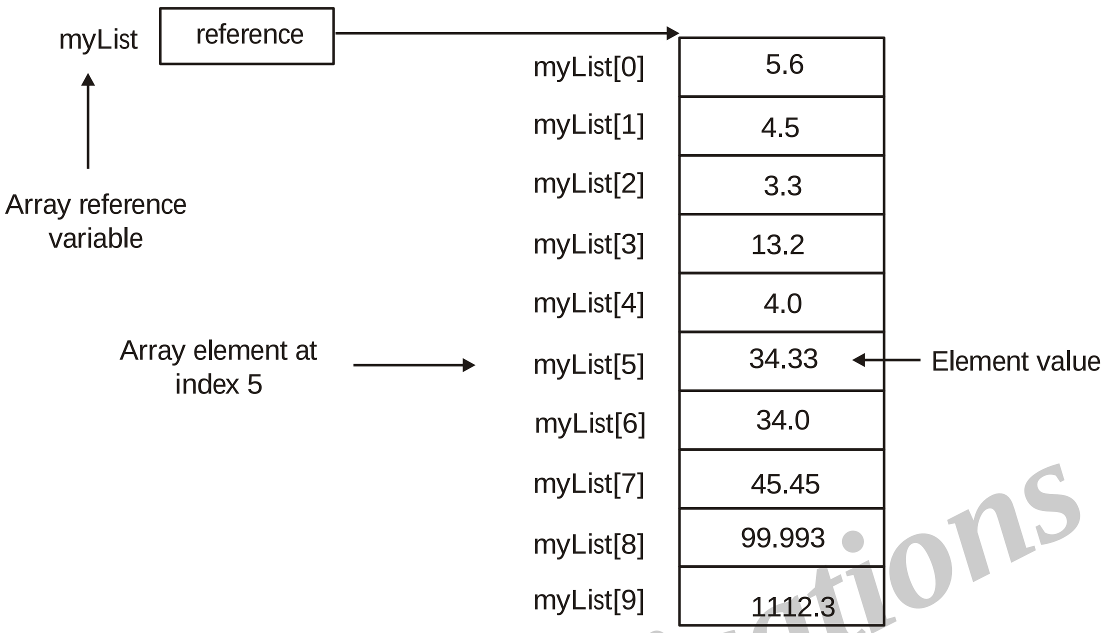
The array elements are accessed through the index. Array indices are 0-based; that is, they start from 0 to arrayRefVar.length-1.

Example

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList “

```
double[] myList=newdouble[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays

When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

Example

Here is a complete example showing how to create, initialize, and process arrays –

```
public class TestArray
{
        public static void main(String[] args)
        {
              double[] myList={1.9,2.9,3.4,3.5};
              // Print all the array elements
              for(inti=0;i<myList.length; i++)
              {
                 System.out.println(myList[i]+"");
              }
              // Summing all elements
              double total =0;
              for(inti=0;i<myList.length; i++)
              {
                    total +=myList[i];
              }
```

```
            System.out.println("Total is "+ total);
            // Finding the largest element
            double max =myList[0];
            for(inti=1;i<myList.length; i++)
            {
                  if(myList[i]> max)
                  max =myList[i];
            }
            System.out.println("Max is "+ max);
      }
}
```

Output

```
1.9
2.9
3.4
3.5
Total is 11.7
Max is 3.5
```

2.5.1 One-Dimensional Arrays

Q13. Discuss briefly about One-Dimensional Arrays.

Ans :

One Dimensional Array in java is always used with only one subscript([]). A one-dimensional array behaves likes a list of variables. You can access the variables of an array by using an index in square brackets preceded by the name of that array. Index value should be an integer.

Declaration of One-dimensional array in java

Before using the array, we must declare it. Like normal variables, we must provide data type of array and name of an array. Data type means which type of elements we want to store in Array. So, we must specify the data type of array according to our needs. We also need to specify the name of an array so that we can use it later by name.

```
datatype[] arrayName;
Or
datatype arrayName[];
Or
datatype []arrayName;
```

- datatype can be a primitive data type(int, char, Double, byte etc.)or Non-primitive data (Objects).
- arrayName is the name of an array
- [] is called subscript.

```
int[] number;
Or
int number[];
Or
int []number;
```

Construction of One-dimensional array in java

For the creation of an array new keyword is used with a data type of array. You must specify the size of the array. The size should be an integer value or a variable that contains an integer value. How many elements you can store an array directly depends upon the size of an array.

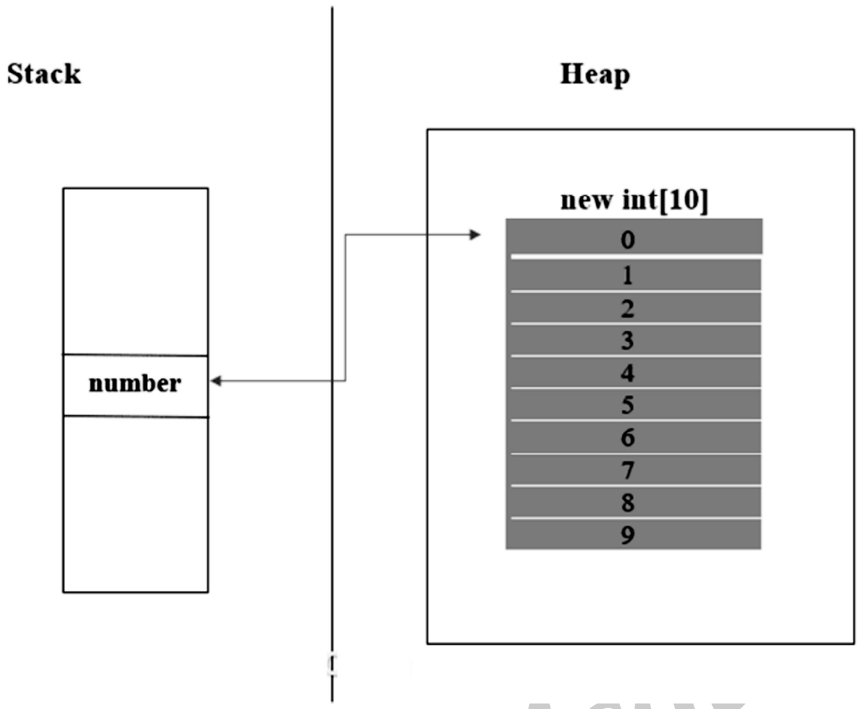
```
arrayName = new DataType[size];
```

new Datatype[size]: It creates an array in heap memory. Because an array is an object, so it stored in heap memory.

arrayName: Assignment operator assigns the newly created array to the reference of variable arrayName. We can access the elements of an array by use of array names.

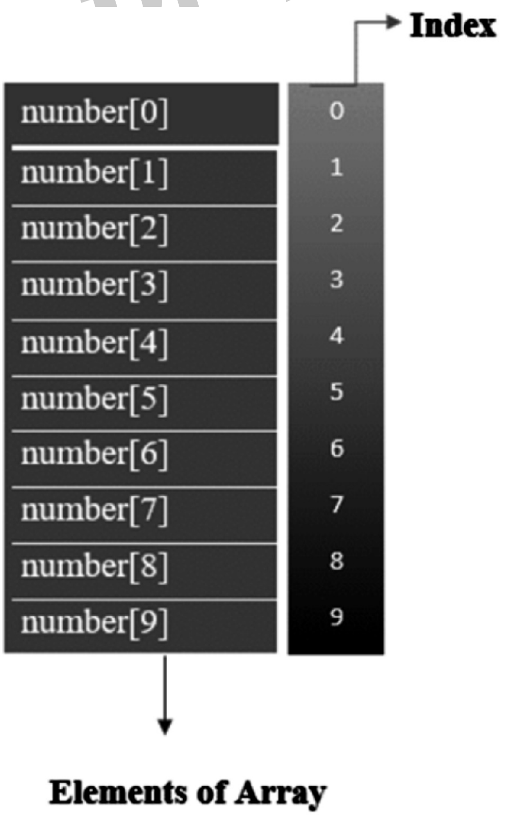
number = new int[10]; // allocating memory to array

In this example new int[10] creates a new array of int type in heap memory. The assignment operator assigns the array object to the reference variable number. Now you can access the array by use of the number. You can access each value of array by using numbers with subscript([]). We must have to use only one subscript([]) for One dimensional array in java We will discuss it later in the Access of array.



Note: When we are creating a new array the elements in the array will automatically be initialized by their default values. For e.g. : zero (int types), false (boolean), or null (for object types).

Memory Representation after Construction



The index of an array starts from 0 and ends with length-1. The first element of an array is number[0], second is number[1] and so on. If the length of an array is n, the last element will be arrayName[length-1]. Since the length of the number array is 10, the last element of the array is number[9].

Initialization of One-dimensional array in java

To initialize the Array we have to put the values at each index of array. In above section we have created a Array with size 10. So now we will see how we can add values in Array.

```
public class MyExample
{
public static void main(String[] args)
{
// Declaration of Array
int[] number;
// Construction of Array with given size
// Here we are giving size 10 it mean it can hold 10 values of int type
number = new int[10];
// Initialization of Array
number[0] = 11;
number[1] = 22;
number[2] = 33;
number[3] = 44;
number[4] = 55;
number[5] = 66;
number[6] = 77;
number[7] = 88;
number[8] = 99;
number[9] = 100;
//Print the values from Array
for(int i = 0; i < number.length; i++)
System.out.println(number[i]);
}
}
```

Output:

```
11
22
33
44
55
66
77
88
99
100
```

2.5.2 Two-Dimensional Arrays

Q14. Discuss briefly about Two-Dimensional Arrays.

Ans :

In Java, a two-dimensional array is stored in the form of rows and columns and is represented in the form of a matrix.

1. Declaring 2 Dimensional Array

Syntax: there are two forms of declaring an array.

Type arrayname[];

Or

type[] array name;

Look at the following examples

Example

int name[][];

or

int[][] name;

2. Creating an Object of a 2d Array

Now, it's time to create the object of a 2d array.

name = new int[3][3]

Creating a 2-dimensional object with 3 rows and 3 columns.

3. Initializing 2d Array

After creating an array object, it is time to initialize it.

In the following code, we describe how to initialize the 2-dimensional array.

Int name[3][3] = {"a","b","c","a1","b1","c1","a2","b2","c2"};

OR

int name[3][3] = {{"a","b","c"},

{"a1","b1","c1"},

{"a2","b2","c2"}};

Create 2D Arrays in Java

We will look at how to create 2 dimensional with the help of an example. Before that, let us look, we have two index values for a 2d array. One is for a row, and another is for the column.

Row Size

Rows are the elements in an array that can store horizontally. For example, Row Size is equal to 4, then the array will create with 4 rows.

Column Size

Columns are the elements in an array that can store vertically. For example, Column Size is equal to 2, then an array that can have 2 Columns in it.

```
public class TwoDArray{
public static void main(String[] args) {
int[][] twoDimentional = {{1,1},{2,2},{3,3},{4,4}};
for(int i = 0 ; i < 4 ; i++){
for(int j = 0 ; j < 2; j++){
System.out.print(twoDimentional[i][j] + " ");
}
System.out.println();
}
}
}
```

Output:

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

2.6 Command-Line Arguments

Q15. Explain about Command-Line Arguments. How they are useful.

Ans : **(June-19, MGU)**

Command Line Argument is information passed to the program when you run the program. The passed information is stored as a string array in the main method.

Important Points

- Command Line Arguments can be used to specify configuration information while launching your application.
- There is no restriction on the number of java command line arguments.You can specify any number of arguments
- Information is passed as Strings.
- They are captured into the String args of your main method

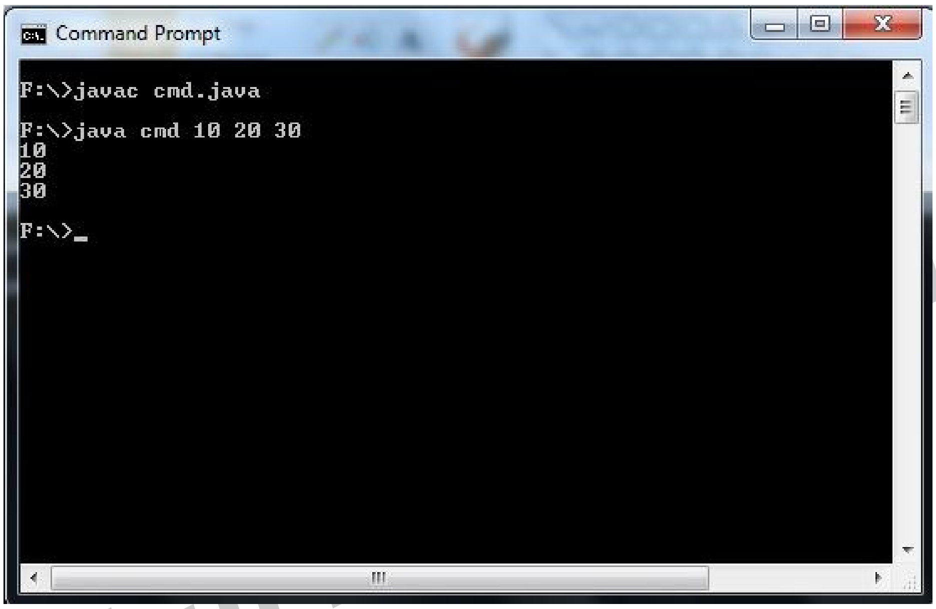
To Learn java Command Line Arguments

Step 1: Copy the following code into an editor.

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

```
		for(int i=0;i<args.length;i++)
		{
			System.out.println(args[i]);
		}
	}
}
```

Step 2: Save & Compile the code



```
F:\>javac cmd.java

F:\>java cmd 10 20 30
10
20
30

F:\>_
```

Step 3: Run the code as **java Demo apple orange.**

Step 4: You must get an output as below.

Output

```
10
20
30
```

2.6.1 Inner Class

Q16. What is Inner Class? Explain.

Ans :

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax

```
class Java_Outer_class
```

```
{
//code
class Java_Inner_class
{
        //code
}
}
```

There are basically three advantages of inner classes in java. They are as follows:

1. Nested classes represent a special type of relationship that is it can access all the members (data members and methods) of outer class including private.
2. Nested classes are used to develop more readable and maintainable code because it logically group classes and interfaces in one place only.
3. Code Optimization: It requires less code to write.

Example

```
class TestMemberOuter1
{
        private int data=30;
        class Inner
        {
                void msg(){System.out.println("data is"+data);}
        }
        public static void main(String args[])
        {
                TestMemberOuter1 obj=new TestMemberOuter1();
                TestMemberOuter1.Inner in=obj.new Inner();
                in.msg();
        }
}
```

2.7 Inheritance

2.7.1 Introduction

Q17. What is Inheritance?

(OR)

Explain the concept of Inheritance.

Ans :

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. It is an important part of OPPs(Object Oriented programming system).

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.Inheritance represents the IS-A relationship, also known as parent-child relationship.

Important terminology

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, 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. By doing this, we are reusing the fields and methods of the existing class.

Syntax of Java Inheritance

```
class Super
{
  .....
  .....
}
class Sub extends Super
{
  .....
  .....
}
```

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of “extends” is to increase the functionality.

In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

```
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(Stringargs[])
        {
             int a =20, b =10;
             My_Calculation demo =newMy_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
```

Q18. Explain the benefits of Inheritance.

Ans :

- **Code sharing:** Code sharing occurs at two levels. At first level many users or projects can use the same class. In the second level sharing occurs when two or more classes developed by a single programmer as part of a project which is being inherited from a single parent class. Here also code is written once and reused. This is possible through inheritance.

- **Consistency of inheritance:** When two or more classes inherit from the same superclass we are assured that the behaviour they inherit will be the same in all cases. Thus we can guarantee that interfaces to similar objects are in fact similar.
- **Software components:** Inheritance provides programmers the ability to construct reusable s/w components. The goal behind these is to provide applications that require little or no actual coding. Already such libraries and packages are commercially available.
- **Rapid Prototyping:** When a s/w system is constructed largely out of reusable components development can be concentrated on understanding the new and unusual portion of the system. Thus s/w system can be generated more quickly and easily leading to a style of programming known as 'Rapid Prototyping' or 'exploratory programming'
- **Polymorphism and Framework:** Generally s/w is written from the bottom up , although it may be designed from top-down. This is like building a wall where every brick must be laid on top of another brick i.e., the lower level routines are written and on top of these slightly higher abstraction are produced and at last more abstract elements are generated. Polymorphism permits the programmer to generate high level reusable components.
- **Information Hiding:** A Programmer who reuses a software need only to understand the nature of the component and its interface. There is no need to have detailed information of the component and it is information hiding.

2.7.2 Types of Inheritance

Q19. What are the different types of Inheritance?

Ans : **(July-21, July-19, Dec.-18-MGU)**

There are five types of inheritance in Java They are :

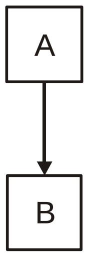
1. Single level inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

2.7.2.1 Single level inheritance

Q20. What is Single Inheritance in JAVA? Explain with an Example.

Ans :

Single inheritance is easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram 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.



Single Inheritance

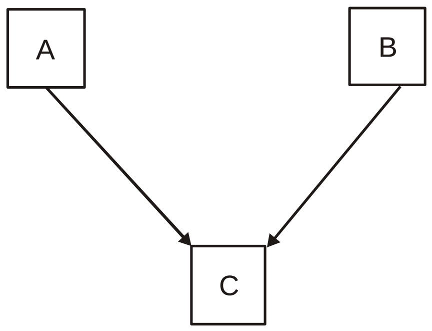
```
Class A
{
        public void methodA()
        {
                System.out.println("Base class method");
        }
}
Class B extends A
{
        public void methodB()
        {
                System.out.println("Child class method");
        }
        public static void main(Stringargs[])
        {
                B obj=new B();
                obj.methodA();//calling super class method
                obj.methodB();//calling local method
        }
}
```

2.7.2.2 Multiple inheritance

Q21. What is multiple Inheritance? How is multiple inheritance achieved in java?

Ans : **(Dec.-18)**

"Multiple Inheritance" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



Multiple Inheritance

Note:

i. Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

ii. Most of the new OO languages like Small Talk, Java, C# do not support Multiple inheritance. Multiple Inheritance is supported in C++.

Multiple Inheritance (Through Interfaces)

In Multiple inheritance ,one class can have more than one superclass and inherit features from all parent classes. Please note that Java does **not** support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.

```
// First Parent class
class Parent1
{
        void fun()
        {
                System.out.println("Parent1");
        }
}
// Second Parent Class
class Parent2
{
        void fun()
        {
            System.out.println("Parent2");
        }
}
// Error : Test is inheriting from multipleclasses
classTest extendsParent1, Parent2
{
        publicstaticvoidmain(String args[])
        {
                Test t = newTest();
                t.fun();
        }
}
```

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

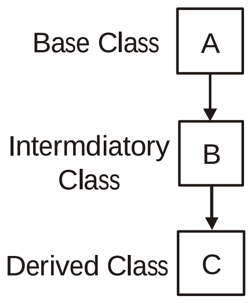
```
class A
{
        void msg()
        {
                System.out.println("Hello");
        }
    }
class B
{
void msg()
{
System.out.println("Welcome");
}
}
class C extends A,B
{
        //suppose if it were
        public static void main(String args[])
{
        C obj=new C();
        obj.msg();//Now which msg() method would
        be invoked?
        }
}
```

2.7.2.3 Multilevel inheritance

Q22. Explain the concept of multilevel Inheritance with example?

Ans :

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



Multilevel Inheriatance

Example Program

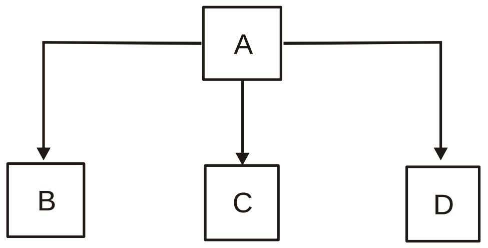
```
Class X
{
        public void methodX()
        {
                System.out.println("Class X method");
        }
}
Class Y extends X
{
        public void methodY()
        {
                System.out.println("class Y method");
        }
}
Class Z extends Y
{
        public void methodZ()
        {
                System.out.println("class Z method");
        }
        public static void main(Stringargs[])
        {
                Z obj=new Z();
                obj.methodX(); //calling grand parent
                                //class method
                obj.methodY();//calling parent class
                                //method
                obj.methodZ();//calling local method
        }
}
```

2.7.2.4 Hierarchical inheritance

Q23. What is Hierarchical Inheritance? Explain.

Ans :

In such kind of inheritance one class is inherited by many sub classes. In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D.



Hierarchical Inheritance

Example Program

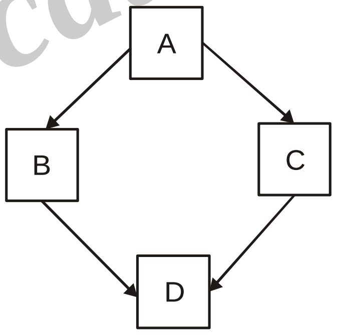
```
class A
{
        public void methodA()
        {
                System.out.println("method of
                Class A");
        }
}
class B extends A
{
        public void methodB()
        {
                System.out.println("method of
                Class B");
        }
}
class C extends A
{
        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 JavaExample
{
        public static void main(Stringargs[])
        {
                B obj1 =new B();
                C obj2 =new C();
                D obj3 =new D();
                //All classes can access the method of
                //class A
                obj1.methodA();
                obj2.methodA();
                obj3.methodA();
        }
}
```

2.7.2.5 Hybrid inheritance

Q24. Discuss about Hybrid Inheritance using Interfaces?

Ans :

It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritances with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



Hybrid Inheritance

Example Program

```
class Animal
{
        void eat(){System.out.println("eating...");}
}
class Dog extends Animal
{
        void bark(){System.out.println
        ("barking...");}
}
class Cat extends Animal
{
        void meow(){System.out.println("meowing
                                              ...");}
}
 class TestInheritance3
{
public static void main(String args[])
{
Cat c=new Cat();
               c.meow();
               c.eat();
               //c.bark();//C.T.Error
        }
}
```

2.7.3 Extends Keyword

Q25. Explain the concept of extends keyword.

Ans :

The most important and widely used keywords in Java which is the extends keyword. It is a reserved word that and we can't use it as an identifier other than the places where we need to inherit the classes in Java. We use the extends keyword in Inheritance in Java.

Inheritance is one of the object-oriented concepts which defines the ability of a class to extend or acquire the properties of another class. The extends keyword plays a significant role in implementing the Inheritance concept in Java. Let's start discussing the extends keyword with examples.

The extends keyword in Java indicates that the child class inherits or acquires all the properties of the parent class. This keyword basically establishes a relationship of an inheritance among classes.

If a class extends another class, then we say that it has acquired all the properties and behavior of the parent class.

We use the extends keyword in Java between two class names that we want to connect in the Inheritance relationship.

The class that extends the properties is the child class or the derived class which comes before the extends keyword, while the class from which the properties are inherited is the parent class or super class or the base class, and this class comes after the extends keyword.

It is not possible to extend multiple classes in Java because there is no support for multiple inheritances in Java. And therefore we cannot write multiple class names after the extended keyword.

But, multiple classes can inherit from a single class as java supports hierarchical inheritance. Therefore we can write multiple class names before the extended keyword.

Syntax of extends keyword in Java

Following is the syntax of using extends keyword in java when using inheritance:

```
class Parent {
//code inside the parent class
}
class Child extends Parent {
//Code inside the child class
}
```

The two important categories are:

(a) **Parent class:** This is the class being inherited. Also called super class or base class.

(b) **Child class:** This class inherits the properties from the parent class. Also called a subclass or derived class.

2.7.4 Method Overriding

Q26. Explain the method overriding with example?

Ans :

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

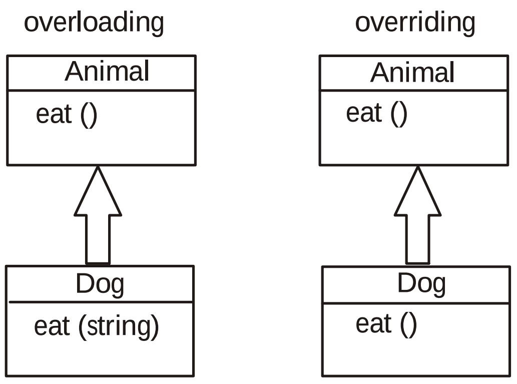
Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules

- The argument list should be exactly the same as that of the overridden method.
- The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
- The access level cannot be more restrictive than the overridden method's access level. For example: If the superclass method is declared public then the overridding method in the sub class cannot be either private or protected.
- Instance methods can be overridden only if they are inherited by the subclass.
- A method declared final cannot be overridden.
- A method declared static cannot be overridden but can be re-declared.
- If a method cannot be inherited, then it cannot be overridden.
- A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
- A subclass in a different package can only override the non-final methods declared public or protected.
- An overriding method can throw any uncheck exceptions, regardless of whether the overridden method throws exceptions or not. However, the overriding method should not throw checked exceptions that are new or broader than the ones declared by the overridden method. The overriding method can throw narrower or fewer exceptions than the overridden method.
- Constructors cannot be overridden.

The benefit of overriding is: ability to define a behavior that's specific to the subclass type, which means a subclass can implement a parent class method based on its requirement.



Example

```
class Animal
{
      Animal myType()
      {
            return new Animal();
      }
}
class Dog extends Animal
{
      Dog myType()  //Legal override after Java5
                     //onward
      {
            return new Dog();
      }
}
```

Method overriding is one of the way by which java achieve Run Time Polymorphism. The version of a method that is executed will be determined by the object that is used to invoke it. If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

Example

```
class Parent
{
void show()
{
System.out.println("Parent's show()");
}
}
class Child extends Parent// Inherited class
{
        // This method overrides show() of Parent @Override
        void show()
{
System.out.println("Child's show()");
}
}
// Driver class
class Main
{
        public static void main(String[] args)
        {
                // If a Parent type reference refers
                // to a Parent object, then Parent's
                // show is called
                Parent obj1 = newParent();
                obj1.show();
                // If a Parent type reference refers
                // to a Child object Child's show()
                // is called. This is called RUN TIME
                // POLYMORPHISM.
                Parent obj2 = newChild();
                obj2.show();
        }
}
```

2.7.5 Super Final Keyword

Q27. What is the use of super keyword in Java programming?

Ans :

Super keyword in java is a reference variable that is used to refer parent class object. Super is an implicit keyword create by JVM and supply each and every java program for performing important role in three places.

(i) Super keyword At Variable Level

(ii) Super keyword At Method Level

(iii) Super keyword At Constructor Level

(i) Use of super keyword with variables

This scenario occurs when a derived class and base class has same data members. In that case there is a possibility of ambiguity for the JVM. We can understand it more clearly using this code snippet:

```
/* Base class vehicle */
class Vehicle
{
int maxSpeed = 120;
}
/* sub class Car extending vehicle */
class Car extends Vehicle
{
int maxSpeed = 180;
void display()
        {
        /* print maxSpeed of base class (vehicle) */
        System.out.println("Maximum Speed: "+ super.maxSpeed);
        }
}
/* Driver program to test */
class Test
{
        public static void main(String[] args)
        {
                Car small = new Car();
                small.display();
        }
}
```

(ii) Use of super keyword with methods

This is used when we want to call parent class method. So whenever a parent and child class have same named methods then to resolve ambiguity we use super keyword.

```
/* Base class Person */
class Person
{
    void message()
    {
        System.out.println("This is person class");
    }
}
/* Subclass Student */
class Student extends Person
{
    void message()
    {
        System.out.println("This is student class");
    }
    // Note that display() is only in Student class
    void display()
    {
    // will invoke or call current class message() method
        message();
    // will invoke or call parent class message() method
        super.message();
    }
}
/* Driver program to test */
class Test
{
    public static void main(String args[])
    {
            Student s = newStudent();
            // calling display() of Student
```

```
                s.display();
        }
}
```

(iii) Use of super with constructors

Super keyword can also be used to access the parent class constructor. One more important thing is that, ''super' can call both parametric as well as non parametric constructors depending upon the situation. Following is the code snippet to explain the above concept:

```
/* superclass Person */
class Person
{
        Person()
        {
        System.out.println("Person class Constructor");
        }
}
/* subclass Student extending the Person class */
class Student extends Person
{
        Student()
        {
                // invoke or call parent class constructor
                super();
                System.out.println("Student class Constructor");
        }
}
/* Driver program to test*/
class Test
{
        public static void main(String[] args)
        {
                Student s = new Student();
        }
}
```

Important points

1. Call to super() must be first statement in Derived(Student) Class constructor.

2. If a constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass. If the superclass does not have a no-argument constructor, you will get a compile-time error. Object does have such a constructor, so if Object is the only superclass, there is no problem.

3. If a subclass constructor invokes a constructor of its superclass, either explicitly or implicitly, you might think that a whole chain of constructors called, all the way back to the constructor of Object. This, in fact, is the case. It is called *constructor chaining*.

Q28. Mention the use of Final keyword in java at different places?

Ans :

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. First of all, final is a non-access modifier applicable only to a variable, a method or a class.Following are different contexts where final is used.

1. Final variable
2. Final method
3. Final class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these.

1. Final variables

When a variable is declared with *final* keyword, it's value can't be modified,essentially, a constant. This also mean that you must initialize a final variable. If the final variable is a reference, this means that the variable cannot be re-bound to reference another object, but internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from final array or final collection. It is good practice to represent final variables in all uppercase, using underscore to separate words.

Initializing a final variable

We must initialize a final variable, otherwise compiler will throw compile-time error.A final variable can only be initialized once, either via an initializer or an assignment statement. There are three ways to initialize a final variable:

(i) You can initialize a final variable when it is declared.This approach is the most common. A final variable is called blank final variable,if it is not initialized while declaration. Below are the two ways to initialize a blank final variable.

(ii) A blank final variable can be initialized inside instance-initializer block or inside constructor. If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown.

(iii) A blank final static variable can be initialized inside static block.

```
class Bike9
{
    final int speedlimit=90;//final variable
    void run()
    {
        speedlimit=400;
    }
    public static void main(String args[])
    {
        Bike9 obj=new  Bike9();
        obj.run();
    }
  }
```

2. Final methods

When a method is declared with *final* keyword, it is called a final method. A final method cannot be overridden. The Object class does this—a number of its methods are final.We must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes. The following fragment illustrates final keyword with a method:

```
class Bike
        {
              final void run(){System.out.println("running");}
        }
        class Honda extends Bike
        {
        void run()
        {
        System.out.println("running safely with 100kmph");
        }
        public static void main(String args[])
        {
        Honda honda= new Honda();
              honda.run();
        }
        }
```

3. Final classes

When a class is declared with final keyword, it is called a final class. A final class cannot be extended(inherited). There are two uses of a final class :

(i) One is definitely to prevent inheritance, as final classes cannot be extended. For example, all Wrapper Classes like Integer,Float etc. are final classes. We can not extend them.

(ii) The other use of final with classes is to create an immutable class like the predefined Stringclass.Youcan not make a class immutable without making it final.

```
final class Bike
{
}
class Honda1 extends Bike
{
void run()
{
System.out.println("running safely with 100kmph");
}
public static void main(String args[])
{
Honda1 honda= new Honda1();
honda.run();
}
}
```

2.8 Abstract Classes

Q29. What is an abstract class and method? What is the need of abstract class?

Ans : **(July-21)**

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

i. Abstract class (0 to 100%)

ii. Interface (100%)

Abstract class

If a class contain any abstract method then the class is declared as abstract class. An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

Syntax

abstract class class_name { }

Abstract method

Method that are declared without any body within an abstract class are called abstract method. The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

Syntax

abstract return_typefunction_name (); // No definition

Example of Abstract class

```
abstract class A
{
      abstract void callme();
}
class B extends A
{
      void callme()
      {
            System.out.println("this is callme.");
      }
      public static void main(String[]args)
      {
            B b=newB();
            b.callme();
      }
}
```

Abstract class with concrete (normal) method

Abstract classes can also have normal methods with definitions, along with abstract methods.

```
abstract class A
{
      abstract void callme();
      public void normal()
      {
            System.out.println("this is concrete
            method");
      }
}
class B extends A
{
      void callme()
      {
            System.out.println("this is callme.");
      }
      public static void main(String[] args)
      {
        B b = new B();
        b.callme();
        b.normal();
    }
}
```

Points to Remember

1. Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
2. An abstract class may or may not have an abstract method. But if any class has even a single abstract method, then it must be declared abstract.
3. Abstract classes can have Constructors, Member variables and Normal methods.
4. Abstract classes are never instantiated.
5. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

Use of Abstract Methods & Abstract Class

Abstract methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. These subclasses extend the same Abstract class and provide different implementations for the abstract methods.

Abstract classes are used to define generic types of behaviors at the top of an object-oriented programming class hierarchy, and use its subclasses to provide implementation details of the abstract class.

2.8.1 Interfaces

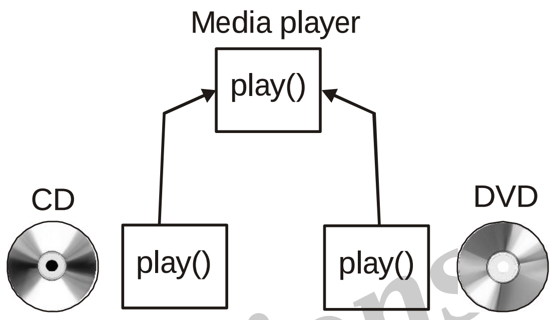
Q30. Define Interface? Explain the declaration and implementation of Interface.

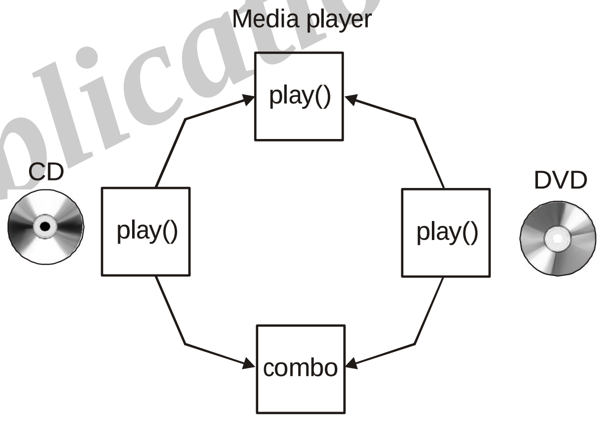
Ans : **(July-21)**

An interface in java is a blueprint of a class. It has static constants and abstract methods.

The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have methods and variables but the methods declared in interface contain only method signature, not body.





An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Java Interface also **represents IS-A relationship**.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways -

- An interface can contain any number of methods.
- An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a .class file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including-

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

Declaring Interfaces

The interface keyword is used to declare an interface. Here is a simple example to declare an interface-

Example

Following is an example of an interface-

```
/* File name : NameOfInterface.java */
importjava.lang.*;
// Any number of import statements
publicinterfaceNameOfInterface{
// Any number of final, static fields
// Any number of abstract method
  declarations\
}
```

Interfaces have the following properties-

- An interface is implicitly abstract. You do not need to use the abstract keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

Example

```
/* File name : Animal.java */
interface Animal
{
        public void eat();
        public void travel();
}
```

Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
/* File name : MammalInt.java */
public class MammalIntimplements Animal
{
public void eat()
{
            System.out.println("Mammal eats");
        }
public void travel()
{
```

```
            System.out.println("Mammal travels");
        }
public int noOfLegs()
{
return0;
}
public static void main(Strin gargs[])
{
MammalInt m =new MammalInt();
        m.eat();
        m.travel();
}
}
```

Output

```
Mammal eats
Mammal travels
```

When overriding methods defined in interfaces, there are several rules to be followed-

➢ Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.

➢ The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.

➢ An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementation interfaces, there are several rules –

➢ A class can implement more than one interface at a time.

➢ A class can extend only one class, but implement many interfaces.

➢ An interface can extend another interface, in a similar way as a class can extend another class.

Extending Interfaces

An interface can extend another interface in the same 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.

The following Sports interface is extended by Hockey and Football interfaces.

Example

```
// Filename: Sports.java
public interface Sports
{
        public void setHomeTeam(String name);
        public void set VisitingTeam(String name);
}
// Filename: Football.java
public interface Football extends Sports
{
        public void homeTeamScored(int points);
        public void visitingTeamScored(int points);
        public void endOfQuarter(int quarter);
}
// Filename: Hockey.java
publicinterfaceHockeyextendsSports
{
        public void homeGoalScored();
        public void visitingGoalScored();
        public void endOfPeriod(int period);
        public void overtimePeriod(intot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

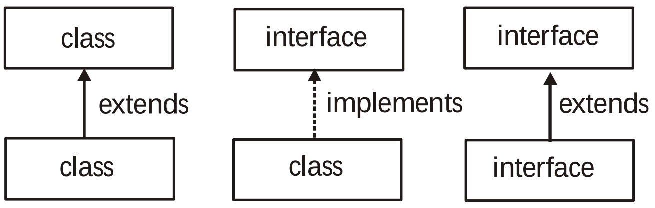
Extending Multiple Interfaces

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.

Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.



Important points about interface or summary of article:

- We can't create instance(interface can't be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
- A class can implement more than one interface.
- An interface can extends another interface or interfaces (more than one interface) .
- A class that implements interface must implements all the methods in interface.
- All the methods are public and abstract. And all the fields are public, static, and final.
- It is used to achieve multiple inheritance.
- It is used to achieve loose coupling.

Q31. Write a program to implement Interfaces?

Ans :

```
import java.io.*;
interface Vehicle
{
        // all are the abstract methods.
        voidc hangeGear(in ta);
        void speedUp(in ta);
        voidapplyBrakes(inta);
}
class Bicycle implements Vehicle
{
        int speed;
        int gear;
              // to change gear@Override
        public void changeGear(int new Gear)
        {
                    gear = new Gear;
        }
        // to increase speed@Override
        public void speedUp(int increment)
        {
        speed = speed + increment;
        }
// to decrease speed @Override
        public void applyBrakes(int decrement)
        {
                          speed = speed - decrement;
        }
        public void printStates()
{
                          System.out.println("speed: "+ speed+ " gear: "+ gear);
        }
}
class Bike implements Vehicle
{
        int speed;
        int gear;
        public void change Gear(intnewGear)// to change gear  @Override
{
                          gear = newGear;
        }
        public void speedUp(int increment)// to increase speed   @Override
        {
                          speed = speed + increment;
        }
              // to decrease speed@Override
```

```
public void applyBrakes(int decrement)
 {
                        speed = speed - decrement;
      }
            public void printStates()
{
                        System.out.println("speed: "+ speed  + " gear: "+ gear);
      }
}
class GFG
{
public static void main (String[] args)
{
          // creating an inatance of Bicycle doing some operations
            Bicycle bicycle = newBicycle();
            bicycle.changeGear(2);
            bicycle.speedUp(3);
            bicycle.applyBrakes(1);
            System.out.println("Bicycle present state :");
            bicycle.printStates();
            // creating instance of bike.
            Bike bike = newBike();
            bike.changeGear(1);
            bike.speedUp(4);
            bike.applyBrakes(3);
            System.out.println("Bike present state :");
            bike.printStates();
}
}
```

Output

```
Bicycle present state :
speed: 2 gear: 2
Bike present state :
speed: 1 gear: 1
```

2.8.2 Abstract Classes Verses Interfaces

Q32. Differentiate the use of Abstract class and Interface?

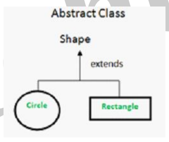
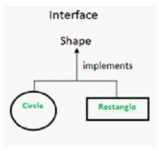
Ans : **(June-19-MGU, Dec.-18-KU)**

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

Abstract class	Interface
Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
An **abstract class**can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
An **abstract class**can be extended using keyword ?extends?.	An **interface class**can be implemented using keyword ?implements?.
A Java**abstractclass**can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape
{
public abstract void draw();
}
	Example:
	public interface Drawable
	{
	void draw();
	}

2.9 PACKAGES

Q33. What is a package in java?

(OR)

Define package.

Ans : **(July-19)**

Package 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 or sub-package. Using package it becomes easier to locate the related classes and it also provides a good structure for projects with hundreds of classes and other files.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee

- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

Types of Packages: Built-in and User defined

- **Built-in Package:** Existing Java package for example java.lang, java.util etc.
- **User-defined-Package:** Java package created by user to categorize their project's classes and interface.

Advantage of Java Package

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides access protection.
3. Java package removes naming collision.

2.9.1 Creating and Using Packages

Q34. How you can create a package?

Ans :

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

```
package mypack;
publicclassemployee
{
statement;
}
```

The above statement will create a package woth name mypack in the project directory. Java uses file system directories to store packages. For example the .java file for any class you define to be part of mypack package must be stored in a directory called mypack.

Additional points about package:

- A package is always defined as a separate folder having the same name as the package name.
- Store all the classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that they are error free)

How to compile Java programs inside packages?

This is just like compiling a normal java program. If you are not using any IDE, you need to follow the steps given below to successfully compile your packages:

```
javac-d directory javafilename
```

Example

```
javac-d . FirstProgram.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

You need to use fully qualified name e.g. learnjava.FirstProgram etc to run the class.

To Compile:

```
javac-d . FirstProgram.java
```

To Run:

```
java learnjava.FirstProgram
```

Import Keyword

Import keyword is used to import built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.

There are 3 different ways to refer to any class that is present in a different package:

Using fully qualified name (But this is not a good practice.)

If you use fully qualified name to import any class into your program, then only that particular class of the package will be accessible in your program, other classes in the same package will not be accessible. For this approach, there is no need to use the import statement. But you will have to use the fully qualified name every time you are accessing the class or the interface, which can look a little untidy if the package name is long.

This is generally used when two packages have classes with same names. For example: java.util and java.sql packages contain Date class.

Example :

```
//save by A.java
package pack;
public class A
{
        public void msg()
        {
                System.out.println("Hello");
        }
}
//save by B.java
package mypack;
class B
{
        public static void main(String args[])
        {
                pack.A obj = new pack.A();
                obj.msg();
        }
}
```

2.10 Access Protection

Q35. What are the restrictions imposed on java packages?

(OR)

Discuss the various levels of access protection available.

Ans :

Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors. The four access levels are,

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Default Access Modifier - No Keyword

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.

Access Protection in Packages

Access modifiers define the scope of the class and its members (data and methods). For example, private members are accessible within the same class members (methods). Java provides many levels of security that provides the visibility of members (variables and methods) within the classes, subclasses, and packages.

Packages are meant for encapsulating, it works as containers for classes and other subpackages. Class acts as containers for data and methods. There are four categories, provided by Java regarding the visibility of the class members between classes and packages:

1. Subclasses in the same package
2. Non-subclasses in the same package
3. Subclasses in different packages
4. Classes that are neither in the same package nor subclasses

The three main access modifiers *private, public* and *protected* provides a range of ways to access required by these categories.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	Yes	Yes	Yes	Yes
Same package non-subclass	Yes	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Simply remember, private cannot be seen outside of its class, public can be access from anywhere, and protected can be accessible in subclass only in the hierarchy.

A class can have only two access modifier, one is default and another is public. If the class has default access then it can only be accessed within the same package by any other code. But if the class has public access then it can be access from anywhere by any other code.

Example:

```
//PCKG1_ClassOne.java
package pckg1;
public class PCKG1_ClassOne
{
    int a = 1;
    private int pri_a = 2;
    protected int pro_a = 3;
    public int pub_a = 4;
    public PCKG1_ClassOne()
    {
        System.out.println("base class constructor called");
        System.out.println("a = " + a);
        System.out.println("pri_a = " + pri_a);
        System.out.println("pro_a "+ pro_a);
        System.out.println("pub_a "+ pub_a);
    }
}
```

The above file PCKG1_ClassOne belongs to package pckg1, and contains data members with all access modifiers.

```
//PCKG1_ClassTwo.java
package pckg1;
```

```
class PCKG1_ClassTwo extends PCKG1_ClassOne
{
        PCKG1_ClassTwo()
        {
              System.out.println("derived class constructor called");
              System.out.println("a = " + a);
              //accessible in same class only
              //System.out.println("pri_a = " + pri_a);
              System.out.println("pro_a "+ pro_a);
              System.out.println("pub_a =" + pub_a);
        }
}
```

2.11 Wrapper Classes in Java

Q36. What are wrapper class? What is its use in JAVA?

Ans : **(Dec.-19, Dec.-18)**

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Importance

There are mainly two uses with wrapper classes.

1. To convert simple data types into objects, that is, to give object form to a data type; here constructors are used.
2. To convert strings into data types (known as parsing operations), here methods of type parseXXX() are used.

Features

1. Wrapper classes convert numeric strings into numeric values.
2. The way to store primitive data in an object.
3. The valueOf() method is available in all wrapper classes except Character
4. All wrapper classes have typeValue() method. This method returns the value of the object as its primitive type.

Primitive Data types and their Corresponding Wrapper class

Primitive type	Wrapper class	Constructor Arguments
byte	Byte	byte or String
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	float, double or String
double	Double	double or String
char	Character	char
boolean	Boolean	boolean or String

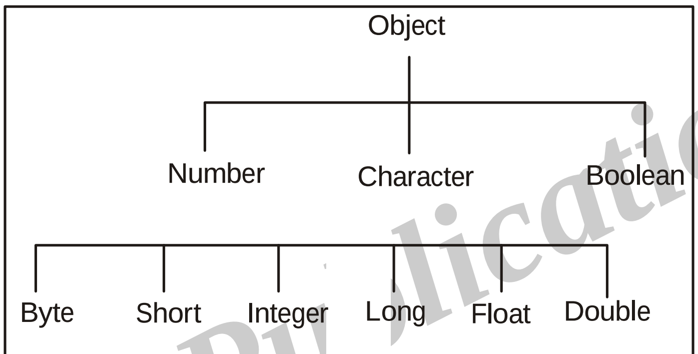


Fig.: Wrapper classes Hierarchy

Example

```
import java.util.ArrayList;
class Autoboxing
{
        public static void main(String[] args)
        {
                char   ch = 'a';
                // Autoboxing- primitive to Character object conversion
                Character a = 'ch';
                ArrayList<Integer> arrayList = new ArrayList<Integer>();
                // Autoboxing because ArrayList stores only objects
                arrayList.add(25);
                // printing the values from object
                System.out.println(arrayList.get(0));
        }
}
```

Output:

```
25
```

Q37. Write a java program to implement Wrapper Class in Java?

Ans : **(Dec.-19)**

```
class WrappingUnwrapping
{
        public static void main(String args[])
       {
              //   byte data type
              byte a = 1;
              // wrapping around Byte object
              Byte byte obj = new Byte(a);
              // int data type
              intb = 10;
              //wrapping around Integer object
              Integer intobj = newInteger(b);
              // float data type
              floatc = 18.6f;
              // wrapping around Float object
              Float float obj = new Float(c);
              // double data type
              doubled = 250.5;
              // Wrapping around Double object
              Double double obj = new Double(d);
              // char data type
                char e='a';
              // wrapping around Character object
              Character char obj=e;
              //   printing the values from objects
              System.out.println("Values of Wrapper objects (printing as objects)");
              System.out.println("Byte object byteobj:   "+ byteobj);
              System.out.println("Integer object intobj:   "+ intobj);
              System.out.println("Float object floatobj:   "+ floatobj);
              System.out.println("Double object doubleobj:   "+ doubleobj);
              System.out.println("Character object charobj:   "+ charobj);
              // objects to data types (retrieving data types from objects)
```

```
            // unwrapping objects to primitive data types
            byte bv = byte obj;
            int iv = int obj;
            float fv = float obj;
            double dv = double obj;
            char cv = charobj;
            // printing the values from data types
            System.out.println("Unwrapped values (printing as data types)");
            System.out.println("byte value, bv: "+ bv);
            System.out.println("int value, iv: "+ iv);
            System.out.println("float value, fv: "+ fv);
            System.out.println("double value, dv: "+ dv);
            System.out.println("char value, cv: "+ cv);
      }
}
```

Output:

```
Values of Wrapper objects (printing as objects)
Byte object byteobj:  1
Integer object intobj:  10
Float object floatobj:  18.6
Double object doubleobj:  250.5
Character object charobj: a
Unwrapped values (printing as data types)
byte value, bv: 1
int value, iv: 10
float value, fv: 18.6
double value, dv: 250.5
char value, cv: a
```

2.12 String Class, StringBuffer Class

Q38. What is a string in java? Name few string methods with example program?

Ans :

Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.

The Java platform provides the String class to create and manipulate strings.

Creating Strings

String can be created in number of ways, here are a few ways of creating string object.

a) Using a String Literal

String literal is a simple string enclosed in double quotes " ". A string literal is treated as a String object.

String str1 = "Hello";

b) Using another String object:

String str2 = newString(str1);

c) Using new Keyword

String str3 = newString("Java");

d) Using + operator (Concatenation)

String str4 = str1 + str2;

or,

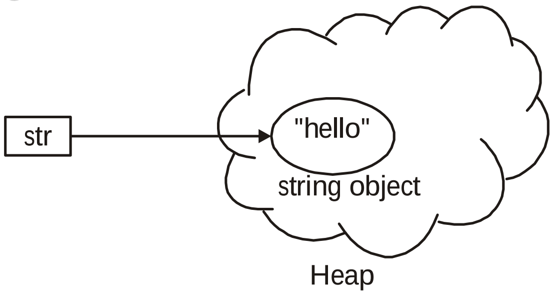
String str5 = "hello" + "Java";

Each time you create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as string constant pool inside the heap memory.

String Object and How they are stored

When we create a new string object using string literal, that string literal is added to the string pool, if it is not present there already.

String str = "Hello";



String Methods

- **int length():** Returns the number of characters in the String.

  "GeeksforGeeks".length(); // returns 13

- **Char charAt(int i):** Returns the character at ith index.

  "GeeksforGeeks".charAt(3); // returns 'k'

- **String substring (int i):** Return the substring from the ith index character to end.

  GeeksforGeeks".substring(3); // returns "ksforGeeks"

- **String substring (int i, int j):** Returns the substring from i to j-1 index.

  "GeeksforGeeks".substring(2, 5); // returns "eks"

- **String concat(String str):** Concatenates specified string to the end of this string.

  String s1 = "Geeks";

  String s2 = "forGeeks";

  String output = s1.concat(s2); // returns "GeeksforGeeks"

- **int indexOf (String s):** Returns the index within the string of the first occurrence of the specified string.

  String s = "Learn Share Learn";

  int output = s.indexOf("Share"); // returns 6

- **int indexOf (String s, int i):** Returns the index within the string of the first occurrence of the specified string, starting at the specified index.

  String s = "Learn Share Learn";

  int output = s.indexOf('a',3);// returns 8

- **Int lastindexOf(int ch):** Returns the index within the string of the last occurrence of the specified string.

  String s = "Learn Share Learn";

  int output = s.lastindexOf('a'); // returns 14

- **boolean equals(Object otherObj):** Compares this string to the specified object.

  Boolean out = "Geeks".equals("Geeks"); // returns true

  Boolean out = "Geeks".equals("geeks"); // returns false

- **boolean equalsIgnoreCase (String anotherString):** Compares string to another string, ignoring case considerations.

  Boolean out= "Geeks".equalsIgnoreCase("Geeks"); // returns true

  Boolean out = "Geeks".equalsIgnoreCase("geeks"); // returns true

- **int compareTo(String anotherString):** Compares two string lexicographically.

  int out = s1.compareTo(s2); // where s1 ans s2 are strings to be compared

  This returns difference s1-s2. If :

  out < 0 // s1 comes before s2

  out = 0 // s1 and s2 are equal.

  out >0 // s1 comes after s2.

- **int compareToIgnoreCase(String anotherString):** Compares two string lexicographically, ignoring case considerations.

  int out = s1.compareToIgnoreCase(s2);

  // where s1 ans s2 are

  // strings to be compared

This returns difference s1-s2. If :

out < 0 // s1 comes before s2

out = 0 // s1 and s2 are equal.

out >0 // s1 comes after s2.

Note: In this case, it will not consider case of a letter (it will ignore whether it is uppercase or lowercase).

- **String toLowerCase():** Converts all the characters in the String to lower case.

  String word1 = "HeLLo";

  String word3 = word1.toLowerCase(); // returns "hello"

  String toUpperCase(): Converts all the characters in the String to upper case.

  String word1 = "HeLLo";

  String word2 = word1.toUpperCase(); // returns "HELLO"

- **String trim():** Returns the copy of the String, by removing whitespaces at both ends. It does not affect whitespaces in the middle.

  String word1 = " Learn Share Learn ";

  String word2 = word1.trim(); // returns "Learn Share Learn"

- **String replace (char oldChar, char newChar):** Returns new string by replacing all occurrences of oldChar with newChar.

  String s1 = "feeksforfeeks";

  String s2 = "feeksforfeeks".replace('f' ,'g'); // returns "geeksgorgeeks"

Program

```
import java.io.*;

import java.util.*;

class Test

{

        public static void main (String[] args)

        {

              String s= "GeeksforGeeks";

              // or S tring s= new String ("GeeksforGeeks");

              // Returns the number of characters in the String.

              System.out.println("String length = "+ s.length());

              // Returns the character at ith index.

              System.out.println("Character at 3rd position = "+ s.charAt(3));

                    // Return the substring from the ith   index character // to end of string

        System.out.println("Substring "+ s.substring(3));

        // Returns the substring from i to j-1 index.
        System.out.println("Substring   = "+ s.substring(2,5));
        // Concatenates string2 to the end of string1.
        String s1 = "Geeks";
        String s2 = "forGeeks";
        System.out.println("Concatenated string   = "+s1.concat(s2));
        // Returns the index within the string  // of the first occurrence of the specified string.
        String s4 = "Learn Share Learn";
        System.out.println("Index of Share "+s4.indexOf("Share"));
        // Returns the index within the string of the // first occurrence of the specified string,
        // starting at the specified index.
        System.out.println("Index of a   = "+ s4.indexOf('a',3));
        // Checking equality of Strings
        Boolean out = "Geeks".equals("geeks");
        System.out.println("Checking Equality   "+ out);
        out = "Geeks".equals("Geeks");
        System.out.println("Checking Equality   "+ out);
        out = "Geeks".equalsIgnoreCase("gEeks ");
        System.out.println("Checking Equality"+ out);
        intout1 = s1.compareTo(s2);
        System.out.println("If s1 = s2"+ out);
        // Converting cases
        String word1 = "GeeKyMe";
        System.out.println("Changing to lower Case "+word1.toLowerCase());
        // Converting cases
        String word2 = "GeekyME";
        System.out.println("Changing to UPPER Case "+ word1.toUpperCase());
        // Trimming the word
        String word4 = " Learn Share Learn ";
        System.out.println("Trim the word "+ word4.trim());
        // Replacing characters
        String str1 = "feeksforfeeks";
        System.out.println("Original String "+ str1);
        String str2 = "feeksforfeeks".replace('f','g') ;
        System.out.println("Replaced f with g -> "+ str2);
        }
}
```

Output :

```
String length = 13
Character at 3rd position = k
Substring ksforGeeks
Substring = eks
Concatenated string = GeeksforGeeks
Index of Share 6
Index of a = 8
Checking Equality false
Checking Equality true
Checking Equalityfalse
If s1 = s2false
Changing to lower Case geekyme
Changing to UPPER Case GEEKYME
Trim the word Learn Share Learn
Original String feeksforfeeks
Replaced f with g -> geeksgorgeeks
```

2.12.1 String Buffer Class

Q39. What is a String Buffer class?

(OR)

Explain the usage of string buffers class.

Ans : **(Dec.-18)**

String Buffer class is used to create a mutable string object i.e its state can be changed after it is created. It represents growable and writable character sequence. As we know that String objects are immutable, so if we do a lot of changes with String objects, we will end up with a lot of memory leak.

So StringBuffer class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors. They are,

1. StringBuffer ( )
2. StringBuffer (int size)
3. StringBuffer (String str)
4. StringBuffer (charSequence [ ]ch)

- ➢ StringBuffer() creates an empty string buffer and reserves room for 16 characters.
- ➢ stringBuffer(int size) creates an empty string and takes an integer argument to set capacity of the buffer.

Methods

Some of the most used methods are:

- **length( ) and capacity( ):** The length of a StringBuffer can be found by the length( ) method, while the total allocated capacity can be found by the capacity( ) method.
- **append( ):** It is used to add text at the end of the existence text. Here are a few of its forms:

  StringBuffer append(String str)

  StringBuffer append(int num)
- **insert( ):** It is used to insert text at the specified index position. These are a few of its forms:

  StringBuffer insert(int index, String str)

  StringBuffer insert(int index, char ch)
- **reverse( ):** It can reverse the characters within a StringBuffer object using reverse( ).This method returns the reversed object on which it was called.
- **delete( ) and deleteCharAt( ):** It can delete characters within a StringBuffer by using the methods delete( )and deleteCharAt ( ). The delete( )method deletes a sequence of characters from the invoking object. Here, start Index specifies the index of the first character to remove, and end Index specifies an index one past the last character to remove. Thus, the substring deleted runs from start Index to endIndex–1. The resulting StringBuffer object is returned. The deleteCharAt( )method deletes the character at the index specified by loc. It returns the resulting StringBuffer object.These methods are shown here:

  String Buffer delete(int startIndex, int endIndex)

  StringBuffer deleteCharAt(int loc)
- **replace( ):** It can replace one set of characters with another set inside a StringBuffer object by calling replace( ). The substring being replaced is specified by the indexes start Index and endIndex. Thus, the substring at start Index through endIndex–1 is replaced. The replacement string is passed in str.The resulting StringBuffer object is returned.Its signature is shown here:

  StringBuffer replace(int startIndex, int endIndex, String str)
- **ensureCapacity( ):** It is used to increase the capacity of a StringBuffer object. The new capacity will be set to either the value we specify or twice the current capacity plus two (i.e. capacity+2), whichever is larger. Here, capacity specifies the size of the buffer.

  void ensureCapacity(int capacity)
- capacity():This method returns the current capacity of StringBuffer object.

  StringBuffer str =newStringBuffer();

  System.out.println(str.capacity());

Some Interesting Facts

1. java.lang.StringBuffer extends (or inherits from) Object class.
2. All Implemented Interfaces of StringBuffer class:Serializable, Appendable, CharSequence.
3. public final class StringBuffer extends Object implements Serializable, CharSequence
4. String buffers are safe for use by multiple threads. The methods can be synchronized wherever necessary so that all the operations on any particular instance behave as if they occur in some serial order.
5. Whenever an operation occurs involving a source sequence (such as appending or inserting from a source sequence) this class synchronizes only on the string buffer performing the operation, not on the source.
6. It inherits some of the methods from Object class which are clone, equals, finalize, getClass, hashCode, notify, notifyAll.

String Builder class

String Builder is identical to StringBuffer except for one important difference that it is not synchronized, which means it is not thread safe. Its because StringBuilder methods are not synchronised.

StringBuilder Constructors

1. **StringBuilder** ( ), creates an empty StringBuilder and reserves room for 16 characters.

2. **StringBuilder** (int size), create an empty string and takes an integer argument to set capacity of the buffer.

3. **StringBuilder** (String str), create a StringBuilder object and initialize it with string str.

Short Question and Answers

1. Define Constructors.

Ans :

Constructor is a special method that creates and return an object of the class in which they are defined. Constructor has the same name as that of class and has no return type not even void

Types of Constructor in Java

There are three types of constructor in Java-

(i) Default Constructor

(ii) Parameterized Constructor

(iii) Copy Constructor

2. Discuss constructors overloading with super keyword.

Ans :

A constructor is a special member function that gets executed on the object creation automatically. It is responsible to initialize the objects of its class and to allocate memory space for them. Constructors have same name as the class name. But they does not have return type and void. They does not even return any values. The constructor will be executed whenever the objects are created. When a class object is created, the instance variables for it must be initialized with some values using the initializer. There are two types of constructors, they are,

(i) Default constructor

(ii) Parameterized constructor.

3. What is an abstract class?

Ans :

If a class contain any abstract method then the class is declared as abstract class. An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

Syntax

abstract class class_name { }

4. Define Interface?

Ans :

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have methods and variables but the methods declared in interface contain only method signature, not body.

5. Protection access

Ans :

Java provides many levels of protections to the variables and methods to be visible which are present inside classes, subclasses and packages. The classes and packages both encapsulates the name space, scope of variables and methods. A class consist of data and code whereas package consist of related classes and sub packages. Because of the interaction between classes and packages,

Java offers the following four categories of visibility for accessing class members. They are,

(i) Subclasses in the same package

(ii) Non-sub classes in the same package

(iii) Sub classes in different packages

6. What is a package in java?

Ans :

Package 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 or sub-package. Using package it becomes easier to locate the related classes and it also provides a good structure for projects with hundreds of classes and other files.

Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:

- Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.
- Packages can be considered as data encapsulation (or data-hiding).

Types of Packages: Built-in and User defined

- **Built-in Package:** Existing Java package for example java.lang, java.util etc.
- **User-defined-Package:** Java package created by user to categorize their project's classes and interface.

7. What is the major difference between an interface and a class.

Ans :

The major difference between an interface and a class is that interface contains the methods that are abstract in nature and it does not include any code where as the class contains methods with some code. The members of interface are always constant where as the members of a class are either variable or constant.

8. When do we decide a method

Ans :

Java defines abstract classes and methods using the keyword "abstract". The class that doesn't have body is called abstract class. Whereas the method that does not have body is called abstract method. An abstract class can contain subclasses in addition to abstract methods. If the method containing in the subclasses does not override the methods of the baseclass then they will be considered as abstract classes. So, all the subclass must provide an implementation for all the base class's abstract methods. Abstract classes cannot be instantiated.

9. What are wrapper class?

Ans :

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need

(i) They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).

(ii) The classes in java.util package handles only objects and hence wrapper classes help in this case also.

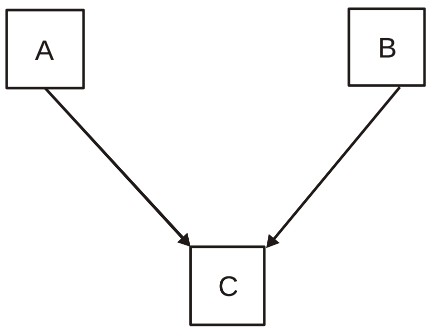
(iii) Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

(iv) An object is needed to support synchronization in multithreading.

10. What is multiple Inheritance?

Ans :

"Multiple Inheritance" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



Multiple Inheritance

Note:

(i) Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

(ii) Most of the new OO languages like Small Talk, Java, C# do not support Multiple inheritance. Multiple Inheritance is supported in C++.

11. Command-Line Arguments

Ans :

Command Line Argument is information passed to the program when you run the program. The passed information is stored as a string array in the main method.

Important Points

- Command Line Arguments can be used to specify configuration information while launching your application.
- There is no restriction on the number of java command line arguments.You can specify any number of arguments
- Information is passed as Strings.
- They are captured into the String args of your main method

12. Parameterized Constructor

Ans :

This constructor is called when an object is created and it is initialized with some values at the time of creation.

Example

```
public class Employee
{
        int id;
        String name;
        public Employee(int i, String n)
        {
            id = i;
            name = n;
        }
        void show()
        {
            System.out.println(id+" "+name);
        }
        public static void main(String args[])
        {
            Employee emp1 = new Employee(1,
            "Govind");
            Employee emp2 = new Employee(2,
            "Akash");
            emp1.show();
            emp2.show();
        }
}
```

Output

```
        1 Govind
        2 Akash
```

13. Explain the usage of string buffers class.

Ans :

String Buffer class is used to create a mutable string object i.e its state can be changed after it is created. It represents growable and writable character sequence. As we know that String objects are immutable, so if we do a lot of changes with String objects, we will end up with a lot of memory leak.

So StringBuffer class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors. They are,

(i) StringBuffer ( )

(ii) StringBuffer (int size)

(iii) StringBuffer (String str)

(iv) StringBuffer (charSequence [ ]ch)

14. Compare and contrast method overloading and method overriding.

Ans :

S.No.	Method Overloading	Method Overriding
1	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.

15. What is Static Variable in Java?

Ans :

Static variable in Java is variable which belongs to the class and initialized only once at the start of the execution.

➢ It is a variable which belongs to the class and not to object(instance)

➢ Static variables are initialized only once, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables

➢ A single copy to be shared by all instances of the class

➢ A static variable can be accessed directly by the class name and doesn't need any object

Syntax:

<class-name>.<variable-name>

16. What is an Array?

Ans :

Java array is an object the 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.

Array in java is index based, first element of the array is stored at 0 index.

Creating Arrays

You can create an array by using the new operator with the following syntax

Syntax

arrayRefVar = new dataType[arraySize];

17. Inheritance

Ans :

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object. It is an important part of OPPs(Object Oriented programming system).

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.Inheritance represents the IS-A relationship, also known as parent-child relationship.

18. Explain the benefits of Inheritance.

Ans :

- **Code sharing:** Code sharing occurs at two levels. At first level many users or projects can use the same class. In the second level sharing occurs when two or more classes developed by a single programmer as part of a project which is being inherited from a single parent class. Here also code is written once and reused. This is possible through inheritance.
- **Consistency of inheritance:** When two or more classes inherit from the same superclass we are assured that the behaviour they inherit will be the same in all cases. Thus we can guarantee that interfaces to similar objects are in fact similar.
- **Software components:** Inheritance provides programmers the ability to construct reusable s/w components. The goal behind these is to provide applications that require little or no actual coding. Already such libraries and packages are commercially available.
- **Rapid Prototyping:** When a s/w system is constructed largely out of reusable components development can be concentrated on understanding the new and unusual portion of the system. Thus s/w system can be generated more quickly and easily leading to a style of programming known as 'Rapid Prototyping' or 'exploratory programming'
- **Polymorphism and Framework:** Generally s/w is written from the bottom up , although it may be designed from top-down. This is like building a wall where every brick must be laid on top of another brick i.e., the lower level routines are written and on top of these slightly higher abstraction are produced and at last more abstract elements are generated. Polymorphism permits the programmer to generate high level reusable components.

Choose the Correct Answers

1. JDK Stands for ________ [d]
   (a) Java Data Kit (b) Java Definition Kit
   (c) Java Design Kit (d) Java Development Kit
2. Which type of inheritance is not supported by java ________ [a]
   (a) Multiple (b) Multilevel
   (c) Single (d) None
3. Multiple inheritance is supported by ________ [b]
   (a) Classes (b) Interfaces
   (c) Objects (d) Abstract classes
4. Java arrays are ________ [b]
   (a) classes (b) Objects
   (c) Variables (d) none
5. Which of these have highest precedence ________ [a]
   (a) ( ) (b) + +
   (c) * (d) > >
6. Which of this keyword must be used to inherit a class? [d]
   (a) super (b) this
   (c) extent (d) extends
7. A class member declared protected becomes a member of subclass of which type? [b]
   (a) public member (b) private member
   (c) protected member (d) static member
8. Which of these is correct way of inheriting class A by class B? [c]
   (a) class B + class A {} (b) class B inherits class A {}
   (c) class B extends A {} (d) class B extends class A {}
9. What is not type of inheritance? [b]
   (a) Single inheritance (b) Double inheritance
   (c) Hierarchical inheritance (d) Multiple inheritance
10. Using which of the following, multiple inheritance in Java can be implemented? [a]
    (a) Interfaces (b) Multithreading
    (c) Protected methods (d) Private methods

Fill in the blanks

1. The ________ statement allows number of possible execution paths.
2. An ________ is structure that hold multiple values of same data type.
3. A class contains only ________ or ________.
4. ________ is extends one class to one class.
5. ________ is a keyword used for implementing interface.
6. Java ________ is a class which is declared inside the class or interface.
7. It is a collection of ________ which is used to perform some operation.
8. ________ are optional and are used in the method declaration.
9. ________ is a special method that creates and return an object of the class in which they are defined.
10. ________ can be used to specify configuration information while launching your application.

ANSWERS

1. Switch
2. Array
3. Methods ,Data
4. Single Inheritance
5. Implements
6. Inner Class
7. statement
8. Modifiers
9. Constructor
10. Command Line Arguments

One Mark Answers

1. Modifiers

Ans :

Modifiers are optional and are used in the method declaration. There are a number of modifiers that can be used with a method declaration.

2. Default Constructor

Ans :

This constructor takes no argument and is called when an object is created without any explicit initialization.

3. Copy Constructor

Ans :

This constructor is called when an object is created and it is initialized with some other object of the same class at the time of creation.

4. Static keyword.

Ans :

The static keyword is used in java mainly for memory management. It is used with variables, methods, blocks and nested class.

5. Arrays

Ans :

Java array is an object the 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.

6. Single Inheritance

Ans :

Single inheritance is easy to understand. When a class extends another one class only then we call it a single inheritance.

7. Extends keyword.

Ans :

The most important and widely used keywords in Java which is the extends keyword. It is a reserved word that and we can't use it as an identifier other than the places where we need to inherit the classes in Java.

8. Super keyword

Ans :

Super keyword in java is a reference variable that is used to refer parent class object.

UNIT III

Exception: Introduction, Types, Exception Handling Techniques, User-Defined Exception. Multithreading: Introduction, Main Thread and Creation of New Threads– By Inheriting the Thread Class or Implementing the Runnable Interface, Thread Lifecycle, Thread Priority and Synchronization.

Input/Output: Introduction, java.io Package, File Streams, FileInputStream Class, FileOutputStream Class, Scanner Class, BufferedInputStream Class, BufferedOutputStream Class, RandomAccessFile Class.

3.1 Exception

3.1.1 Introduction

Q1. What is exception?

(OR)

Define exception.

Ans : **(June-19, MGU, Dec.-18, MGU)**

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

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.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

- **Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions.

  Note: Since the methods read() and close() of FileReader class throws IOException, you can observe that the compiler notifies to handle IOException, along with FileNotFoundException.

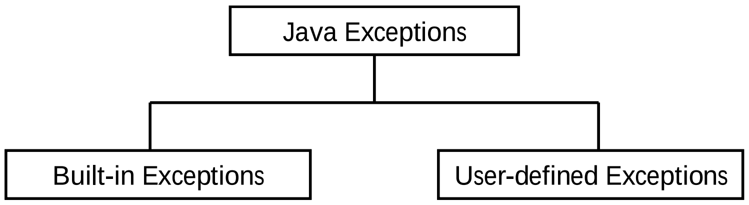
- **Unchecked exceptions:** An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation.

3.1.2 Types

Q2. Explain various types of exceptions.

Ans :

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their own exceptions.



I. Built-in Exceptions

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **Arithmetic Exception:** It is thrown when an exceptional condition has occurred in an arithmetic operation.
2. **Array Index Out Of Bound Exception:** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
3. **Class Not Found Exception:** This Exception is raised when we try to access a class whose definition is not found
4. **File Not Found Exception:** This Exception is raised when a file is not accessible or does not open.
5. **IO Exception:** It is thrown when an input-output operation failed or interrupted
6. **Interrupted Exception:** It is thrown when a thread is waiting , sleeping, or doing some processing , and it is interrupted.
7. **No Such Field Exception:** It is thrown when a class does not contain the field (or variable) specified
8. **No Such Method Exception:** It is thrown when accessing a method which is not found.
9. **Null Pointer Exception:** This exception is raised when referring to the members of a null object. Null represents nothing.
10. **Number Form at Exception:** This exception is raised when a method could not convert a string into a numeric format.
11. **Runtime Exception:** This represents any exception which occurs during runtime.
12. **String Index Out Of Bounds Exception:** It is thrown by String class methods to indicate that an index is either negative than the size of the string.

Programs :

StringIndexOutOfBound Exception

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

```
        {
              try
              {
              String a = “This is like chipping “; // length is 22
              charc = a.charAt(24); // accessing 25th element
                            System.out.println(c);
              }
              catch(StringIndexOutOfBoundsException e)
              {
                     System.out.println(“StringIndexOutOfBoundsException”);
              }
        }
}
```

NumberFormat Exception

```
class   NumberFormat_Demo
{
        public static void main(String args[])
        {
              try
              {
                     // “akki” is not a number
                   intnum = Integer.parseInt (“akki”) ;
                    System.out.println(num);
              }
              catch(NumberFormatException e)
              {
                     System.out.println(“Number format exception”);
              }
        }
}
```

ArrayIndexOutOfBounds Exception

```
class ArrayIndexOutOfBound_Demo
{
        public static void main(String args[])
        {
              try
              {
                     inta[] = newint[5];
                   a[6] = 9; // accessing 7th element in an array of
                   // size 5
              }
```

```
            catch(ArrayIndexOutOfBoundsException e)
            {
                  system.out.println (“Array Index is Out Of Bounds”);
            }
      }
}
```

II. User-Defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called ‘user-defined Exceptions’.

Following steps are followed for the creation of user-defined Exception.

- The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:
- class MyException extends Exception
- We can write a default constructor in his own exception class.

MyException(){}

- We can also create a parameterized constructor with a string as a parameter.

We can use this to store exception details. We can call super class(Exception) constructor from this and send the string there.

```
MyException(String str)
{
super(str);
}
```

- To raise exception of user-defined type, we need to create an object to his exception class and throw it using throw clause, as:
- MyException me = new MyException(“Exception details”);
- throw me;
- In main() method, the details are displayed using a for-loop. At this time, check is done if in any account the balance amount is less than the minimum balance amount to be ept in the account.
- If it is so, then MyException is raised and a message is displayed “Balance amount is less”.

```
      class MyException extends Exception
{
      //store account information
      private static int accno[] = {1001, 1002, 1003, 1004};
      private static String name[] ={“Nish”, “Shubh”, “Sush”, “Abhi”, “Akash”};
      private static double bal[] ={10000.00, 12000.00, 5600.0, 999.00, 1100.55};
      // default constructor
      MyException()
      {
      }
```

```
        // parametrized constructor
        MyException(String str)
        {
              super(str);
        }
        // write main()
        public static void main(String[] args)
        {
              try
              {
                     // display the heading for the table
                     System.out.println("ACCNO"+ "\t"+ "CUSTOMER"+ "\t"+ "BALANCE");
                     // display the actual account information
                     for(int i = 0; i < 5; i++)
                     {
                            System.out.println(accno[i] + "\t"+ name[i] +"\t"+ bal[i]);
                            // display own exception if balance < 1000
                            if(bal[i] < 1000)
                            {
                                   MyException me =newMyException("Balance is less than 1000");
                                   throwme;
                            }
                     }
        }
        catch(MyException e)
        {
              e.printStackTrace();
        }
    }
}
MyException: Balance is less than 1000
at MyException.main(fileProperty.java:36)
```

Output:

```
               ACCNO          CUSTOMER       BALANCE
               1001           Nish           10000.0
               1002           Shubh          12000.0
               1003           Sush            5600.0
               1004           Abhi             999.0
```

3.1.3 Exception Handling Techniques

Q3. Explain the handling of exception in Java.

Ans : **(June-19-MGU, Dec.-18-MGU)**

Exception handling can be defined as a mechanism of handling exceptions that can be occurred at run-time. This mechanism is commonly used to prevent the malfunctions such as computer deadlock or computer hanging. Some examples of run-time exception are divide by zero, array out of bounds exceptions.

The run-time exception can be handled by using five keywords namely try, catch, throw, throws and finally. The code that can generate exception is usually placed in try block. If an exception is occurred, execution flow finds the similar catch block and leaves the try block. The catch block handles the exception and generates a message specifying the type of exception. Some of these exception types are as follows,

(i) ArithmeticException

(ii) ArrayIndexOutOfBoundsException.

However, there is no rule that the try block must contain corresponding catch block. Each try block may contain zero or multiple catch blocks.

When an exception is thrown and if it is matched with any of the catch block, then the statements of corresponding catch block can be executed. Otherwise, the catch block can be skipped and the statements present next to the catch block will be executed. The finally block is declared after all the catch blocks whose code can be executed irrespective to the occurrence of exception. Note that, the finally block is optional.

If a method throw an exception, then it can be handled by catch block. This exception can be thrown using ‘throws’ clause. When control leaves the throws block, then it cannot return back to the ‘throws’ clause. If the thrown exception cannot be handled by any catch block, then it can be handled by default handler called as ‘Termination Model Of Exception Handler’.

Syntax for Exception Handling

```
try
{
    //Block of code to check exception
}
catch (Exception Typel exception object)
{
    //code to handle exceptions
}
catch (Exception Type2 exception object)
{
    //code to handle exceptions
}
finally
{
//  code to be executed before/after try-catch block
}
```

Q4. State the Benefits of Exception Handling.

Ans :

The benefits of exception handling are as a follows,

(i) It can control run-time errors that occurs in the program.

(ii) It can avoid abnormal termination of the program and also shows the behavior of program to users.

(iii) It can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception.

(iv) It can separate the error handling code and normal code by using try-catch block.

(v) It can produce the normal execution flow for a program.

(vi) Its mechanism can implement a clean way to propagate error. That is when an invoking method cannot manage a particular situation, then it throws an exception and asks the invoking method to deal with such situation.

(vii) Its mechanism develops a powerful coding that ensures that the exceptions can be prevented.

Q5. What are the different ways to handle exceptions.

Ans :

In java, exception handling is done using five keywords,

1. try
2. catch
3. throw
4. throws
5. finally

Exception handling is done by transferring the execution of a program to an appropriate exception handler when exception occurs.

try and catch block in Java

When your program performs an operation that generates an exception, an exception will be thrown. You can catch that exception and handle it with the help of try and catch blocks.

try block in Java

➢ In try block, we place the line of codes that may cause an exception.

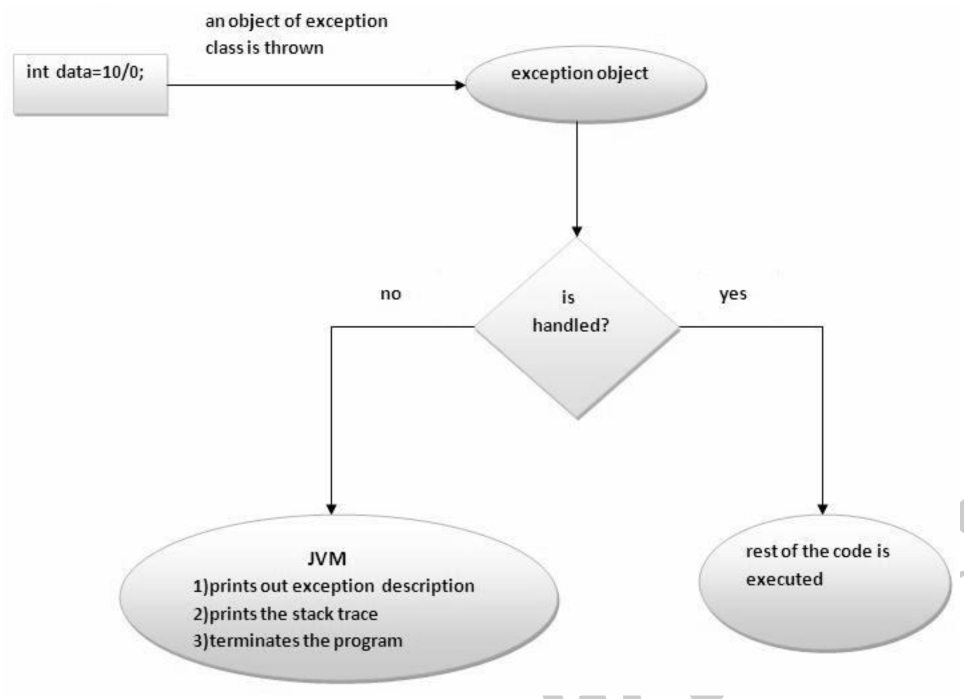
➢ A try block must be followed by a catch block or a finally block.

catch block in Java

➢ The catch block is used to write the code that handles the exception.

➢ A catch block is only executed if it catches the exception coming from the try block.

If no exception is generated from the try block, then all the catch blocks are skipped.


an object of exception class is thrown
int data=10/0;
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Syntax of java try-catch

```
try
{
        //code that may throw exception
}
        catch(Exception_class_Name ref)
{
}
```

Program

```
class Exceptions
{
        public static void main(String[] args)
        {
            String languages[] = { "C", "C++", "Java", "Perl", "Python" };
            try
            {
                for (int c = 1; c <= 5; c++)
                {
                        System.out.println(languages[c]);
                }
            }
            catch (Exception e)
            {
                System.out.println(e);
            }
        }
}
```

Points to remember

- In a method, there can be more than one statements that might throw exception, So put all these statements within its own try block and provide separate exception handler within own catch block for each of them.
- If an exception occurs within the try block, that exception is handled by the exception handler associated with it. To associate exception handler, we must put catch block after it. There can be more than one exception handlers. Each catch block is a exception handler that handles the exception of the type indicated by its argument. The argument, ExceptionType declares the type of the exception that it can handle and must be the name of the class that inherits from Throwable class.
- For each try block there can be zero or more catch blocks, but only one finally block.
- The finally block is optional. It always gets executed whether an exception occurred in try block or not . If exception occurs, then it will be executed after try and catch blocks. And if exception does not occur then it will be executed after the try block. The finally block in java is used to put important codes such as clean up code e.g. closing the file or closing the connection.

Q6. Explain the use of Multiple Catch statements in exceptions?

Ans :

Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

A try can have multiple catch block. The catch block that catches the exception is executed while all others are skipped. If no exception arises from the try block, all the catch blocks are skipped.

Syntax

```
try
{
    //code that may generate exception.
}
catch(ExceptionClass1 obj)
{
    //code to handle the exception.
}
catch(ExceptionClass2 obj)
{
    //code to handle the exception.
}
...
catch(ExceptionClassn obj)
{
    //code to handle the exception.
}
```

Program

```
class ExceptionHandling
{
        public static void main(String args[])
        {
              try
              {
                    int a[] = new int[3];
                    a[4] = 30;
              }
              catch(IndexOutOfBoundsException i){
                    System.out.println(i);
              }
              catch(ArrayIndexOutOfBoundsException ai){
                    System.out.println(ai);
              }
        }
}
```

Q7. Write a short note on :

(a) throw

(b) throws

(c) finally

Ans :

(a) throw

'throw' is a Java keyword used in exception handling. Generally, a try block checks if any exceptions are raised and when an error occurs, it throws the error which is caught by the catch statement. Basically, the exceptions thrown by the Java run-time system are being caught, but throw statement allows a program to throw an exception explicitly.

Syntax

```
throw ThrowableInstance;
```

The ThrowableInstance should be an object of either Throwable class or any of its subclass

(b) throws

throws clause handles the unhandled exceptions that are generated by a method. This clause list the types of exceptions that can be thrown by a method. The throws clause can be used for specifying any exception except the exception belonging to error class or any of its subclass.

Syntax

```
return-type method-name (arg-list) throws exception-list
{
      //body
}
```

In the above syntax, exception-list is the list of exception separated by comma. The throws keyword is used to list the exceptions that can be thrown by a method. If the possible exceptions are not listed in the method of throws clauses, the compile time error is generated.

(c) finally

Every try block should be associated with atleast one catch or finally block. However, it is optional to have a finally block within a program. If a finally block is included in a program, then it will be executed after the try/ catch block but before the code following this try/catch block. The finally block will be executed irrespective of the occurrence of exceptions. A finally block helps in closing opened files, deallocating the allocated memories etc.

The syntax of the finally block is,

```
try
{
    // Statements
}
catch(...)
{
    // Statements
}
catch(...)
{
    // Statements
}
finally
{
    // Statements
}
    // statement/code following try/catch block.
```

3.2 User-Defined Exception

Q8. Define User-Defined Exception.

Ans :

An exception which can be created by the user manually are called as user-defined exception. These exceptions are handled by Java. While creating user-defined exceptions, user must satisfy the below steps,

Step 1: The user-defined exceptions must be created as child exceptions of throwable class.

Step 2: If the user-defined exception is a checked exception, the exception class must be extended.

Step 3: When the runtime exception is declared as user-defined exception, then the RuntimeException class mm be extended.

Constructors

A user-defined exception can be created using the following constructors,

(a) MyException()

(b) MyException(String error)

(c) MyException(String error, Exception ex).

3.3 Multithreading

3.3.1 Introduction

Q9. Explain the Multithreading in Java.

(OR)

Write about Multithreading in Java.

Ans : **(Dec.-19)**

Multithreading in java is a process of executing multiple threads simultaneously.

Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Multitasking

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- ➢ Process-based Multitasking(Multiprocessing)
- ➢ Thread-based Multitasking(Multithreading)

Process-based Multitasking (Multiprocessing)

- ➢ Each process have its own address in memory i.e. each process allocates separate memory area.
- ➢ Process is heavy weight.

- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

3.3.2 Main Thread

Q10. Explain in detail about Main Thread.

Ans :

The main thread is the first thread that will begin its execution immediately when the Java program starts up. The main thread is important for two reasons. First, it allows the other child threads to be created. Second, reason is that the thread perform various shutdown actions. Therefore, it must be the last thread to finish execution.

Eventhough the main thread starts executing immediately, it can be controlled using an object of Thread class. The object of Thread is created by using its static method currentThread().

General Format

```
public static Thread currentThread()
```

This method when called will returns a reference to currently executing thread. After a reference to the main thread is obtained it can be controlled like any other threads. For example, the methods such as getName(), setNamef), sleep() etc., can be invoked on the main thread.

Program

```
import java.io.*;
class MainThread
{
public static void main(String args[])
{
Thread thd = Thread.currentThread();
System.out.println("Running thread:" + thd);
// change the name of the thread
thd.setName("First Thread");
System.out.println("Running Thread is:" + thd);
try {
for(int i=5;i> 0;i—)
{
System.out.println(i);
Thread. sleep( 1000);
}
} catch (InterruptedException e)
{
System.out.println("Main thread got interrupted");
}
}
}
```

3.3.3 Creation of New Threads

3.3.3.1 By Inheriting the Thread Class or Implementing the Runnable Interface

Q11. Define Threads. List different ways of creating threads. Explain with examples.

Ans : **(July-21, June-19, MGU, Dec.-18, MGU)**

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class

- ➢ Thread class provide constructors and methods to create and perform operations on a thread.
- ➢ Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- ➢ Thread()
- ➢ Thread(String name)
- ➢ Thread(Runnable r)
- ➢ Thread(Runnable r,String name)

Commonly used methods of Thread class:

1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long milliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.

11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.
21. **public boolean isInterrupted():** tests if the thread has been interrupted.
22. **public static boolean interrupted():** tests if the current thread has been interrupted.

Runnable Interface

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

Starting a thread

start() method of Thread class is used to start a newly created thread. It performs following tasks:

- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.

When the thread gets a chance to execute, its target run() method will run.

```
class Multi extends Thread
{
        public void run()
        {
        System.out.println("thread is running...");
        }
        public static void main(String args[]){
        Multi t1=new Multi();
        t1.start();
        }
   }
```

Output

```
        thread is running...
```

Java Thread By Implementing Runnable Interface

- A Thread can be created by extending Thread class also. But Java allows only one class to extend, it wont allow multiple inheritance. So it is always better to create a thread by implementing Runnable interface. Java allows you to implement multiple interfaces at a time.
- By implementing Runnable interface, you need to provide implementation for run() method.
- To run this implementation class, create a Thread object, pass Runnable implementation class object to its constructor. Call start() method on thread class to start executing run() method.
- Implementing Runnable interface does not create a Thread object, it only defines an entry point for threads in your object. It allows you to pass the object to the Thread(Runnable implementation) constructor.

```
class Multi3 implements Runnable
{
     public void run()
{
           System.out.println(“thread is
           running...”);
     }
     public static void main(String args[])
     {
           Multi3 m1=new Multi3();
           Thread t1 =new Thread(m1);
           t1.start();
     }
}
```

3.3.4 Thread Lifecycle

Q12. Describe the life cycle of thread.

Ans :

Every thread has a lifecycle, which consists of following states,

(a) Bom state

(b) Runnable state

(c) Running state

(d) Blocked state

(e) Dead state

A thread can be present in any one of these states at an instance. It can switch from one state to another state using various methods. The following figure depicts the life cycle of thread,

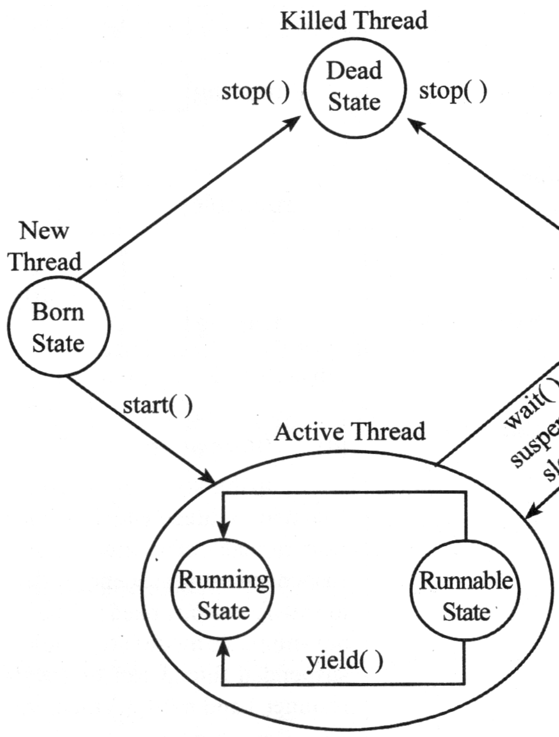


Fig. : Life Cycle of a Thread

(a) Born State

In this state, a new thread can be bom by creating a thread object. This new thread can be used to perform the following task,

(i) It can be scheduled to running state using start() method, thereby changing its state from bom to runnable.

(ii) It can be killed by using stop() method, thereby changing its state from bom to dead.

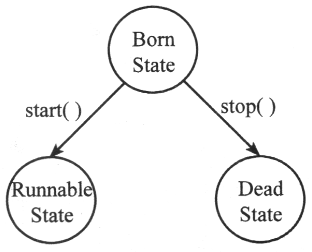


Fig. : Block Diagram of Schedule Born Thread

(b) Runnable State

In this state, the thread is ready for execution and waits for the availability of processor. The thread which is in runnable state joins the queue of thread that are waiting for processor. Once, the processor is empty, next waiting thread with highest priority gets executed. If multiple threads in the queue have equal priority, then the threads can be scheduled (allocating time slots) using round-robin algorithm (first-come, first-serve manner). However in runnable state, there is a facility of executing a thread before its turn using yield() method.

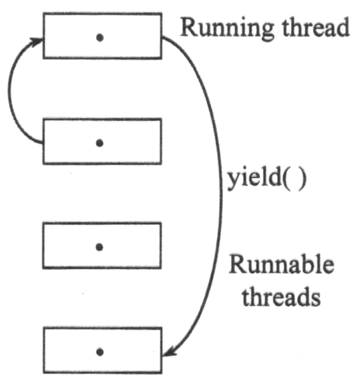


Fig. : Block Diagram of Relinquish Control Using yield)) Method

(c) Running State

In this state, the execution of a thread starts. The executing thread can leave the control by its own or by preempting with the thread that have highest priority. The running thread can leave (relinquish) the control in following situations.

Case 1: If running thread is suspended by using suspend() method. A thread can be suspended whenever there is a requirement to keep the thread idle for sometime due to which the running thread moves to block state. This suspended thread can be resumed to its execution using resume() method.

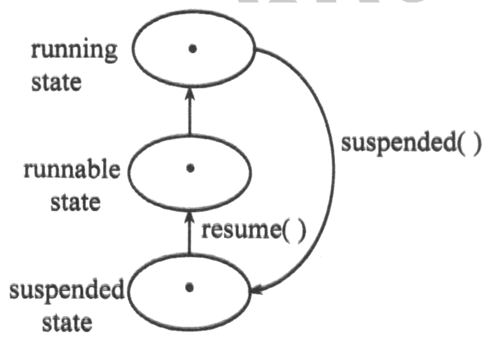


Fig. : Block Diagram of Relinquish Control Using resume!) Method

Case 2: If running thread is made to sleep for specified time using sleep (time) method then the thread will enter into the suspend state. It resides out of the queue until the time period is completed. After the completion of time period, the thread automatically enters into the runnable state.

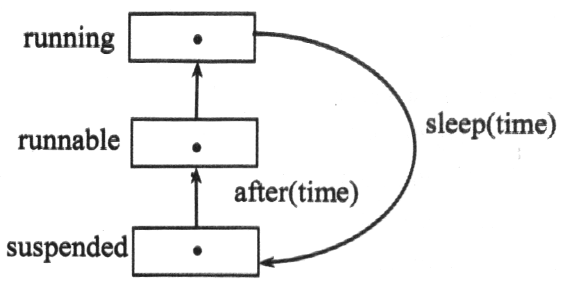


Fig. : Block Diagram To Relinquish Control Using sleepl) Method

Case 3: If running is made to wait thread using wait() method then the thread must wait until an event is occurred. The waiting thread can be resumed to its execution using notify() method.

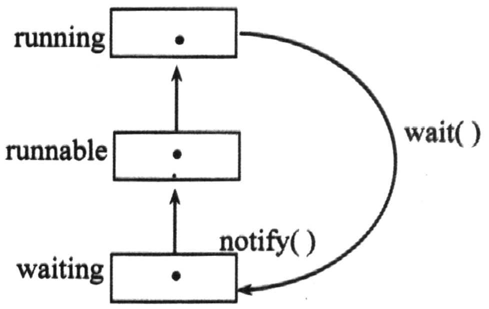


Fig. : Block Diagram of Relinquish Control Using waitf) Method

(d) Blocked State

In this state, the thread can be blocked until it gets resumed. A thread which is suspended can be prevented from runnable state and running state. Any thread can be suspended using suspend(), sleep() and wait() methods to satisfy various needs. The blocked thread will be in non-runnable mode (i.e., in idle mode) in dead state. The suspended thread can be resumed to execution using resume() and notify() methods.

(e) Dead State

In this state, the thread can be killed. After the successful completion of execution, thread will undergo natural death. However if a thread is killed in bom state, runnable state, running state and suspended state using stop( ) method then this sort of death is be called as premature death.

3.3.5 Thread Priority

Q13. Discuss the Priority of threads used in JAVA?

Ans :

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Get and Set Thread Priority:

1. **public final int getPriority():** java.lang.Thread.getPriority() method returns priority of given thread.
2. **public final void setPriority(int newPriority):** java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

```
// Java program to demonstrate getPriority() and setPriority()
importjava.lang.*;
class ThreadDemo extendsThread
{
        public voidrun()
        {
              System.out.println("Inside run method");
        }
        public static void main(String[] args)
        {
                    ThreadDemo t1 = newThreadDemo();
                    ThreadDemo t2 = newThreadDemo();
                    ThreadDemo t3 = newThreadDemo();
                    System.out.println("t1 thread priority : "+t1.getPriority()); // Default 5
                    System.out.println("t2 thread priority : "+t2.getPriority()); // Default 5
                    System.out.println("t3 thread priority : "+t3.getPriority()); // Default 5
                           t1.setPriority(2);
                    t2.setPriority(5);
                    t3.setPriority(8);

// t3.setPriority(21); will throw IllegalArgumentException
                    System.out.println("t1 thread priority : "+t1.getPriority());   //2
                    System.out.println("t2 thread priority : "+t2.getPriority()); //5
                    System.out.println("t3 thread priority : "+t3.getPriority());//8

                    System.out.print(Thread.currentThread().getName());
                    System.out.println("Main thread priority : "+ Thread.currentThread().getPriority());
                           Thread.currentThread().setPriority(10);
                    System.out.println("Main thread priority : "+ Thread.currentThread().getPriority());
        }
}
```

Output:

```
t1 thread priority : 5
t2 thread priority : 5
t3 thread priority : 5
t1 thread priority : 2
t2 thread priority : 5
t3 thread priority : 8
Main thread priority : 5
Main thread priority : 10
```

Tips

- Thread with highest priority will get execution chance prior to other threads. Suppose there are 3 threads t1, t2 and t3 with priorities 4, 6 and 1. So, thread t2 will execute first based on maximum priority 6 after that t1 will execute and then t3.
- Default priority for main thread is always 5, it can be changed later. Default priority for all other threads depends on the priority of parent thread.
- If two threads have same priority then we can't expect which thread will execute first. It depends on thread scheduler's algorithm(Round-Robin, First Come First Serve, etc)

If we are using thread priority for thread scheduling then we should always keep in mind that underlying platform should provide support for scheduling based on thread priority.

3.3.6 Synchronization

Q14. What are the synchronized methods and statements?

Ans :

Synchronization of threads ensures that if two or more threads need to access a shared resource then that resource is used by only one thread at a time. You can synchronize your code using the synchronized keyword. You can invoke only one synchronized method for an object at any given time.

Synchronization is based on the concept of monitor. A monitor, also known as a semaphore, is an object that is used as a mutually exclusive lock. All objects and classes are associated with a monitor and only one thread can won a monitor at a given time.

The monitor controls the way in which synchronized methods access an object or class. When a thread acquires a lock, it is said to have entered the monitor. The monitor ensures that only one thread has access to the resources at any given time. To enter an object's monitor, you need to call a synchronized method.

When a thread is within a synchronized method, all the other threads that try to call it on the same instance have to wait. During the execution of a synchronized method, the object is locked so that no other synchronized method can be invoked. The monitor is automatically released when the method completes its execution. The monitor can also be released when the synchronized method executes the wait() method. When a thread calls the wait() method, it temporarily releases the locks that it holds.

The Synchronized Statement

Synchronization among threads is achieved by using synchronized statements. The synchronized statements are used where the synchronization methods are not used in a class and you do not have access to the source code. You can synchronize the access to an object of this class by placing the calls to the methods defined by it inside a synchronized block .

Syntax:

```
synchronized(obj)
{
    // statements;
}
```

Inter-thread communication

Java supports inter-thread communication using wait(), notify(), notifyAll() methods. These methods are implemented as final methods in **Object**. So all classes have them. all three methods can be called only from within a synchronized context.

- **wait()** tells the calling thread to give up the monitor and go to sleep until some other thread enters the same monitor and calls notify()
- **notify()** wakes up the first thread that called wait() on the same object.
- **notifyAll()** wakes up all the threads that called wait() on the same object. The highest priority thread will run first.

3.4 Input/Output

3.4.1 Introduction, java.io Package

Q15. What is java.io Package? Explain the different ways of input output methods.

Ans : **(July-21)**

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform file handling in java by Java I/O API.

IO Stream

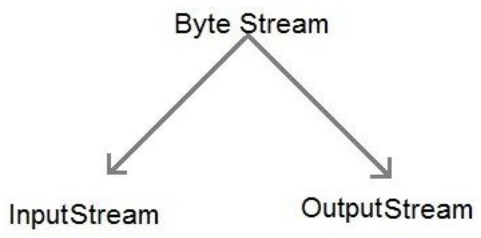
Java performs I/O through Streams. A Stream is linked to a physical layer by java I/O system to make input and output operation in java. In general, a stream means continuous flow of data. Streams are clean way to deal with input/output without having every part of your code understand the physical.

Java encapsulates Stream under java.io package. Java defines two types of streams. They are,

1. **Byte Stream :** It provides a convenient means for handling input and output of byte.
2. **Character Stream :** It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

Byte Stream Classes

Byte stream is defined by using two abstract class at the top of hierarchy, they are InputStream and OutputStream.



These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc.

Some important Byte stream Classes

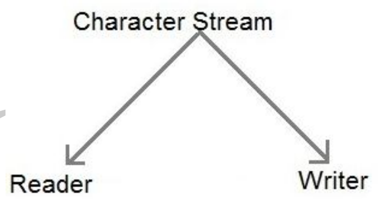
Stream class	Description
BufferedInputStream	Used for Buffered Input Stream.
BufferedOutputStream	Used for Buffered Output Stream.
DataInputStream	Contains method for reading java standard datatype
DataOutputStream	An output stream that contain method for writing java standard data type
FileInputStream	Input stream that reads from a file
FileOutputStream	Output stream that write to a file.
InputStream	Abstract class that describe stream input.
OutputStream	Abstract class that describe stream output.
PrintStream	Output Stream that contain print() and println() method

These classes define several key methods. Two most important are

1. read() : reads byte of data.
2. write() : Writes byte of data.

Character Stream Classes

Character stream is also defined by using two abstract class at the top of hierarchy, they are Reader and Writer.



These two abstract classes have several concrete classes that handle unicode character.

Some Important Charcter Stream Classes

Stream class	Description
BufferedReader	Handles buffered input stream.
BufferedWriter	Handles buffered output stream.
FileReader	Input stream that reads from file.
FileWriter	Output stream that writes to file.
InputStreamReader	Input stream that translate byte to character
OutputStreamReader	Output stream that translate character to byte.
PrintWriter	Output Stream that contain print() and println() method.
Reader	Abstract class that define character stream input
Writer	Abstract class that define character stream output

Q16. What is File class? Explain in detail?

Ans :

Files are a common source or destination of data in Java applications. Therefore this text will give you a brief overview of working with files in Java. It is not the intention to explain every technique in detail here, but rather to provide you with enough knowledge to decide on a file access method. Separate pages will describe each of these methods or classes in more detail, including examples of their usage etc.

Reading Files via Java IO

If you need to read a file from one end to the other you can use a FileInputStream or a FileReaderdepending on whether you want to read the file as binary or textual data. These two classes lets you read a file one byte or character at a time from the start to the end of the file, or read the bytes into an array of byte or char, again from start towards the end of the file. You don't have to read the whole file, but you can only read bytes and chars in the sequence they are stored in the file.

If you need to jump around the file and read only parts of it from here and there, you can use aRandomAccessFile.

If you need to jump around the file and read only parts of it from here and there, you can use aRandomAccessFile.

Writing File via Java IO

If you need to write a file from one end to the other you can use a FileOutputStream or a FileWriterdepending on whether you need to write binary data or characters. You can write a byte or character at a time from the beginning to the end of the file, or write arrays of byte and char. Data is stored sequentially in the file in the order they are written.

If you need to skip around a file and write to it in various places, for instance appending to the end of the file, you can use a RandomAccessFile.

Random Access to Files via Java IO

As I have already mentioned, you can get random access to files with Java IO via the RandomAccessFileclass.

Random access doesn't mean that you read or write from truly random places. It just means that you can skip around the file and read from or write to it at the same time in any way you want. No particular access sequence is enforced. This makes it possible to overwrite parts of an existing file, to append to it, delete from it, and of course read from the file from wherever you need to read from it.

File and Directory Info Access

Sometimes you may need access to information about a file rather than its content. For instance, if you need to know the file size or the file attributes of a file. The same may be true for a directory. For instance, you may want to get a list of all files in a given directory. Both file and directory information is available via the File class.

Java IO: InputStream

he InputStream class is the base class (superclass) of all input streams in the Java IO API. InputStreamSubclasses include the FileInputStream, BufferedInputStream and thePushbackInputStream among others. To see a full list of InputStream subclasses, go to the bottomtable of the Java IO Overview page.

InputStreams and Sources

An InputStream is typically always connected to some data source, like a file, network connection, pipe etc. This is also explained in more detail in the Java IO Overview text.

Java InputStream Example

Java InputStream's are used for reading byte based data, one byte at a time.

This example creates a new FileInputStream instance. FileInputStream is a subclass of InputStream so it is safe to assign an instance of FileInputStream to an InputStream variable (the inputstream variable).

Note: The proper exception handling has been skipped here for the sake of clarity. To learn more about correct exception handling, go to Java IO Exception Handling.

From Java 7 you can use the try-with-resources construct to make sure the InputStream is properly closed after use.

Once the executing thread exits the try block, the inputstream variable is closed.

read()

The read() method of an InputStream returns an int which contains the byte value of the byte read. Here is an InputStream read()

Example:

```
int data = inputstream.read();
```

You can case the returned int to a char like this:

```
char aChar = (char) data;
```

Subclasses of InputStream may have alternative read() methods. For instance, theDataInputStream allows you to read Java primitives like int, long, float, double, boolean etc. with its corresponding methods readBoolean(), readDouble() etc.

End of Stream

If the read() method returns -1, the end of stream has been reached, meaning there is no more data to read in the InputStream. That is, -1 as int value, not -1 as byte or short value. There is a difference here!

When the end of stream has been reached, you can close the InputStream.

read(byte[])

The InputStream class also contains two read() methods which can read data from the InputStream's source into a byte array. These methods are:

- int read(byte[])
- int read(byte[], int offset, int length)

Reading an array of bytes at a time is much faster than reading one byte at a time, so when you can, use these read methods instead of the read() method.

The read(byte[]) method will attempt to read as many bytes into the byte array given as parameter as the array has space for. The read(byte[]) method returns an int telling how many bytes were actually read. In case less bytes could be read from the InputStream than the byte array has space for, the rest of the bytearray will contain the same data as it did before the read started. Remember to inspect the returned int to see how many bytes were actually read into the byte array.

The read(byte[], int offset, int length) method also reads bytes into a byte array, but starts at offsetbytes into the array, and reads a maximum of length bytes into the array from that position. Again, the read(byte[], int offset, int length) method returns an int telling how many bytes were actually read into the array, so remember to check this value before processing the read bytes.

For both methods, if the end of stream has been reached, the method returns -1 as the number of bytes read.

First this example create a byte array. Then it creates an int variable named bytesRead to hold the number of bytes read for each read(byte[]) call, and immediately assigns bytesRead the value returned from the first read(byte[]) call.

Inside the while loop the do SomethingWithData() method is called, passing along the data byte array as well as how many bytes were read into the array as parameters. At the end of the while loop data is read into the byte array again.

It should not take much imagination to figure out how to use the read(byte[], int offset, int length)method instead of read(byte[]). You pretty much just replace the read(byte[]) calls with read(byte[], int offset, int length) calls.

mark() and reset()

The InputStream class has two methods called mark() and reset() which subclasses of InputStream may or may not support.

If an InputStream subclass supports the mark() and reset() methods, then that subclass should override the markSupported() to return true. If the markSupported() method returnsfalsethenmark()and reset() are not supported.

The mark() sets a mark internally in the InputStream which marks the point in the stream to which data has been read so far. The code using the InputStream can then continue reading data from it. If the code using the InputStream wants to go back to the point in the stream where the mark was set, the code calls reset()on the InputStream. The InputStream then “rewinds” and go back to the mark, and start returning (reading) data from that point again. This will of course result in some data being returned more than once from the InputStream.

The methods mark() and reset() methods are typically used when implementing parsers. Sometimes a parser may need to read ahead in the InputStream and if the parser doesn’t find what it expected, it may need to rewind back and try to match the read data against something else.

3.4.2 File Streams

3.4.2.1 FileInputStream Class, FileOutputStream Class

Q17. Explain in detail about FileInputStream Class with an example.

Ans : **(Dec.-19, July-19, Dec.-18, KU)**

The FileInputStream class makes it possible to read the contents of a file as a stream of bytes.

The FileInputStream class is a subclass of InputStream. This means that you use theFileInputStream as an InputStream (FileInputStream behaves like an InputStream).

FileInputStream Constructors

The FileInputStream class has a three different constructors you can use to create a FileInputStreaminstance. I will cover the first two here.

The first constructor takes a String as parameter. This String should contain the path in the file system to where the file to read is located.

Notice the path String. It needs double backslashes (\\) to create a single backslash in the String, because backslash is an escape character in Java Strings. To get a single backslash you need to use the escape sequence \\.

Notice the use of the for-slash (the normal slash character) as directory separator. That is how you write file paths on unix. Actually, in my experience Java will also understand if you use a / as directory separator on Windows (e.g. c:/user/data/thefile.txt), but don't take my word for it. Test it on your own system!

Which of the constructors you should use depends on what form you have the path in before opening the FileInputStream. If you already have a String or File, just use that as it is. There is no particular gain in converting a String to a File, or a File to a String first.

read()

The read() method of a FileInputStream returns an int which contains the byte value of the byte read. If the read() method returns -1, there is no more data to read in the FileInputStream, and it can be closed. That is, -1 as int value, not -1 as byte value. There is a difference here!

You use the read() method just like the read() method of an InputStream.

read(byte[])

Being an InputStream the FileInputStream also has two read() methods which can read data into a bytearray. You can read more about them in my tutorial about the InputStream (see link elsewhere in this text).

close()

Just like any other InputStream a FileInputStream needs to be closed properly after use. You do that by calling the FileInputStream's close() method. You can see that in the FileInputStream example further up this text, and remember to check the Java IO exception handling tutorial for more information about proper exception handling (link below the example further up the page).

Java FileInputStream class methods

Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Program

```
import java.io.FileInputStream;
public class DataStreamExample
{
        public static void main(String args[])
        {
        Try
              {
              FileInputStream fin=new FileInputStream("D:\\testout.txt");
              int i=fin.read();
              System.out.print((char)i);
              fin.close();
              }
              catch(Exception e)
              {
                     System.out.println(e);
              }
        }
}
```

Q18. Explain in detail about FileOutputStream Class with an example.

Ans : **(Dec.-19, July-19, Dec.-18, KU)**

The FileOutputStream class makes it possible to write a file as a stream of bytes. The FileOutputStreamclass is a subclass of OutputStream meaning you can use a FileOutputStream as an OutputStream.

Note: The proper exception handling has been skipped here for the sake of clarity. To learn more about correct exception handling, go to Java IO Exception Handling.

FileOutputStream Constructors

The FileOutputStream class contains a set of different useful constructors. I will cover the most commonly used constructors here.

The first constructor takes a String which contains the path of the file to write to. Here is anNoticethe path String. It needs double backslashes (\\) to create a single backslash in the String, because backslash is an escape character in Java Strings. To get a single backslash you need to use the escape sequence \\.

Overwriting vs. Appending the File

When you create a FileOutputStream pointing to a file that already exists, you can decide if you want to overwrite the existing file, or if you want to append to the existing file. You decide that based on which of the FileOutputStream constructors you choose to use.

This constructor which takes just one parameter, the file name, will overwrite any existing file:

OutputStream output = new FileOutputStream("c:\\data\\output-text.txt");

There is a constructor that takes 2 parameters too: The file name and a boolean. The boolean indicates whether to append or overwrite an existing file. Here are two examples:

```
    OutputStream output = new FileOutputStream("c:\\data\\output-text.txt", true); //appends to file
    OutputStream output = new FileOutputStream("c:\\data\\output-text.txt", false); //overwrites file
```

write()

The write() method of a FileOutputStream takes an int which contains the byte value of the byte to write.

The FileOutputStream has other constructors too, letting you specify the file to write to in different ways. Look in the official JavaDoc for more detailed info.

Writing Byte Arrays

Since the FileOutputStream is a subclass of OutputStream, you can write arrays of bytes to the FileOutputStream too, instead of just a single byte at a time. See my tutorial about the OutputStream classfor more information about how to write arrays of bytes.

flush()

When you write data to a FileOutputStream the data may get cached internally in the memory of the computer and written to disk at a later time. For instance, every time there is X amount of data to write, or when the FileOutputStream is closed.

If you want to make sure that all written data is written to disk without having to close the FileOutputStreamyou can call its flush() method. Calling flush() will make sure that all data which has been written to the FileOutputStream so far, is fully written to disk too.

close()

Like any other OutputStream a FileOutputStream instance needs to be closed after use. You do so by calling its close() method as shown in the example earlier in this text.

FileOutputStream class methods

Method	Description
protected void finalize()	It is sued to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Program

```
import java.io.FileOutputStream;
public class FileOutputStreamExample
{
        public static void main(String args[])
        {
         try
```

```
            {
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success...");
            }
            catch(Exception e){System.out.println(e);}
        }
}
```

3.4.3 Scanner Class

Q19. What is a scanner class? What is it used for?

Ans :

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method		Description
public String next()	→	it returns the next token from the scanner.
public String nextLine()	→	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	→	it scans the next token as a byte.
public short nextShort()	→	it scans the next token as a short value.
public int nextInt()	→	it scans the next token as an int value.
public long nextLong()	→	it scans the next token as a long value.
public float nextFloat()	→	it scans the next token as a float value.
public double nextDouble()	→	it scans the next token as a double value.

Java Scanner Example to get input from console

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

Program

```
import java.util.Scanner;
class ScannerTest
{
        public static void main(String args[])
{
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter your rollno");
        int rollno=sc.nextInt();
        System.out.println("Enter your name");
        String name=sc.next();
        System.out.println("Enter your fee");
        double fee=sc.nextDouble();
        System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
        sc.close();
    }
}
```

Output:

```
        Enter your rollno
        111
        Enter your name
        Ratan
        Enter
        450000
        Rollno:111 name:Ratan fee:450000
```

Q20. Write a program to implement Scanner Class?

Ans :

```
import java.util.*;
public class ScannerTest2
{
    public static void main(String args[])
    {
        String input = "10 tea 20 coffee 30 tea biscuits";
        Scanner s = new Scanner(input).useDelimiter("\\s");
        System.out.println(s.nextInt());
        System.out.println(s.next());
        System.out.println(s.nextInt());
        System.out.println(s.next());
        s.close();
    }
}
```

Output:

```
10
tea
20
coffee
```

3.4.4 BufferedInputStream Class, BufferedOutputStream Class

Q21. Explain about BufferedInputStream Class with an example.

Ans :

The BufferedInputStream class provides buffering to your input streams. Buffering can speed up IO quite a bit. Rather than read one byte at a time from the network or disk, the BufferedInput Stream reads a larger block at a time into an internal buffer. When you read a byte from the BufferedInputStream you are therefore reading it from its internal buffer. When the buffer is fully read, the BufferedInputStream reads another larger block of data into the buffer. This is typically much faster than reading a single byte at a time from an InputStream, especially for disk access and larger data amounts.

BufferedInputStream Example

To add buffering to an InputStream simply wrap it in a BufferedInputStream. Here is how that looks:

InputStream input = new BufferedInput Stream(new FileInputStream("c:\\data\\input-file.txt"));

As you can see, using a Buffered InputStream to add buffering to a non-buffered InputStream is pretty easy. The BufferedInputStream creates a byte array internally, and attempts to fill the array by calling the InputStream.read(byte[]) methods on the underlying InputStream.

Setting Buffer Size of a BufferedInputStream

You can set the buffer size to use internally by the BufferedInputStream. You provide the buffer size as a parameter to the Buffered Input Stream constructor, like this:

int bufferSize = 8 * 1024;

InputStreaminput = new BufferedInput Stream (new FileInputStream("c:\\data\\input-file.txt"), bufferSize);

This example sets the internal buffer used by the BufferedInputStream to 8 KB. It is best to use buffer sizes that are multiples of 1024 bytes. That works best with most built-in buffering in hard disks etc.

Except for adding buffering to your input streams, BufferedInputStream behaves exactly like an InputStream.

Optimal Buffer Size for a BufferedInput Stream

You should make some experiments with different buffer sizes to find out which buffer size seems to give you the best performance on your concrete hardware. The optimal buffer size may depend on whether you are using the BufferedInput Stream with a disk or network InputStream.

With both disk and network streams, the optimal buffer size may also depend on the concrete hardware in the computer. If the hard disk is anyways reading a minimum of 4KB at a time, it's stupid to use less than a 4KB buffer. It is also better to then use a buffer size that is a multiple of 4KB. For instance, using 6KB would be stupid too.

Even if your disk reads blocks of e.g. 4KB at a time, it can still be a good idea to use a buffer that is larger than this. A disk is good at reading data sequentially - meaning it is good at reading multiple blocks that are located after each other. Thus, using a 16KB buffer, or a 64KB buffer (or even larger) with a BufferedInputStream may still give you a better performance than using just a 4KB buffer.

Also keep in mind that some hard disks have a read cache of some mega bytes. If your hard disk anyways reads, say 64KB, of your file into its internal cache, you might as well get all of that data into your BufferedInputStream using one read operation, instead of using multiple read operations. Multiple read operations will be slower, and you risk that the hard disk's read cache gets erased between read operations, causing the hard disk to re-read that block into the cache.

To find the optimal BufferedInputStream buffer size, find out the block size your hard disk reads in, and possibly also its cache size, and make the buffer a multiple of that size. You will definitely have to experiment to find the optimal buffer size. Do so by measuring read speeds with different buffer sizes.

mark() and reset()

An interesting aspect to note about the BufferedInputStream is that is supports the mark() and reset()methods inherited from the InputStream. Not all InputStream subclasses support these methods. In general you can call the markSupported() method to find out if mark() and reset() are supported on a given InputStream or not, but the BufferedInputStream supports them.

Java BufferedInputStream class methods

Method	Description
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.
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

```
import java.io.*;
public class BufferedInputStreamExample
{
        public static void main(String args[])
        {
              try
              {
                    FileInputStream fin=new FileInputStream("D:\\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);
                }
        }
}
```

Q22. Explain about BufferedOutputStream Class with an example.

Ans :

The BufferedOutputStream class provides buffering to your output streams. Buffering can speed up IO quite a bit. Rather than write one byte at a time to the network or disk, you write a larger block at a time. This is typically much faster, especially for disk access and larger data amounts.

To add buffering to your OutputStream's simply wrap them in a BufferedOutputStream. Here is how that looks:

OutputStream output = new BufferedOutputStream(new FileOutputStream("c:\\data\\output-file.txt"));

Setting Buffer Size of a BufferedOutputStream

You can set the buffer size to use internally by the BufferedOutputStream. You provide the size as a constructor parameter, like this:

int bufferSize = 8 * 1024;

OutputStream output = new BufferedOutputStream(new FileOutputStream("c:\\data\\output-file.txt"),bufferSize);

Except for adding buffering to your input streams, BufferedOutputStream behaves exactly like an OutputStream. The only difference is that you may need to call flush() if you need to be absolutely sure that the data written until now is flushed out of the buffer and onto the network or disk.

Optimal Buffer Size for a BufferedOutputStream

You should make some experiments with different buffer sizes to find out which buffer size seems to give you the best performance on your concrete hardware. The optimal buffer size may depend on whether you are using the BufferedOutputStream with a disk or networkOutputStream.

With both disk and network streams, the optimal buffer size may also depend on the concrete hardware in the computer. If the hard disk is anyways writing a minimum of 4KB at a time, it's stupid to use less than a 4KB buffer. It is also better to then use a buffer size that is a multiple of 4KB. For instance, using 6KB would be stupid too.

Even if your disk writes blocks of e.g. 4KB at a time, it can still be a good idea to use a buffer that is larger than this. A disk is good at writing data sequentially - meaning it is good at writing multiple blocks that are located after each other. Thus, using a 16KB buffer, or a 64KB buffer (or even larger) with a BufferedOutputStream may still give you a better performance than using just a 4KB buffer.

To find the optimal BufferedOutputStream buffer size, find out the block size your hard disk writes in, and make the buffer a multiple of that size. You will definitely have to experiment to find the optimal buffer size. Do so by measuring write speeds with different buffer sizes.

Java BufferedOutputStream class methods

Method	Description
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

```
import java.io.*;
public class BufferedOutputStreamExample
{
        public static void main(String args[])throws Exception
        {
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            BufferedOutputStream bout=new BufferedOutputStream(fout);
            String s="Welcome to Rahul Publication.";
            byte b[]=s.getBytes();
            bout.write(b);
            bout.flush();
            bout.close();
            fout.close();
            System.out.println("success");
        }
}
```

Output

```
        Welcome to Rahul Publication.
```

3.4.5 RandomAccessFile Class

Q23. What is a RandomAccessFile Class? Write a program for reading / writing using random access file.

Ans : **(July-19)**

The RandomAccessFile class in the Java IO API allows you to move around a file and read from it or write to it as you please. You can replace existing parts of a file too. This is not possible with the FileInputStreamor FileOutputStream.

Creating a RandomAccessFile

Before you can work with the RandomAccess File class you must instantiate it. Here is how that looks:

RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");

Notice the second input parameter to the constructor: "rw". This is the mode you want to open file in. "rw" means read/write mode. Check the JavaDoc for more details about what modes you can open a RandomAccessFile in.

Moving Around a RandomAccessFile

To read or write at a specific location in a RandomAccessFile you must first position the file pointer at the location to read or write. This is done using the seek() method. The current position of the file pointer can be obtained by calling the getFilePointer() method.

Here is a simple example:

```
RandomAccessFile file = new RandomAccessFile("c:\\data\\file.txt", "rw");
file.seek(200);
long pointer = file.getFilePointer();
file.close();
```

Reading from a RandomAccessFile

Reading from a RandomAccessFile is done using one of its many read() methods. Here is a simple

The read() method reads the byte located a the position in the file currently pointed to by the file pointer in the RandomAccessFile instance.

Here is a thing the JavaDoc forgets to mention: The read() method increments the file pointer to point to the next byte in the file after the byte just read! This means that you can continue to call read() without having to manually move the file pointer.

Writing to a RandomAccessFile

Writing to a RandomAccessFile can be done using one it its many write() methods.

Just like with the read() method the write() method advances the file pointer after being called. That way you don't have to constantly move the file pointer to write data to a new location in the file.

close()

The RandomAccessFile has a close() method which must be called when you are done using the RandomAccessFile instance. You can see example of calls to close() in the examples above.

RandomAccessFile Exception Handling

The proper exception handling of a RandomAccessFile is left out of this text for clarity. However, a RandomAccessFile must be closed properly after use, just like with a stream or reader / writer. This requires proper exception handling around the close() call.

Short Question and Answers

1. What is exception?

Ans :

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

In exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

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.

Based on these, we have three categories of Exceptions. You need to understand them to know how exception handling works in Java.

2. User-Defined Exceptions

Ans :

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called 'user-defined Exceptions'.

Following steps are followed for the creation of user-defined Exception.

- The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:
- class MyException extends Exception
- We can write a default constructor in his own exception class.

3. Write about Multithreading in Java.

Ans :

Multithreading in java is a process of executing multiple threads simultaneously.

Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

4. FileInputStream

Ans :

The FileInputStream class has a three different constructors you can use to create a FileInputStreaminstance. I will cover the first two here.

The first constructor takes a String as parameter. This String should contain the path in the file system to where the file to read is located.

Notice the path String. It needs double backslashes (\\) to create a single backslash in the String, because backslash is an escape character in Java Strings. To get a single backslash you need to use the escape sequence \\.

Notice the use of the for-slash (the normal slash character) as directory separator. That is how you write file paths on unix. Actually, in my experience Java will also understand if you use a / as directory separator on Windows (e.g. c:/user/data/thefile.txt), but don't take my word for it. Test it on your own system!

Which of the constructors you should use depends on what form you have the path in before opening the FileInputStream. If you already have a String or File, just use that as it is. There is no particular gain in converting a String to a File, or a File to a String first.

5. FileOutputStream Class

Ans :

The FileOutputStream class makes it possible to write a file as a stream of bytes. The FileOutputStreamclass is a subclass of OutputStream meaning you can use a FileOutputStream as an OutputStream.

FileOutputStream Constructors

The FileOutputStream class contains a set of different useful constructors. I will cover the most commonly used constructors here.

The first constructor takes a String which contains the path of the file to write to. Here is anNoticethe path String. It needs double backslashes (\\) to create a single backslash in the String, because backslash is an escape character in Java Strings. To get a single backslash you need to use the escape sequence \\.

Overwriting vs. Appending the File

When you create a FileOutputStream pointing to a file that already exists, you can decide if you want to overwrite the existing file, or if you want to append to the existing file. You decide that based on which of the FileOutputStream constructors you choose to use.

6. Define thread.

Ans :

Thread can be defined as a set of executable instructions that are executed independently. A program can be divided into multiple subprograms and each subprogram is called a thread. Every individual thread is executed separately, thereby decreasing the execution time of a program.

7. How do we set priorities for threads?

Ans :

The priority of a thread can be set by making use of setPriority( ) method. This method is a member of instead class. The syntax for setPriority( ) is as follows :

final void setPriority(int priority)

Here,

final is a keyword

void is the return type

setPriority is the method name used for setting the priority

priority is an integer value which can store the priority of thread.

8. Synchronization

Ans :

Synchronization of threads ensures that if two or more threads need to access a shared resource then that resource is used by only one thread at a time. You can synchronize your code using the synchronized keyword. You can invoke only one synchronized method for an object at any given time.

Synchronization is based on the concept of monitor. A monitor, also known as a semaphore, is an object that is used as a mutually exclusive lock. All objects and classes are associated with a monitor and only one thread can won a monitor at a given time.

The monitor controls the way in which synchronized methods access an object or class. When a thread acquires a lock, it is said to have entered the monitor. The monitor ensures that only one thread has access to the resources at any given time. To enter an object's monitor, you need to call a synchronized method.

When a thread is within a synchronized method, all the other threads that try to call it on the same instance have to wait. During the execution of a synchronized method, the object is locked so that no other synchronized method can be invoked. The monitor is automatically released when the method completes its execution. The monitor can also be released when the synchronized method executes the wait() method. When a thread calls the wait() method, it temporarily releases the locks that it holds.

9. Benefits of Exception Handling.

Ans :

(i) It can control run-time errors that occurs in the program.

(ii) It can avoid abnormal termination of the program and also shows the behavior of program to users.

(iii) It can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception.

(iv) It can separate the error handling code and normal code by using try-catch block.

(v) It can produce the normal execution flow for a program.

(vi) Its mechanism can implement a clean way to propagate error. That is when an invoking method cannot manage a particular situation, then it throws an exception and asks the invoking method to deal with such situation.

(vii) Its mechanism develops a powerful coding that ensures that the exceptions can be prevented.

10. Exception Handling

Ans :

Exception handling can be defined as a mechanism of handling exceptions that can be occurred at run-time. This mechanism is commonly used to prevent the malfunctions such as computer deadlock or computer hanging. Some examples of run-time exception are divide by zero, array out of bounds exceptions.

The run-time exception can be handled by using five keywords namely try, catch, throw, throws and finally. The code that can generate exception is usually placed in try block. If an exception is occurred, execution flow finds the similar catch block and leaves the try block. The catch block handles the exception and generates a message specifying the type of exception. Some of these exception types are as follows,

(i) ArithmeticException

(ii) ArrayIndexOutOfBoundsException.

However, there is no rule that the try block must contain corresponding catch block. Each try block may contain zero or multiple catch blocks.

When an exception is thrown and if it is matched with any of the catch block, then the statements of corresponding catch block can be executed. Otherwise, the catch block can be skipped and the statements present next to the catch block will be executed. The finally block is declared after all the catch blocks whose code can be executed irrespective to the occurrence of exception. Note that, the finally block is optional.

11. Multitasking

Ans :

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

➢ Process-based Multitasking(Multiprocessing)

➢ Thread-based Multitasking(Multithreading)

Process-based Multitasking (Multiprocessing)

➢ Each process have its own address in memory i.e. each process allocates separate memory area.

- Process is heavy weight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.

12. What is a scanner class?

Ans :

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them.

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Choose the Correct Answers

1. Which of the following stores all the standard java classes [b]
   (a) lang (b) java
   (c) util (d) java package
2. Which of these keyword is used to define package [c]
   (a) pkg (b) pack age
   (c) package (d) None
3. Access specifier of the package ________ [a]
   (a) Public (b) Private
   (c) Protected (d) All
4. Which of the following is correct [c]
   (a) import package_name (b) import package
   (c) Import package_name.* (d) Import package.*
5. When does exception occur in java [a]
   (a) Runtime (b) Compile Time
   (c) Any Time (d) None
6. Which of these is not a part of Exception [c]
   (a) try (b) catch
   (c) throw (d) throws
7. What is the MIN_PRIORITY and MAX _PRIORITY in threads [b]
   (a) 0 to 256 (b) 1 & 10
   (c) 0 to 10 (d) 10 to 0
8. Which of these interfaces is implemented by threads class [a]
   (a) runnable (b) connections
   (c) set (d) map
9. Which of these class is superclass of StringBuffer class ________ [b]
   (a) java.util (b) java.lang
   (c) Arraylist (d) String class
10. Which data type value is returned by equal( )of string class ________ [c]
   (a) char (b) int
   (c) boolean (d) none

Fill in the blanks

1. A _________ is defined as a group of related classes and interfaces.
2. Once _________ is created a string object cannot be changed.
3. _________ is a mechanism to convert primitive into object and vice versa.
4. _________ is abnormal condition in Java.
5. Each catch block is an _________.
6. _________ is small piece of code that execute in a sequence.
7. _________ ensure that same resource is shared by each thread (one or more).
8. Java performs input/output operations through _________.
9. _________ is the predefined class used for reading data dynamically.
10. _________ method is used to read character in BufferedReader.
11. The priority values ranges from _________.
12. A _________ thread runs in the background of a program.

ANSWERS

1. Package
2. String class
3. Wrapper class
4. Exception
5. Exception Handling
6. Thread
7. Synchronization
8. Streams
9. Scanner Class
10. read()
11. 1 to 10
12. Daemon

One Mark Answers

1. Exception.

Ans :

The exception handling in java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

2. User-Defined Exception.

Ans :

An exception which can be created by the user manually are called as user-defined exception.

3. Multithreading in Java.

Ans :

Multithreading in java is a process of executing multiple threads simultaneously. Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java.

4. Main Thread.

Ans :

The main thread is the first thread that will begin its execution immediately when the Java program starts up.

5. Synchronization

Ans :

Synchronization of threads ensures that if two or more threads need to access a shared resource then that resource is used by only one thread at a time.

6. FileOutputStream Class

Ans :

The FileOutputStream class makes it possible to write a file as a stream of bytes. The FileOutputStreamclass is a subclass of OutputStream meaning you can use a FileOutputStream as an OutputStream.

7. What is a scanner class

Ans :

The Java Scanner class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

UNIT IV

Applets: Introduction, Example, Life Cycle, Applet Class, Common Methods Used in Displaying the Output (Graphics Class).

Event Handling: Introduction, Types of Events, Example.

AWT: Introduction, Components, Containers, Button, Label, Checkbox, Radio Buttons, Container Class, Layouts.

Swings: Introduction, Differences between Swing and AWT, JFrame, JApplet, JPanel, Components in Swings, Layout Managers, JTable.

4.1 Applets

4.1.1 Introduction, Example

Q1. Explain the concept of Applets with an example.

Ans :

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following.

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

Example

```
import j ava.awt. *;
import java.applet.*;
/*
<applet code="Demo" width=500 height
=500>
</applet>
*/
public class Demo extends Applet
{
public void paint(Graphics g)
{
g.drawString("Welcome", 20, 30);
}
}
```

4.1.2 Life Cycle

Q2. Explain applet life cycle.

(OR)

Explain the sequence of applet's life cycle.

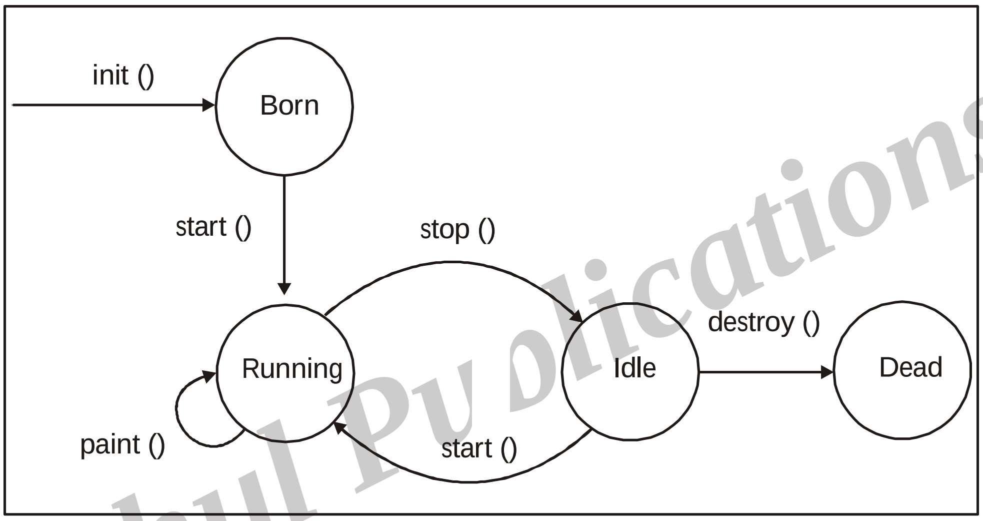
(OR)

Write about applet's life cycle.

Ans : **(July-21, Dec.-18, Dec.-18, KU)**

Four methods in the Applet class gives you the framework on which you build any serious applet.

- **init** - This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** - This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** - This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** - This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** - Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.



4.1.3 Applet Class

Q3. Write about Applet Classes?

Ans :

Every applet is an extension of the java.applet.Applet class. The base Applet class provides methods that a derived Applet class may call to obtain information and services from the browser context.

These include methods that do the following

- Get applet parameters
- Get the network location of the HTML file that contains the applet
- Get the network location of the applet class directory
- Print a status message in the browser
- Fetch an image
- Fetch an audio clip
- Play an audio clip
- Resize the applet

Additionally, the Applet class provides an interface by which the viewer or browser obtains information about the applet and controls the applet's execution. The viewer may-

- Request information about the author, version, and copyright of the applet
- Request a description of the parameters the applet recognizes
- Initialize the applet
- Destroy the applet
- Start the applet's execution
- Stop the applet's execution

The Applet class provides default implementations of each of these methods. Those implementations may be overridden as necessary.

The "Hello, World" applet is complete as it stands. The only method overridden is the paint method.

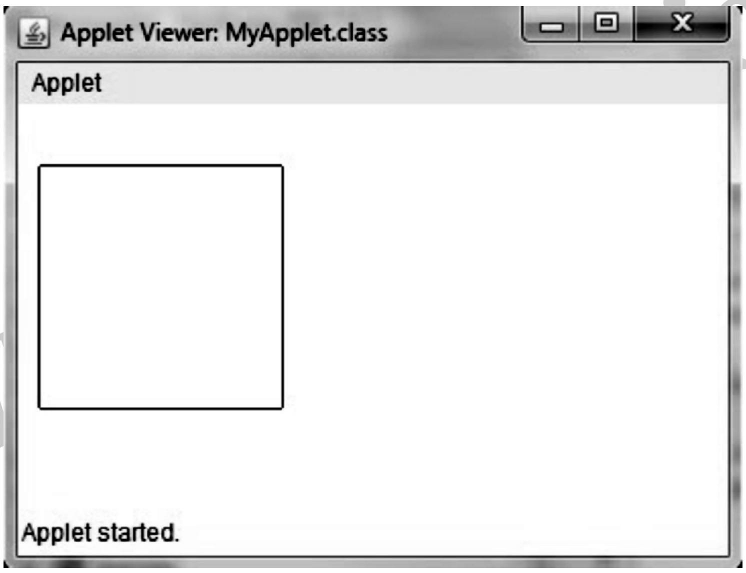
Most applets override these four methods. These four methods forms Applet lifecycle.

- **init() :** init() is the first method to be called. This is where variable are initialized. This method is called only once during the runtime of applet.
- **start() :** start() method is called after init(). This method is called to restart an applet after it has been stopped.
- **stop() :** stop() method is called to suspend thread that does not need to run when applet is not visible.
- **destroy() :** destroy() method is called when your applet needs to be removed completely from memory.

Syntax

```
import java.awt.*;

import java.applet.*;

public class AppletTest extends Applet

{

public void init()

{

     //initialization

}

public void start ()

{

     //start or resume execution
```

```
}
public void stop()
{
      //suspend execution
}
public void destroy()
{
      //perform shutdown activity
}
public void paint (Graphics g)
{
      //display the content of window
```



4.1.4 Applet Implementation

Q4. How to run an Applet Program? (Or) Explain how to implement an Applet?

Ans

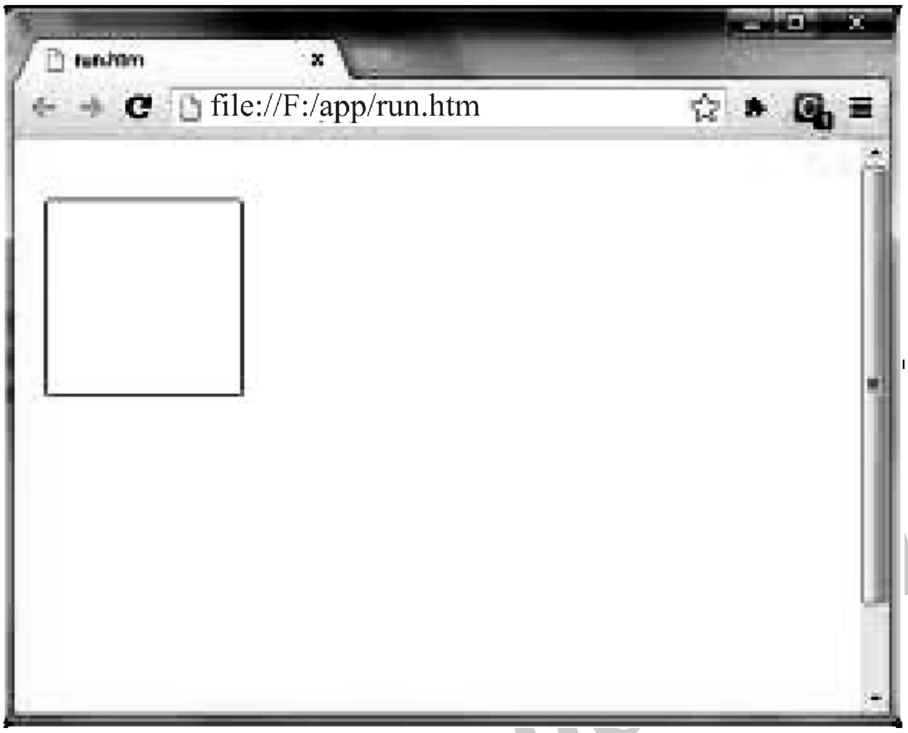
An Applet program is compiled in the same way as you have been compiling your console programs. However there are two ways to run an applet.

➢ Executing the Applet within Java-compatible web browser.

➢ Using an Applet viewer, such as the standard tool, applet viewer. An applet viewer executes your applet in a window

For executing an Applet in an web browser, create short **HTML file** in the same directory. Inside **body** tag of the file, include the following code. (**applet** tag loads the Applet class)

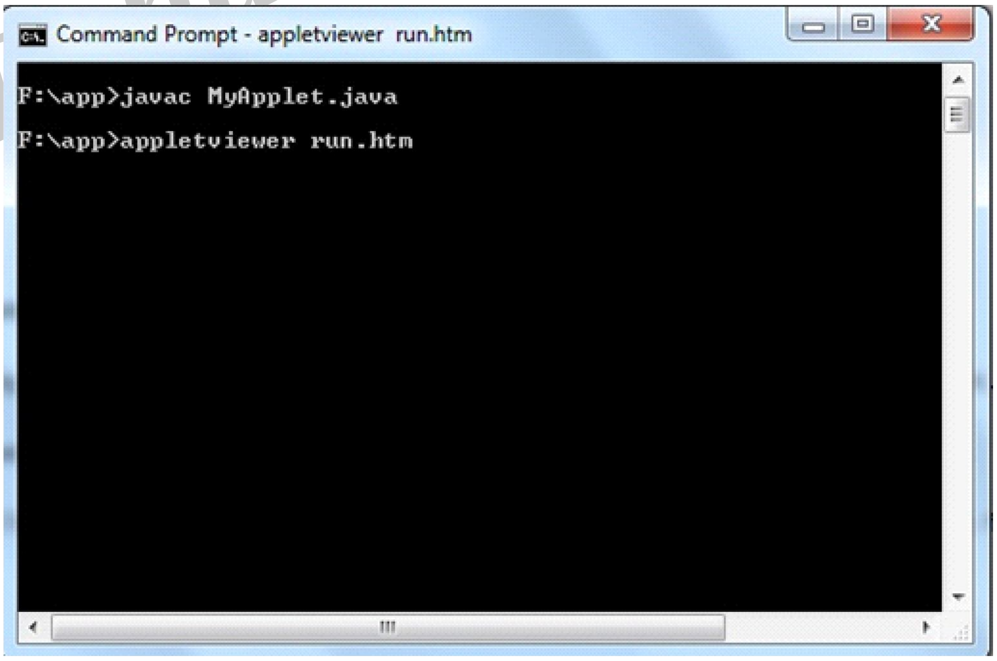
<applet code = “MyApplet” width=400 height=400 >< /applet > Run the HTML file



Running Applet using Applet Viewer

To execute an Applet with an applet viewer, write short HTML file as discussed above. If you name it as **run.htm**, then the following command will run your applet program.

f:/>appletviewer run.htm



```
F:\app>javac MyApplet.java

F:\app>appletviewer run.htm
```

Q5. How does Applet differ form Application?

Ans :

Although both the Applets and stand-alone applications are Java programs, there are certain restrictions are imposed on Applets due to securityconcerns.

There are some important differences between an applet and a standalone Java application, including the following-

- ➢ An applet is a Java class that extends the java.applet.Applet class.
- ➢ A main() method is not invoked on an applet, and an applet class will not define main().
- ➢ Applets are designed to be embedded within an HTML page.
- ➢ When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- ➢ A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- ➢ The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- ➢ Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.
- ➢ Other classes that the applet needs can be downloaded in a single Java Archive (JAR) file.

4.1.5 Common Methods Used in Displaying the Output (Graphics Class)

Q6. What are the Common Methods Used in Displaying the Output (Graphics Class)

Ans :

The common methods used in displaying the output are as follows,

1. **public abstract void drawString(String str, int x, int** y): This method is used to draw the given string.
2. **public void drawRect(int** x, **int** y, **int width, int height):** This method is used to draw a rectangle with the given width and height.
3. **public abstract void fillRect(int x, int** y, **int width, int height):** This method is used to fill rectangle with the default color and the given width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** This method is used to draw oval with the given width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** This method is used to fill oval with the default color and the given width and height.
6. **public abstract void drawLinefint xl, int yl, int x2, int y2):** This method is used to draw line between the given points i.e., (xl, yl) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** This method is used draw a particular image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** This method is used draw a circular or elliptical arc.
9. **public abstract void filLArc(int** x, **int** y, **int width, int height, int startAngle, int arcAngle):** This method is used to fill a circular or elliptical arc with a default color.
10. **public abstract void setColor(Color c):** This method is used to set the graphics current color to the given color
11. **public abstract void setFont(Font font):** This method is used to set the graphics current font to the given font

4.2 Event Handling

4.2.1 Introduction

Q7. Define the terms :

(a) Event

(b) Event Handling

Ans :

(a) Event

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. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

(b) Event Handling

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.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

➢ **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.

➢ **Listener** - It is also known as event handler.Listener is responsible for generating response to an event. From java implementation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

The benefit of this approach is that the user interface logic is completely separated from the logic that generates the event. The user interface element is able to delegate the processing of an event to the separate piece of code. In this model ,Listener needs to be registered with the source object so that the listener can receive the event notification. This is an efficient way of handling the event because the event notifications are sent only to those listener that want to receive them.

Q8. Explain various steps involved in even handling.

Ans : **(June-19(MGU)**

➢ The User clicks the button and the event is generated.

➢ Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.

➢ Event object is forwarded to the method of registered listener class.

➢ The method is now getting executed and returns.

➢ Points to remember about listener

➢ In order to design a listener class we have to develop some listener interfaces.These Listener interfaces forecast some public abstract callback methods which must be implemented by the listener class.

➢ If you do not implement the any if the predefined interfaces then your class can not act as a listener class for a source object.

4.2.2 Types of Events, Example

Q9. Explain the various types of event classes.

Ans :

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	ActionListener
MouseEvent	generated when mouse is dragged, moved,clicked,pressed or released.	MouseListener
KeyEvent	generated when input is received from keyboard	KeyListener
ItemEvent	generated when check-box or list item is clicked	ItemListener
TextEvent	generated when value of textarea or textfield is changed	TextListener
MouseWheelEvent	generated when mouse wheel is moved	MouseWheelListener
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	WindowListener
ComponentEvent	generated when component is hidden, moved, resized or set visible	ComponentEventListener
ContainerEvent	generated when component is added or removed from container	ContainerListener
AdjustmentEvent	generated when scroll bar is manipulated	AdjustmentListener
FocusEvent	generated when component gains or loses keyboard focus	FocusListener

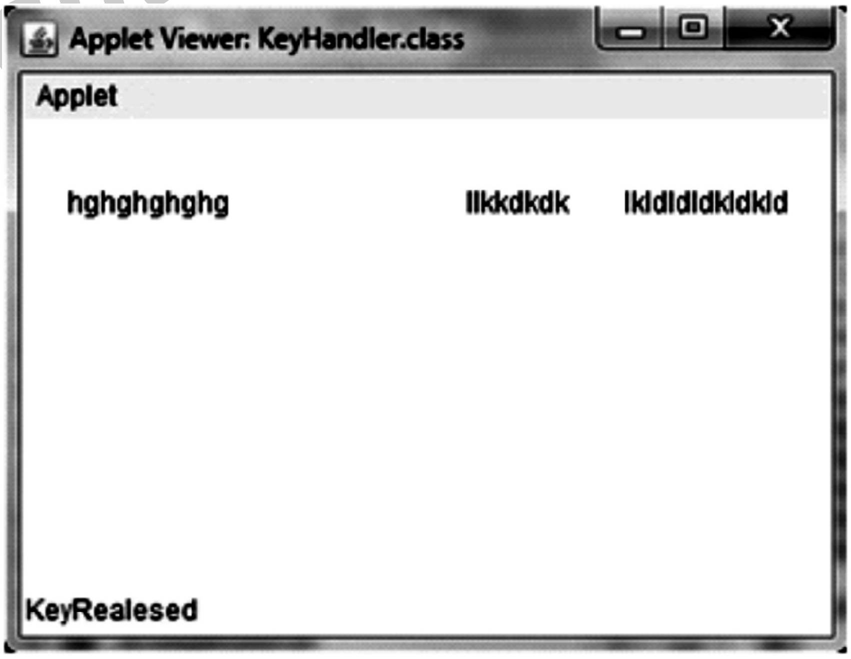
Q10. Write a program to implement Event Handling?

Ans :

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.applet.*;
import java.awt.event.*;
zimport java.awt.*;
public class Test extends Applet implements KeyListener
{
String msg="";
public void init()
{
    addKeyListener(this);
}
public void keyPressed(KeyEvent k)
{
      showStatus("KeyPressed");
}
public void keyReleased(KeyEvent k)
{
      showStatus("KeyRealesed");
}
public void keyTyped(KeyEvent k)
{
      msg = msg+k.getKeyChar();
      repaint();
}
public void paint(Graphics g)
{
      g.drawString(msg, 20, 40);
}
}
```

HTML code

```
<applet code="Test" width=300, height=100 >
```



```
hghghghghg                 llkkdkdk     lkldldldkldkld
```

```
KeyRealesed
```

4.3 AWT

4.3.1 Introduction

Q11. What are AWT's? Explain AWT Hierarchy in JAVA?

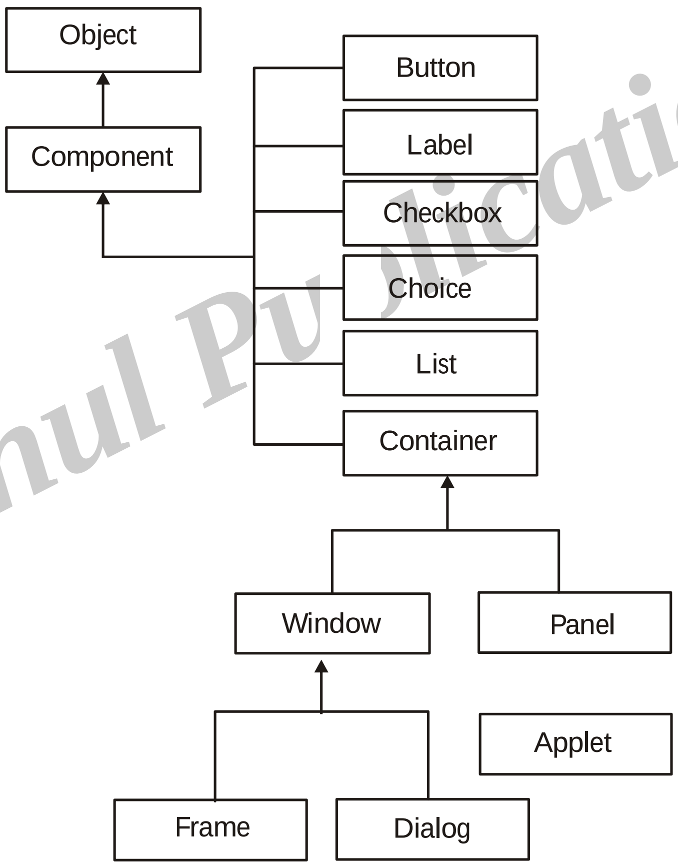
Ans : **(July-21)**

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



➢ **Container:** The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

➢ **Window:** The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

➢ **Panel:** The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

➢ **Frame :**The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

➢ By extending Frame class (inheritance)

➢ By creating the object of Frame class (association)

AWT Example by Inheritance

```
import java.awt.*;
class First extends Frame
{
First()
{
    Button b=new Button("click me");
    b.setBounds(30,100,80,30);// setting button position
    add(b);//adding button into frame
    setSize(300,300);//frame size 300 width and 300 height
    setLayout(null);//no layout manager
    setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[])
{
    First f=new First();
}
}
```

AWT Example by Association

```
    import java.awt.*;
class First2
```

```
{
        First2()
        {
        Frame f=new Frame();
        Button b=new Button("click me");
        b.setBounds(30,50,80,30);
        f.add(b);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
        }
        public static void main(String args[])
        {
        First2 f=new First2();
        }
}
```

Q12. What is the purpose of AWT?

Ans : **(July-21)**

Unlike C/C++, Java also supports GUI environment. Through the use of GUI (Graphical User Interface) environments it is possible to take user input into a running program or it can display some message to the user dynamically. The user interfaces can either be command line or GUI. All the classes and interfaces required for developing GUI components and drawing graphics are provided by two packages i.e., java, awt and java.awt.event.

4.3.2 Components

Q13. What are the Components of AWT?

(OR)

Explain various AWT classes used in java.

Ans : **(Dec.-19)**

The class Component is the abstract base class for the non menu user-interface controls of AWT. Component represents an object with graphical representation.

Class declaration

Following is the declaration for java. awt. Component class:

public abstract class Component extends

ObjectimplementsImage Observer, Menu Container, Serializable

Field

Following are the fields for java. awt. Component class:

- **static float BOTTOM_ALIGNMENT** — Ease-of-use constant for getAlignmentY.
- **static float CENTER_ALIGNMENT** — Ease-of-use constant for getAlignmentY and getAlignmentX.
- **static float LEFT_ALIGNMENT** — Ease-of-use constant for getAlignmentX.
- **static float RIGHT_ALIGNMENT** — Ease-of-use constant for getAlignmentX.
- **static float TOP_ALIGNMENT** — Ease-of-use constant for getAlignmentY().

Class constructors

S.No.	Cnstructor & Description
1	**Protected Componen():** This creates a new Component.

Methods inherited

This class inherits methods from the following classes: **java.lang.Object**

After having the basic idea of GUI, let us know how many Java AWT Components classes exist with **java.awt** package.

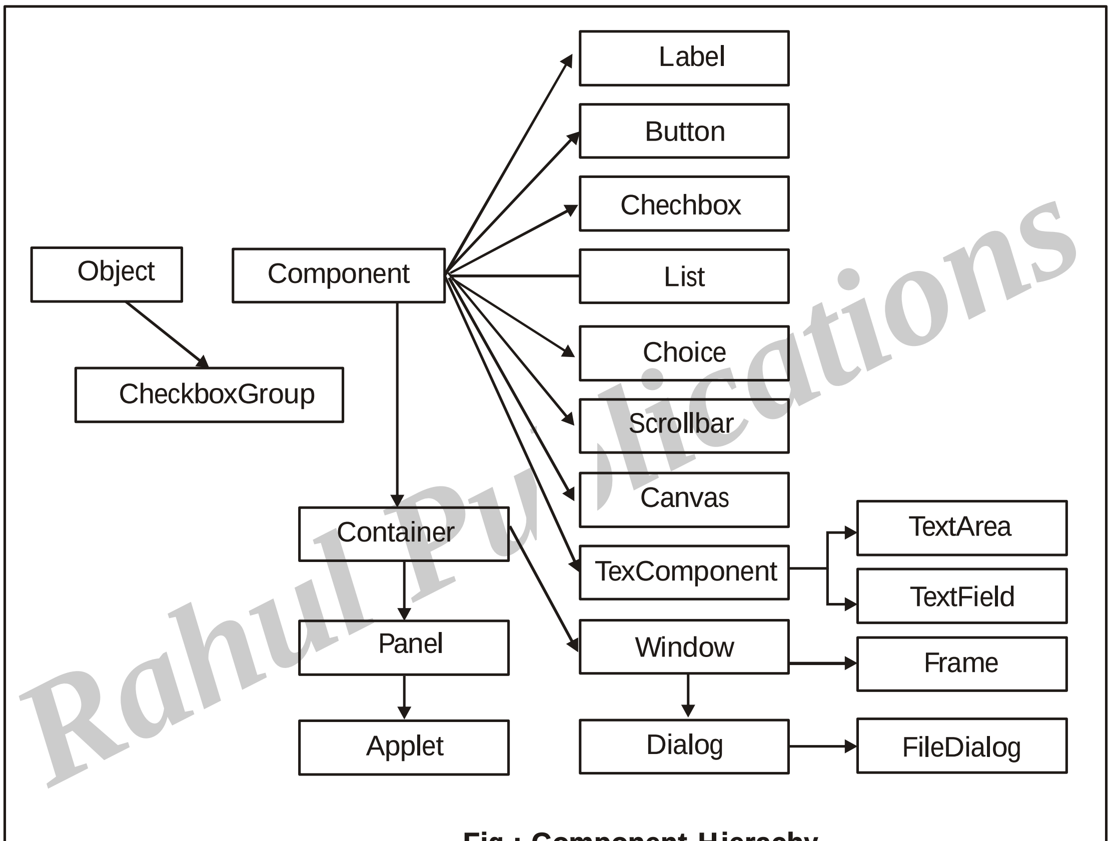


Fig.: Component Hierachy

As you can observe from the above hierarchy, Component is the super class of all Java components and is declared as abstract. That is, we cannot create objects of Component class directly.

Following is the class signature

public abstract class Component extends Object implements ImageObserver, Menu Container, Serializable

Properties of Java AWT Components

1. A Component object represents a graphical interactive area displayable on the screen that can be used by the user.
2. Any subclass of a Component class is known as a component. For example, button is a component.
3. Only components can be added to a container, like frame.

Important methods of Component class

Following are the important methods, practiced very often, of Component class that can be used by all the sub classes (components).

1. **setEnabled(boolean):** The parameter false makes the component disabled so that it does not respond to user interactions (say, clicks).
2. **setBounds():** Using this method, the programmer can give his own size to the component in terms of width and height and also the location where he can place the component in the container. This method is not used often as the programmer prefers to place the component in the container using layout managers.
3. **getWidth():** The programmer can obtain the width of the component.
4. **getHeight():** The programmer can obtain the height of the component.
5. **paint():** Used very extensively to draw graphics. This method is called implicitly when the frame is created or resized.
6. **repaint():** Used by the programmer to call the paint() method explicitly anywhere in the program.
7. **setBackground():** Used to give a background color to a component.
8. **setForeground():** Used to give a foreground color to a component.
9. **setFont():** Used to give a font to a component to display the text.
10. **getSize():** Returns the size of the component.
11. **setSize():** Used to give the size to the component.
12. **update():** The is method is called implicitly by the repaint(). A call to this method clears the earlier drawings existing on the container.

4.3.3 Containers

Q14. Discuss about various containers of AWT.

Ans : **(Dec.-18(MGU)**

The class Container is the super class for the containers of AWT. Container object can contain other AWT components.

Class declaration

Following is the declaration for java. awt. Container class:

public class Container extends Component

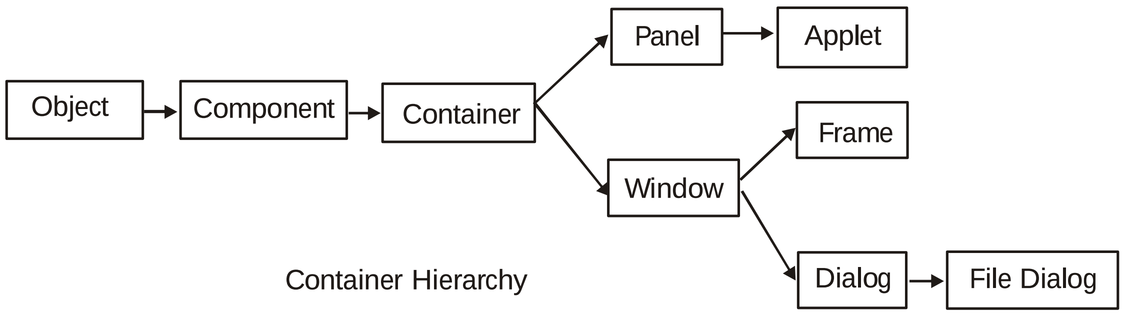
Class constructors

S.No.	Constructor & Description
1	**Container():** This creates a new Container

After knowing what a component is, let us go to the list of containers Java supports. For this, the container hierarchy is separated from the earlier component hierarchy and displayed here.

Infact, the Container is a component as it is a subclass of Component class. It can use all the methods of Component class and an extra added feature is components can be added to this component (container).

1. Any class subclass of a Container class is known as a container.
2. Only components can be added to container.
3. Every container comes with a default layout manager that can be changed explicitly with setLayout() method.



Container Hierarchy

As you can observe from the above hierarchy, Container is the super class of all Java containers.

Following is the class signature

public class Container extends Component

Following are the important methods of the Container class.

1. **add():** This method is overloaded with which a component can be added to the container.
2. **invalidate():** Used to invalidate the present set up of components in the container.
3. **validate():** Used to revalidate the current set up of components after calling invalidate().

The important containers used very often are applets, frames and panels. Rare are dialog boxes. Panels work very differently. Panel behaves both as component and container. As component panel can be added to a container and as container we can add components to panel.

Types of containers

The AWT provides four container classes. They are class Window and its two subtypes — class Frame and class Dialog — as well as the Panel class. In addition to the containers provided by the AWT, the Applet class is a container — it is a subtype of the Panel class and can therefore hold components. Brief descriptions of each container class provided by the AWT are provided below.

Window	A top-level display surface (a window). An instance of the Window class is not attached to nor embedded within another container. An instance of the Window class has no border and no title.
Frame	A top-level display surface (a window) with a border and title. An instance of the Frame class may have a menu bar. It is otherwise very much like an instance of the Window class.
Dialog	A top-level display surface (a window) with a border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.
Panel	A generic container for holding components. An instance of the Panel class provides a container to which to add components.

4.3.4 Button

Q15. Explain about buttons.

Ans :

Button is a control component that has a label and generates an event when pressed. When a button is pressed and released, AWT sends an instance of ActionEvent to the button, by calling processEvent on the button. The button's processEvent method receives all events for the button; it passes an action event along by calling its own processActionEvent method. The latter method passes the action event on to any action listeners that have registered an interest in action events generated by this button.

If an application wants to perform some action based on a button being pressed and released, it should implement ActionListener and register the new listener to receive events from this button, by calling the button's addActionListener method. The application can make use of the button's action command as a messaging protocol.

Class declaration

Following is the declaration for **java.awt.Button** class:

public class Button extends Component implements Accessible

Class constructors

S. N.	Constructor & Description
1	**Button():** Constructs a button with an empty string for its label.
2	**Button(String text):** Constructs a new button with specified label.

Class methods

S.No.	Method & Description
1	**void addActionListener(ActionListener l):** Adds the specified action listener to receive action events from this button.
2	**void addNotify():** Creates the peer of the button.
3	**AccessibleContext getAccessibleContext():** Gets the AccessibleContext associated with this Button.
4	**String getActionCommand():** Returns the command name of the action event fired by this button.
5	**ActionListener[] getActionListeners():** Returns an array of all the action listeners registered on this button.
6	**String getLabel():** Gets the label of this button.
7	**<T extends EventListener>T[] getListeners(Class<T> listenerType):** Returns an array of all the objects currently registered as FooListeners upon this Button.
8	**protected String paramString():** Returns a string representing the state of this Button.
9	**protected void processActionEvent(ActionEvent e):** Processes action events occurring on this button by dispatching them to any registered ActionListener objects.
10	**protected void processEvent(AWTEvent e):** Processes events on this button.
11	**void removeActionListener(ActionListener l):** Removes the specified action listener so that it no longer receives action events from this button.
12	**void setActionCommand(String command):** Sets the command name for the action event fired by this button.
13	**void setLabel(String label):** Sets the button's label to be the specified string.

Methods inherited

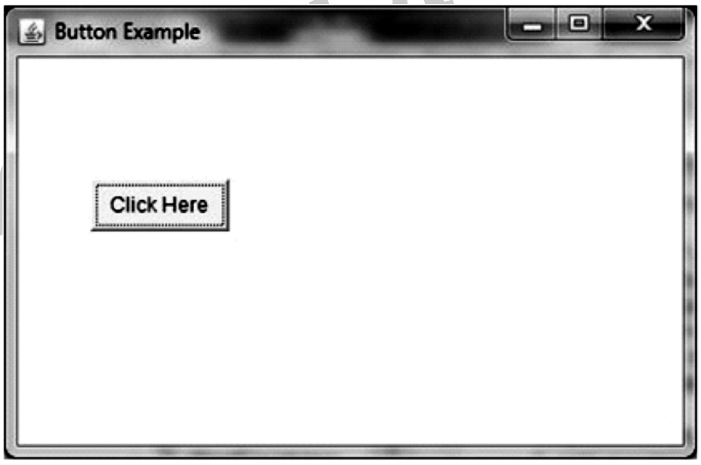
This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

Program

```
import java.awt.*;
public class ButtonExample
{
public static void main(String[] args)
{
    Frame f=new Frame("Button Example");
    Button b=new Button("Click Here");
    b.setBounds(50,100,80,30);
    f.add(b);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output



4.3.5 Label

Q16. Explain the API of Label Class?

Ans :

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

Class declaration

Following is the declaration for **java.awt.Label** class:

public class Label extends Component implements Accessible

Field

Following are the fields for **java.awt.Component** class:

- **static int CENTER** — Indicates that the label should be centered.
- **static int LEFT** — Indicates that the label should be left justified.
- **static int RIGHT** — Indicates that the label should be right justified.

Class constructors

S.No.	Constructor & Description
1	**Label():** Constructs an empty label.
2	**Label(String text):** Constructs a new label with the specified string of text, left justified.
3	**Label(String text, int alignment):** Constructs a new label that presents the specified string of text with the specified alignment.

Class methods

S.No.	Method & Description
1	**void addNotify():** Creates the peer for this label.
2	**AccessibleContext getAccessibleContext():** Gets the AccessibleContext associated with this Label.
3	**int getAlignment():** Gets the current alignment of this label.
4	**String getText():** Gets the text of this label.
5	**protected String paramString():** Returns a string representing the state of this Label.
6	**void setAlignment(int alignment):** Sets the alignment for this label to the specified alignment.
7	**void setText(String text):** Sets the text for this label to the specified text.

Methods inherited

This class inherits methods from the following classes:

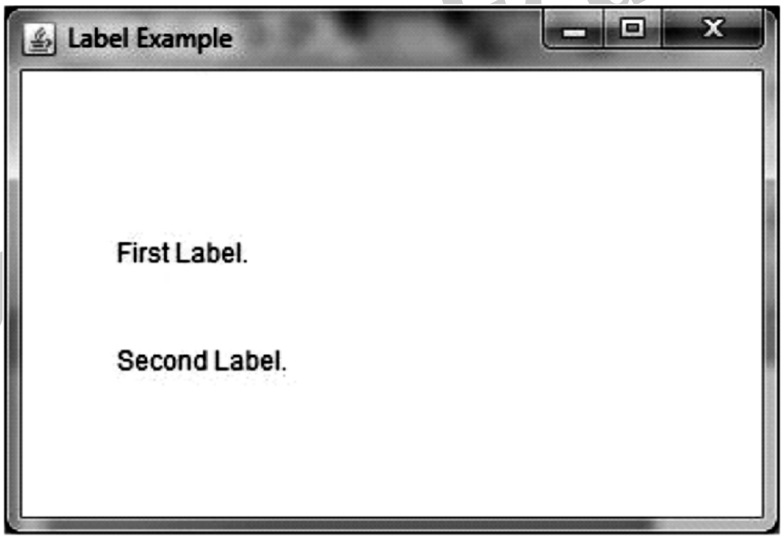
- java.awt.Component
- java.lang.Object

Program

```
import java.awt.*;
class LabelExample
{
```

```
public static void main(String args[])
{
    Frame f= new Frame("Label Example");
    Label l1,l2;
    l1=new Label("First Label.");
    l1.setBounds(50,100, 100,30);
    l2=new Label("Second Label.");
    l2.setBounds(50,150, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
}
```

Output:



First Label.

Second Label.

4.3.6 Checkbox / Radio Buttons

Q17. Explain in detail about Checkbox control with example?

Ans : **(Dec.-19, Dec.-18(MGU)**

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

Class declaration

Following is the declaration for java.awt.Checkbox class:

public class Checkboxe xtends Component implementsItem Selectable, Accessible

Class constructors

S.No.	Constructor & Description
1	**Checkbox():** Creates a check box with an empty string for its label.
2	**Checkbox(String label):** Creates a check box with the specified label.
3	**Checkbox(String label, boolean state):** Creates a check box with the specified label and sets the specified state.
4	**Checkbox(String label, boolean state, CheckboxGroup group)** Constructs a Checkbox with the specified label, set to the specified state, and in the specified check box group.
5	**Checkbox(String label, CheckboxGroup group, boolean state)** Creates a check box with the specified label, in the specified check box group, and set to the specified state.

Class methods

S.No.	Method & Description
1	**void addItemListener(ItemListener l)** Adds the specified item listener to receive item events from this check box.
2	**void addNotify()** Creates the peer of the Checkbox.
3	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this Checkbox.
4	**CheckboxGroup getCheckboxGroup()** Determines this check box's group.
5	**ItemListener[] getItemListeners()** Returns an array of all the item listeners registered on this checkbox.
6	**String getLabel()** Gets the label of this check box.
7	**<T extends EventListener>T[] getListeners(Class<T> listenerType)** Returns an array of all the objects currently registered as FooListeners upon this Checkbox.
8	**Object[] getSelectedObjects()** Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
9	**boolean getState()** Determines whether this check box is in the **on** or **off** state.
10	**protected String paramString()** Returns a string representing the state of this Checkbox.
11	**protected void processEvent(AWTEvent e)** Processes events on this check box.
12	**protected void processItemEvent(ItemEvent e)** Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
13	**void removeItemListener(ItemListener l)** Removes the specified item listener so that the item listener no longer receives item events from this check box.
14	**void setCheckboxGroup(CheckboxGroup g)** Sets this check box's group to the specified check box group.
15	**void setLabel(String label)** Sets this check box's label to be the string argument.
16	**void setState(boolean state)** Sets the state of this check box to the specified state.

Methods inherited

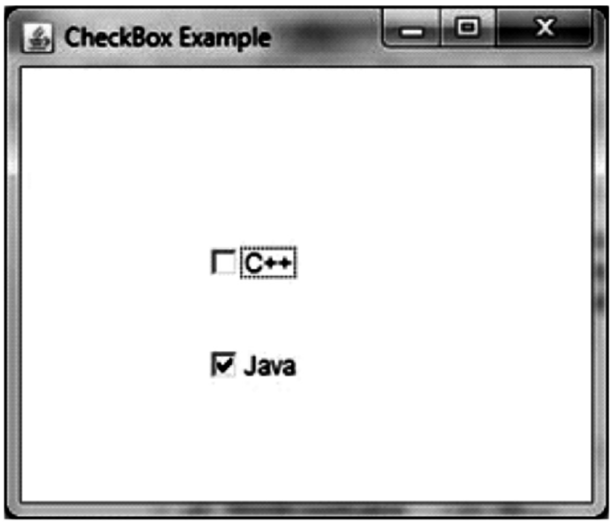
This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

Program

```
public class CheckboxExample
{
CheckboxExample()
{
    Frame f= new Frame("Checkbox Example");
    Checkbox checkbox1 = new Checkbox("C++");
    checkbox1.setBounds(100,100, 50,50);
    Checkbox checkbox2 = new Checkbox("Java", true);
    checkbox2.setBounds(100,150, 50,50);
    f.add(checkbox1);
    f.add(checkbox2);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String args[])
{
    new CheckboxExample();
}
}
```

Out put



Q18. Explain the elements and methods used to group the set of CheckBox?

Ans :

The CheckboxGroup class is used to group the set of checkbox.

Class declaration

Following is the declaration for **java.awt.CheckboxGroup** class:

public class Checkbox Groupex tends Objectimplements Serializable

Class constructors

S.No.	Constructor & Description
1	**CheckboxGroup() ():** Creates a new instance of CheckboxGroup.

Class methods

S.No.	Method & Description
1	**Checkbox getCurrent():** Deprecated. As of JDK version 1.1, replaced by getSelectedCheckbox().
2	**Checkbox getSelectedCheckbox():** Gets the current choice from this check box group.
3	**void setCurrent(Checkbox box):** Deprecated. As of JDK version 1.1, replaced by setSelectedCheckbox(Checkbox).
4	**void setSelectedCheckbox(Checkbox box):** Sets the currently selected check box in this group to be the specified check box.
5	**String toString():** Returns a string representation of this check box group, including the value of its current selection.

Methods inherited

This class inherits methods from the following classes:

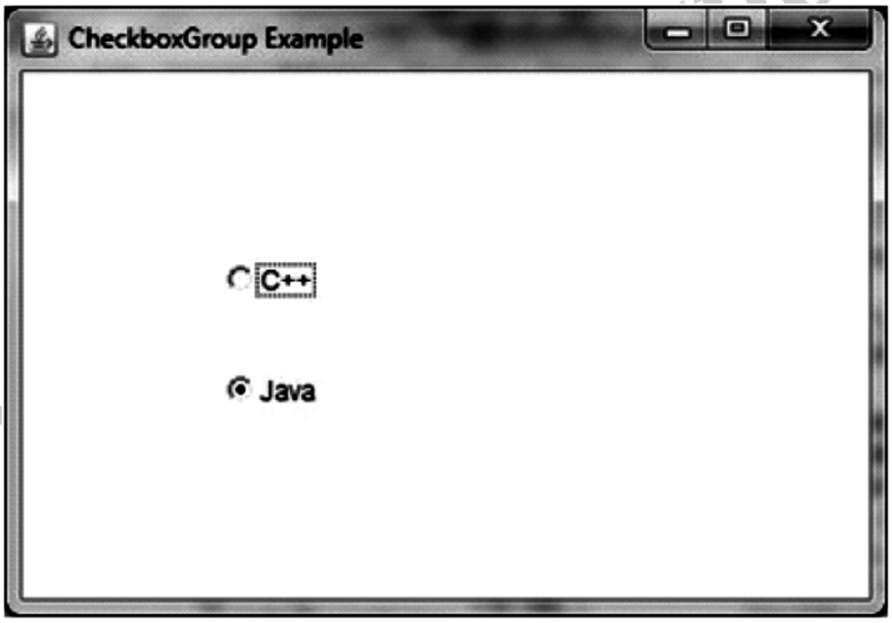
- java.lang.Object

Program

```
import java.awt.*;
public class CheckboxGroupExample
{
CheckboxGroupExample()
{
    Frame f= new Frame(“CheckboxGroup Example”);
    CheckboxGroup cbg = new CheckboxGroup();
```

```
        Checkbox checkBox1 = new Checkbox("C++", cbg, false);
        checkBox1.setBounds(100,100, 50,50);
        Checkbox checkBox2 = new Checkbox("Java", cbg, true);
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
    new CheckboxGroupExample();
    }
    }
```

Output:



4.3.7 Layouts

Q19. What is Layout Manager?

(OR)

What is the Purpose of Layouts?

Ans : **(Dec,-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 is done automatically by the Layout Manager.

Layout Manager

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

- It is very tedious to handle a large number of controls within the container.
- Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide us with various layout manager to position the controls. The properties like size,shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

Following are the interfaces defining functionalities of Layout Managers.

S.No.	Interface & Description
1	**Layout Manager -**The LayoutManager interface declares those methods which need to be implemented by the class whose object will act as a layout manager.
2	**LayoutManager2 -**The LayoutManager2 is the sub-interface of the LayoutManager.This interface is for those classes that know how to layout containers based on layout constraint object.

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

AWT Layout Manager Classes:

Following is the list of commonly used controls while designed GUI using AWT.

S.No.	LayoutManager & Description
1	**BorderLayout -** The borderlayout arranges the components to fit in the five regions: east, west, north, south and center.
2	**CardLayout -** The CardLayout object treats each component in the container as a card. Only one card is visible at a time.
3	**FlowLayout -** The FlowLayout is the default layout.It layouts the components in a directional flow.
4	**GridLayout -** The GridLayout manages the components in form of a rectangular grid.
5	**GridBagLayout -** This is the most flexible layout manager class.The object of GridBagLayout aligns the component vertically,horizontally or along their baseline without requiring the components of same size.

Q20. Explain types of layout managers with an example.

(OR)

Explain different layout managers in java with examples.

Ans : **(June-19(MGU), Dec.-18, Dec.-18(MGU)**

1. BorderLayout Manager

The class BorderLayout arranges the components to fit in the five regions: east, west, north, south and center. Each region is can contain only one component and each component in each region is identified by the corresponding constant NORTH, SOUTH, EAST, WEST, and CENTER.

Class declaration

Following is the declaration for **java.awt.BorderLayout** class:

public class BorderLayout extends Object implements LayoutManager2,Serializable

Field

Following are the fields for **java.awt.BorderLayout** class:

- **static String AFTER_LAST_LINE** — Synonym for PAGE_END.
- **static String AFTER_LINE_ENDS** — Synonym for LINE_END.
- **static String BEFORE_FIRST_LINE** — Synonym for PAGE_START.
- **static String BEFORE_LINE_BEGINS** — Synonym for LINE_START.
- **static String CENTER** — The center layout constraint (middle of container).
- **static String EAST** — The east layout constraint (right side of container).
- **static String LINE_END** — The component goes at the end of the line direction for the layout.
- **static String LINE_START** — The component goes at the beginning of the line direction for the layout.
- **static String NORTH** — The north layout constraint (top of container).
- **static String PAGE_END** — The component comes after the last line of the layout's content.
- **static String PAGE_START** — The component comes before the first line of the layout's content.
- **static String SOUTH** — The south layout constraint (bottom of container).
- **static String WEST** — The west layout constraint (left side of container).

Class constructors

S.N.	Constructor & Description
1	**BorderLayout() -** Constructs a new border layout with no gaps between components.
2	**BorderLayout(int hgap, int vgap) -** Constructs a border layout with the specified gaps between components.

Class methods

S.N.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints) -** Adds the specified component to the layout, using the specified constraint object.
2	**void addLayoutComponent(String name, Component comp) -** If the layout manager uses a per-component string, adds the component comp to the layout, associating it with the string specified by name.
3	**int getHgap() -** Returns the horizontal gap between components.
4	**float getLayoutAlignmentX(Container parent) -** Returns the alignment along the x axis.
5	**float getLayoutAlignmentY(Container parent) -** Returns the alignment along the y axis.
6	**int getVgap() -** Returns the vertical gap between components.
7	**void invalidateLayout(Container target) -** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
8	**void layoutContainer(Container target) -** Lays out the container argument using this border layout.
9	**Dimension maximumLayoutSize(Container target) -** Returns the maximum dimensions for this layout given the components in the specified target container.
10	**Dimension minimumLayoutSize(Container target) -** Determines the minimum size of the target container using this layout manager.
11	**Dimension preferredLayoutSize(Container target) -** Determines the preferred size of the target container using this layout manager, based on the components in the container.
12	**void removeLayoutComponent(Component comp) -** Removes the specified component from this border layout.
13	**void setHgap(int hgap) -** Sets the horizontal gap between components.
14	**void setVgap(int vgap) -** Sets the vertical gap between components.
15	**String toString() -** Returns a string representation of the state of this border layout.

Methods inherited

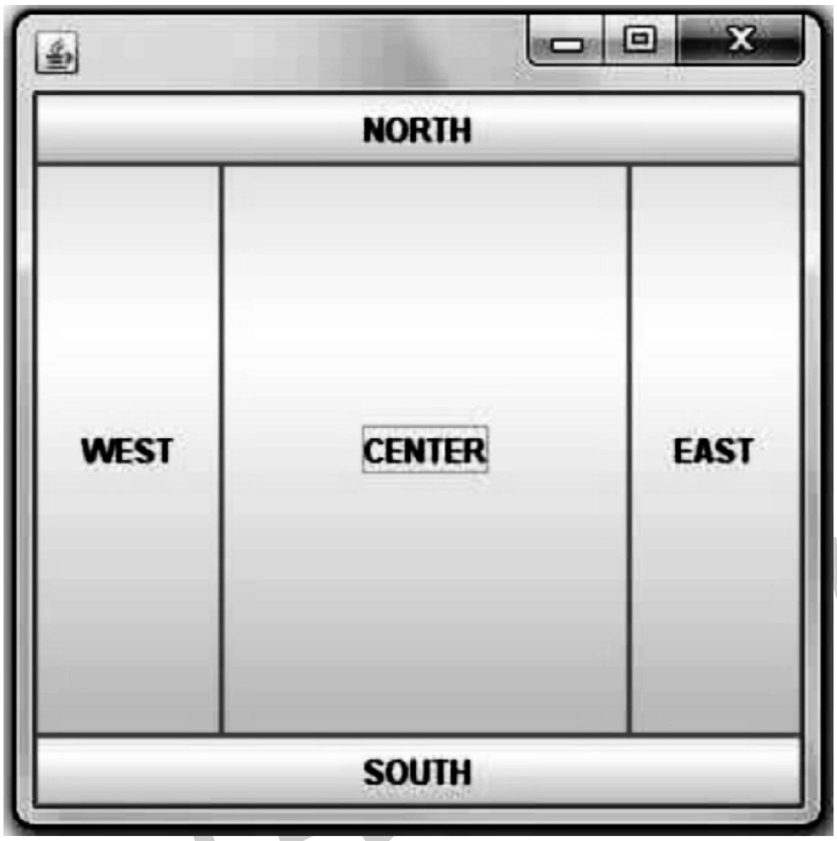
This class inherits methods from the following classes:

- java.lang.Object

Program

```
    import java.awt.*;
    import javax.swing.*;
    public class Border
{
    JFrame f;
    Border()
{
    f=new JFrame();
    JButton b1=new JButton("NORTH");;
    JButton b2=new JButton("SOUTH");;
    JButton b3=new JButton("EAST");;
    JButton b4=new JButton("WEST");;
    JButton b5=new JButton("CENTER");;
    f.add(b1,BorderLayout.NORTH);
    f.add(b2,BorderLayout.SOUTH);
    f.add(b3,BorderLayout.EAST);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);
    f.setSize(300,300);
    f.setVisible(true);
}
    public static void main(String[] args)
{
    new Border();
}
}
```

Output:



2. CardLayout Manager

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.

Class declaration

Following is the declaration for **java.awt.CardLayout** class:

publicclassCardLayout

extends Objectimplements Layout Manager 2,Serializable

Class constructors

S.N.	**Constructor & Description**
1	**CardLayout() -** Creates a new card layout with gaps of size zero.
2	**CardLayout(int hgap, int vgap) -** Creates a new card layout with the specified horizontal and vertical gaps.

Class methods

S.N.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints) -** Adds the specified component to this card layout's internal table of names.
2	**void addLayoutComponent(String name, Component comp) -** If the layout manager uses a per-component string, adds the component comp to the layout, associating it with the string specified by name.
3	**void first(Container parent) -** Flips to the first card of the container.
4	**int getHgap() -** Gets the horizontal gap between components.
5	**float getLayoutAlignmentX(Container parent) -** Returns the alignment along the x axis.
6	**float getLayoutAlignmentY(Container parent) -** Returns the alignment along the y axis.
7	**int getVgap() -** Gets the vertical gap between components.
8	**void invalidateLayout(Container target) -** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
9	**void last(Container parent) -** Flips to the last card of the container.
10	**void layoutContainer(Container parent) -** Lays out the specified container using this card layout.
11	**Dimension maximumLayoutSize(Container target) -** Returns the maximum dimensions for this layout given the components in the specified target container.
12	**Dimension minimumLayoutSize(Container parent) -** Calculates the minimum size for the specified panel.
13	**void next(Container parent) -** Flips to the next card of the specified container.
14	**Dimension preferredLayoutSize(Container parent) -** Determines the preferred size of the container argument using this card layout.
15	**void previous(Container parent) -** Flips to the previous card of the specified container.
16	**void removeLayoutComponent(Component comp) -** Removes the specified component from the layout.
17	**void setHgap(int hgap) -** Sets the horizontal gap between components.
18	**void setVgap(int vgap) -** Sets the vertical gap between components.
19	**void show(Container parent, String name) -** Flips to the component that was added to this layout with the specified name, using addLayoutComponent.
20	**String toString()-** Returns a string representation of the state of this card layout.

Methods inherited

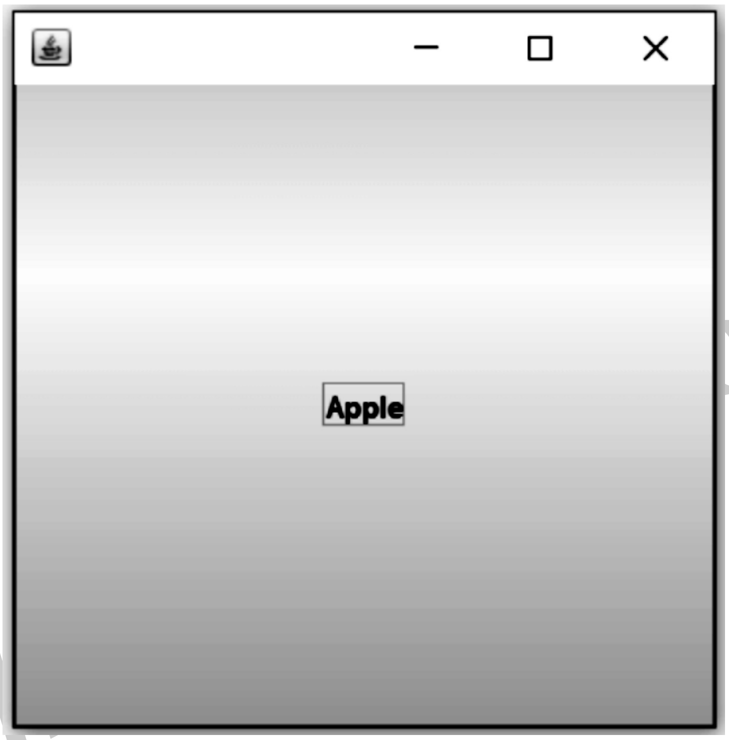
This class inherits methods from the following classes:

- java.lang.Object

Program

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener
{
CardLayout card;
JButton b1,b2,b3;
Container c;
CardLayoutExample()
{
c=getContentPane();
card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
c.setLayout(card);
b1=new JButton("Apple");
b2=new JButton("Boy");
b3=new JButton("Cat");
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
c.add("a",b1);c.add("b",b2);c.add("c",b3);
}
public void actionPerformed(ActionEvent e)
{
card.next(c);
}
public static void main(String[] args)
{
CardLayoutExample cl=new CardLayoutExample();
cl.setSize(400,400);
cl.setVisible(true);
cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

Output:



3. FlowLayout Manager

The class **FlowLayout** components in a left-to-right flow.

Class declaration

Following is the declaration for **java.awt.FlowLayout** class:

public class FlowLayout extends Object implements LayoutManager,Serializable

Field

Following are the fields for **java.awt.BorderLayout** class:

- **static int CENTER** — This value indicates that each row of components should be centered.
- **static int LEADING** — This value indicates that each row of components should be justified to the leading edge of the container's orientation, for example, to the left in left-to-right orientations.
- **static int LEFT** — This value indicates that each row of components should be left-justified.
- **static int RIGHT** — This value indicates that each row of components should be right-justified.
- **static int TRAILING** — This value indicates that each row of components should be justified to the trailing edge of the container's orientation, for example, to the right in left-to-right orientations.

Class constructors

S.N.	Constructor & Description
1	**FlowLayout() -** Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.
2	**FlowLayout(int align) -** Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.
3	**FlowLayout(int align, int hgap, int vgap) -** Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

Class methods

S.N.	Method & Description
1	**void addLayoutComponent(String name, Component comp) -** Adds the specified component to the layout.
2	**int getAlignment() -** Gets the alignment for this layout.
3	**int getHgap() -** Gets the horizontal gap between components.
4	**int getVgap() -** Gets the vertical gap between components.
5	**void layoutContainer(Container target) -** Lays out the container.
6	**Dimension minimumLayoutSize(Container target) -** Returns the minimum dimensions needed to layout the visible components contained in the specified target container.
7	**Dimension preferredLayoutSize(Container target) -** Returns the preferred dimensions for this layout given the visible components in the specified target container.
8	**void removeLayoutComponent(Component comp) -** Removes the specified component from the layout.
9	**void setAlignment(int align) -** Sets the alignment for this layout.
10	**void setHgap(int hgap) -** Sets the horizontal gap between components.
11	**void setVgap(int vgap) -** Sets the vertical gap between components.
12	**String toString() -** Returns a string representation of this FlowLayout object and its values.

Methods inherited

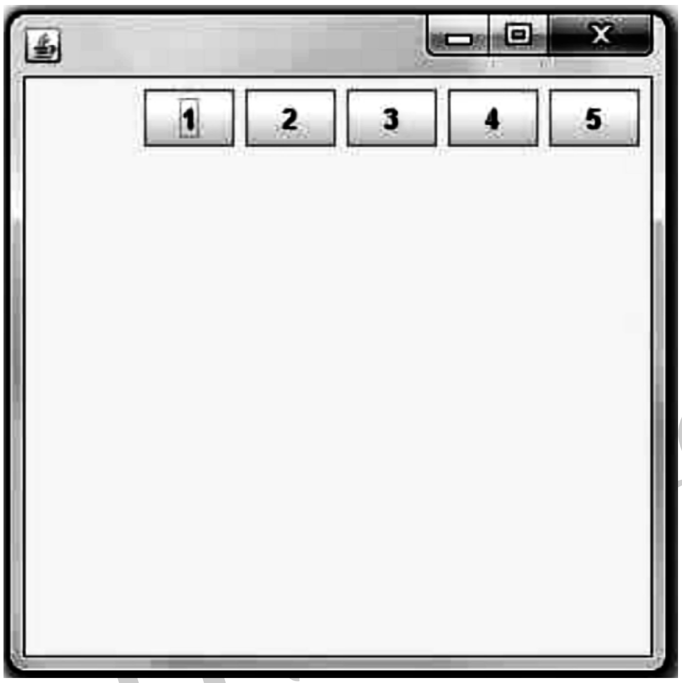
This class inherits methods from the following classes:

java.lang.Object

Program

```
import java.awt.*;
import javax.swing.*;
public class MyFlowLayout
{
    JFrame f;
    MyFlowLayout()
    {
    f=new JFrame();
    JButton b1=new JButton("1");
    JButton b2=new JButton("2");
    JButton b3=new JButton("3");
    JButton b4=new JButton("4");
    JButton b5=new JButton("5");
    f.add(b1);
    f.add(b2);
    f.add(b3);
    f.add(b4);
    f.add(b5);
    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment
    f.setSize(300,300);
    f.setVisible(true);
    }
    public static void main(String[] args)
    {
    new  MyFlowLayout();
    }
}
```

Output:



4. GridLayout Manager

The class **GridLayout** arranges components in a rectangular grid.

Class declaration

Following is the declaration for **java.awt.GridLayout** class:

public class GridLayout extends Object implements LayoutManager,Serializable

Class constructors

S.N.	Constructor & Description
1	**GridLayout() -** Creates a grid layout with a default of one column per component, in a single row.
2	**GridLayout(int rows, int cols) -** Creates a grid layout with the specified number of rows and columns.
3	**GridLayout(int rows, int cols, int hgap, int vgap) -** Creates a grid layout with the specified number of rows and columns.

Class methods

S.N.	Method & Description
1	**void addLayoutComponent(String name, Component comp) -** Adds the specified component with the specified name to the layout.
2	**int getColumns() -** Gets the number of columns in this layout.
3	**int getHgap() -** Gets the horizontal gap between components.
4	**int getRows() -** Gets the number of rows in this layout.
5	**int getVgap() -** Gets the vertical gap between components.
6	**void layoutContainer(Container parent) -** Lays out the specified container using this layout.
7	**Dimension minimumLayoutSize(Container parent) -** Determines the minimum size of the container argument using this grid layout.
8	**Dimension preferredLayoutSize(Container parent) -** Determines the preferred size of the container argument using this grid layout.
9	**void removeLayoutComponent(Component comp) -** Removes the specified component from the layout.
10	**void setColumns(int cols) -** Sets the number of columns in this layout to the specified value.
11	**void setHgap(int hgap) -** Sets the horizontal gap between components to the specified value.
12	**void setRows(int rows) -** Sets the number of rows in this layout to the specified value.
13	**void setVgap(int vgap) -** Sets the vertical gap between components to the specified value.
14	**String toString() -** Returns the string representation of this grid layout's values.

Methods inherited

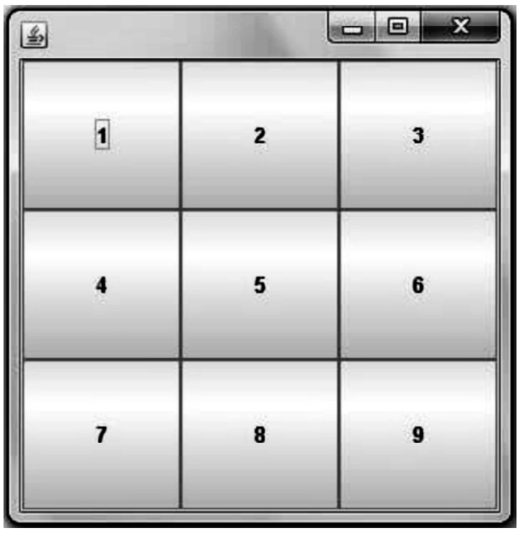
This class inherits methods from the following classes:

java.lang.Object

Program

```
import  java.awt.*;
import javax.swing.*;
public class MyGridLayout
{
      JFrame f;
      MyGridLayout()
      {
      f=new JFrame();
      JButton b1=new JButton("1");
      JButton b2=new JButton("2");
      JButton b3=new JButton("3");
      JButton b4=new JButton("4");
      JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
      JButton b8=new JButton("8");
      JButton b9=new JButton("9");
      f.add(b1);
      f.add(b2);
      f.add(b3);
      f.add(b4);
      f.add(b5);
      f.add(b6);
      f.add(b7);
      f.add(b8);
      f.add(b9);
      f.setLayout(new GridLayout(3,3));
      f.setSize(300,300);//setting grid layout of 3 rows and 3 columns
      f.setVisible(true);
      }
      public static void main(String[] args)
      {
      new MyGridLayout();
      }
}
```

Output:



5. GridBagLayout Manager

The class GridBagLayout arranges components in a horizontal and vertical manner.

Class declaration

Following is the declaration for **java.awt.GridBagLayout** class:

public class GridBagLayout extends Objec timplements LayoutManager2, Serializable

Field

Following are the fields for **java.awt.BorderLayout** class:

- **double[] columnWeights** — This field holds the overrides to the column weights.
- **int[] columnWidths** — This field holds the overrides to the column minimum width.
- **protected Hashtable comptable** — This hashtable maintains the association between a component and its gridbag constraints.
- **protected GridBagConstraints defaultConstraints** — This field holds a gridbag constraints instance containing the default values, so if a component does not have gridbag constraints associated with it, then the component will be assigned a copy of the defaultConstraints.
- **protected java.awt.GridBagLayoutInfo** — This field holds the layout information for the gridbag.
- **protected static int MAXGRIDSIZE** — The maximum number of grid positions (both horizontally and vertically) that can be laid out by the grid bag layout.
- **protected static int MINSIZE** — The smallest grid that can be laid out by the grid bag layout.
- **protected static int PREFERREDSIZE** — The preferred grid size that can be laid out by the grid bag layout.
- **int[] rowHeights** — This field holds the overrides to the row minimum heights.
- **double[] rowWeights** — This field holds the overrides to the row weights.

Class constructors

S.No.	Constructor & Description
1	**GridBagLayout()** - Creates a grid bag layout manager.

Class methods

S.No.	Method & Description
1	**void addLayoutComponent(Component comp, Object constraints) -** Adds the specified component to the layout, using the specified constraints object.
2	**void addLayoutComponent(String name, Component comp) -** Adds the specified component with the specified name to the layout.
3	**protected void adjustForGravity(GridBagConstraints constraints, Rectangle r) -** Adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
4	**protected void AdjustForGravity(GridBagConstraints constraints, Rectangle r) -** This method is obsolete and supplied for backwards compatability only; new code should call adjustForGravity instead.
5	**protected void arrangeGrid(Container parent) -** Lays out the grid.
6	**protected void ArrangeGrid(Container parent) -** This method is obsolete and supplied for backwards compatability only; new code should call arrangeGrid instead.
7	**GridBagConstraints getConstraints(Component comp) -** Gets the constraints for the specified component.
8	**float getLayoutAlignmentX(Container parent) -** Returns the alignment along the x axis.
9	**float getLayoutAlignmentY(Container parent) -** Returns the alignment along the y axis.
10	**int[][] getLayoutDimensions() -** Determines column widths and row heights for the layout grid.
11	**protected java.awt.GridBagLayoutInfo getLayoutInfo(Container parent, int sizeflag) -** Fills in an instance of GridBagLayoutInfo for the current set of managed children.
12	**protected java.awt.GridBagLayoutInfo GetLayoutInfo(Container parent, int sizeflag) -** This method is obsolete and supplied for backwards compatability only; new code should call getLayoutInfo instead.
13	**Point getLayoutOrigin() -** Determines the origin of the layout area, in the graphics coordinate space of the target container.
14	**double[][] getLayoutWeights() -** Determines the weights of the layout grid's columns and rows.
15	**protected Dimension getMinSize(Container parent, java.awt.GridBagLayoutInfo info) -** Figures out the minimum size of the master based on the information from getLayoutInfo().
16	**protected Dimension GetMinSize(Container parent, java.awt.GridBagLayoutInfo info) -** This method is obsolete and supplied for backwards compatability only; new code should call getMinSize instead.
17	**void invalidateLayout(Container target) -** Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
18	**void layoutContainer(Container parent) -** Lays out the specified container using this grid bag layout.
19	**Point location(int x, int y) -** Determines which cell in the layout grid contains the point specified by (x, y).
20	**protected GridBagConstraints lookupConstraints(Component comp) -** Retrieves the constraints for the specified component.
21	**Dimension maximumLayoutSize(Container target) -**Returns the maximum dimensions for this layout given the components in the specified target container.
22	**Dimension minimumLayoutSize(Container parent) -** Determines the minimum size of the parent container using this grid bag layout.
23	**Dimension preferredLayoutSize(Container parent) -** Determines the preferred size of the parent container using this grid bag layout.
24	**void removeLayoutComponent(Component comp) -** Removes the specified component from this layout.
25	**void setConstraints(Component comp, GridBagConstraints constraints) -** Sets the constraints for the specified component in this layout.
26	**String toString() -** Returns a string representation of this grid bag layout's values.

Methods inherited

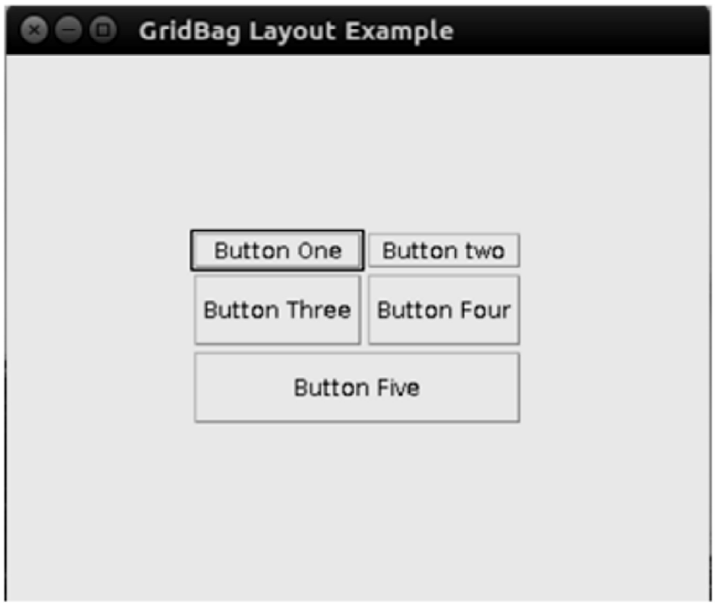
This class inherits methods from the following classes:

Program

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame
{
    public static void main(String[] args)
    {
    GridBagLayoutExample a = new GridBagLayoutExample();
    }
    public GridBagLayoutExample()
    {
    GridBagLayoutgrid = new GridBagLayout();
    GridBagConstraints gbc = new GridBagConstraints();
    setLayout(grid);
    setTitle("GridBag Layout Example");
    GridBagLayout layout = new GridBagLayout();
    this.setLayout(layout);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;
    this.add(new Button("Button One"), gbc);
    gbc.gridx = 1;
    gbc.gridy = 0;
    this.add(new Button("Button two"), gbc);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.ipady = 20;
```

```
        gbc.gridx = 0;
        gbc.gridy = 1;
        this.add(new Button("Button Three"), gbc);
        gbc.gridx = 1;
        gbc.gridy = 1;
        this.add(new Button("Button Four"), gbc);
        gbc.gridx = 0;
        gbc.gridy = 2;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.gridwidth = 2;
        this.add(new Button("Button Five"), gbc);
        setSize(300, 300);
        setPreferredSize(getSize());
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        }
    }
```

Output:



4.4 Swings

4.4.1 Introduction

Q21. What are swings? State the features of Swings?

Ans :

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Features

(i) Light Weight

Swing components 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.

(ii) Rich Controls

Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.

(iii) Highly Customizable

Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.

(iv) Pluggable look-and-feel

SWING based GUI Application look and feel can be changed at run-time, based on available values.

4.4.2 Differences between Swing and AWT

Q22. Compare and contrast AWT and Swing.

(OR)

What are the differences between AWT and Swing?

Ans : **(Dec.-19)**

S.No	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.

4.4.3 JFrame

Q23. Explain about JFrame by giving an example.

Ans : **(July-19)**

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of set Default Close Operation(int) method.

Nested Class

Modifier and Type	Class	Description
protected class	JFrame.AccessibleJFrame	This class implements accessibility support for the JFrame class.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The accessible context property.
static int	EXIT_ON_CLOSE	The exit application default window close operation.
protected JRootPane	rootPane	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled	If true then calls to add and setLayout will be forwarded to the contentPane.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified Graphics Configuration of a screen device and a blank title.
JFrame (String title)	It creates a new, initially invisible Frame with the specified title.
JFrame (String title, Graphics Configuration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Useful Methods

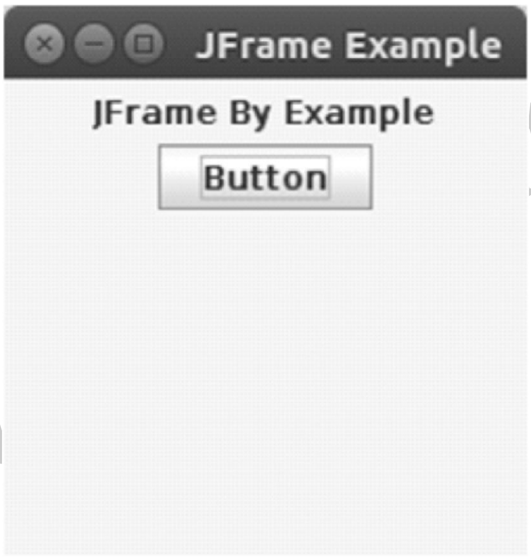
Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.
Void	setContentPane(Containe contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
Void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
Void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
Void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

Program

```
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 frame = new JFrame("JFrame Example");
     JPanel panel = new JPanel();
     panel.setLayout(new FlowLayout());
     JLabel label = new JLabel("JFrame By Example");
     JButton button = new JButton();
```

```
            button.setText("Button");
            panel.add(label);
            panel.add(button);
            frame.add(panel);
            frame.setSize(200, 300);
            frame.setLocationRelativeTo(null);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
            }
      }
```

Output:



4.4.4 JApplet

Q24. Explain about JApplet by giving an example.

Ans : **(July-19)**

JApplet is a class that represents the Swing applet. It is a subclass of Applet class and must be extended by all the applets that use Swing. It provides all the functionalities of the AWT applet as well as support for menubars and layering of components. Whenever we require to add a component to it, the component is added to the content pane. The JApplet class extends the Applet class.

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends
JApplet implements ActionListener
```

```
{
       JButton b;
       JTextField tf;
       public void init()
       {
       tf=new JTextField();
       tf.setBounds(30,40,150,20);
       b=new JButton(“Click”);
       b.setBounds(80,150,70,40);
       add(b);add(tf);
       b.addActionListener(this);
       setLayout(null);
       }
       public void action Performed (ActionEvent e)
       {
       tf.setText(“Welcome”);
       }
}
```

HTML CODE

```
<html>
<body>
<applet code=”EventJApplet.class” width=”300" height=”300">
</applet>
</body>
</html>
```

4.4.5 JPanel

Q25. Explain about JPanel by giving an example.

Ans : **(July-19)**

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn’t have title bar.

Class Declaration

Following is the declaration

for **javax.swing.JPanel** class “

public class JPanel extends JComponent implements Accessible

Class Constructors

S.No.	Constructor & Description
1	**JPanel()** Creates a new JPanel with a double buffer and a flow layout.
2	**JPanel(boolean isDoubleBuffered)** Creates a new JPanel with FlowLayout and the specified buffering strategy.
3	**JPanel(LayoutManager layout)** Creates a new buffered JPanel with the specified layout manager.
4	**JPanel(LayoutManager layout, boolean isDoubleBuffered)** Creates a new JPanel with the specified layout manager and buffering strategy.

Class Methods

Sr.No.	Method & Description
1	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JPanel.
2	**PanelUI getUI()** Returns the look and feel (L&F) object that renders this component.
3	**String getUIClassID()** Returns a string that specifies the name of the L&F class which renders this component.
4	**protected String paramString()** Returns a string representation of this JPanel.
5	**void setUI(PanelUI ui)** Sets the look and feel (L&F) object that renders this component.
6	**void updateUI()** Resets the UI property with a value from the current look and feel.

Methods Inherited

This class inherits methods from the following classes “

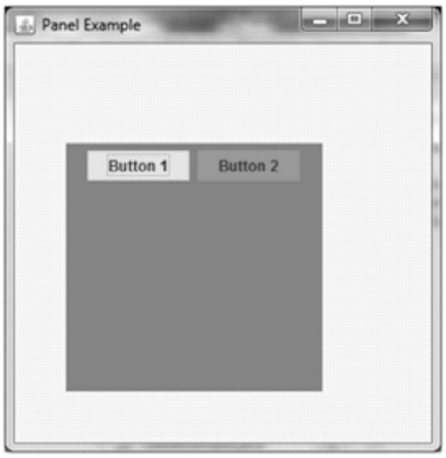
- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Program

```
import java.awt.*;
import javax.swing.*;
public class PanelExample
```

```
{
        PanelExample()
        {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
        public static void main(String args[])
{
        new PanelExample();
}
}
```

Output:



4.4.6 Components in Swings

Q26. Describe about various components in swings.

Ans :

Swing components are the basic building blocks of an application. We know that Swing is a GUI widget toolkit for Java. Every application has some basic interactive interface for the user. For example, a button, check-box, radio-button, text-field, etc. These together form the components in Swing. So, to summarise, Swing components are the interactive elements in a Java application.

Components

1. JButton

JButton class is used to create a push-button on the UI. The button can contain some display text or image. It generates an event when clicked and double-clicked. A JButton can be implemented in the application by calling one of its constructors.

Example:

```
JButton okBtn = new JButton("Ok");
```

This constructor returns a button with text Ok on it.

```
JButton homeBtn = new JButton(homeIcon);
```

It returns a button with a homeIcon on it.

```
JButton btn2 = new JButton(homeIcon, "Home");
```

It returns a button with the home icon and text Home.

2. JLabel

JLabel class is used to render a read-only text label or images on the UI. It does not generate any event.

Example:

```
JLabel textLbl = new JLabel("This is a text label.");
```

This constructor returns a label with text.

```
JLabel imgLabel = new JLabel(homeIcon);
```

It returns a label with a home icon.

3. JTextField

JTextField renders an editable single-line text box. A user can input non-formatted text in the box. To initialize the text field, call its constructor and pass an optional integer parameter to it. This parameter sets the width of the box measured by the number of columns. It does not limit the number of characters that can be input in the box.

Example:

```
JTextField txtBox = new JTextField(20);
```

It renders a text box of 20 column width.

4. JTextArea

JTextArea class renders a multi-line text box. Similar to the JTextField, a user can input non-formatted text in the field. The constructor for JTextArea also expects two integer parameters which define the height and width of the text-area in columns. It does not restrict the number of characters that the user can input in the text-area.

Example:

```
JTextArea txtArea = new JTextArea("This text is default text for text area.", 5, 20);
```

The above code renders a multi-line text-area of height 5 rows and width 20 columns, with default text initialized in the text-area.

5. JPasswordField

JPasswordField is a subclass of JTextField class. It renders a text-box that masks the user input text with bullet points. This is used for inserting passwords into the application.

Example:

```
JPasswordField pwdField = new JPasswordField(15);
var pwdValue = pwdField.getPassword();
```

It returns a password field of 15 column width. The getPassword method gets the value entered by the user.

6. JCheckBox

JCheckBox renders a check-box with a label. The check-box has two states – on/off. When selected, the state is on and a small tick is displayed in the box.

Example:

```
CheckBox chkBox = new JCheckBox("Show Help", true);
```

It returns a checkbox with the label Show Help. Notice the second parameter in the constructor. It is a boolean value that indicates the default state of the check-box. True means the check-box is defaulted to on state.

7. JRadioButton

JRadioButton is used to render a group of radio buttons in the UI. A user can select one choice from the group.

Example:

```
ButtonGroup radioGroup = new ButtonGroup();
JRadioButton rb1 = new JRadioButton("Easy", true);
JRadioButton rb2 = new JRadioButton("Medium");
JRadioButton rb3 = new JRadioButton("Hard");
radioGroup.add(rb1);
radioGroup.add(rb2);
radioGroup.add(rb3);
```

The above code creates a button group and three radio button elements. All three elements are then added to the group. This ensures that only one option out of the available options in the group can be selected at a time. The default selected option is set to Easy.

8. JList

JList component renders a scrollable list of elements. A user can select a value or multiple values from the list. This select behavior is defined in the code by the developer.

Example:

```
DefaultListItem cityList = new DefaultListItem();
cityList.addElement("Mumbai"):
cityList.addElement("London"):
cityList.addElement("New York"):
cityList.addElement("Sydney"):
cityList.addElement("Tokyo"):
JList cities = new JList(cityList);
cities.setSelectionModel(ListSelectionModel.SINGLE_SELECTION);
```

The above code renders a list of cities with 5 items in the list. The selection restriction is set to SINGLE_SELECTION. If multiple selections is to be allowed, set the behavior to MULTIPLE_INTERVAL_SELECTION.

9. JComboBox

JComboBox class is used to render a dropdown of the list of options.

Example:

```
String[] cityStrings = { "Mumbai", "London", "New York", "Sydney", "Tokyo" };
JComboBox cities = new JComboBox(cityList);
cities.setSelectedIndex(3);
```

The default selected option can be specified through the setSelectedIndex method. The above code sets Sydney as the default selected option.

10. JFileChooser

JFileChooser class renders a file selection utility. This component lets a user select a file from the local system.

Example:

```
JFileChooser fileChooser = new JFileChooser();
JButton fileDialogBtn = new JButton("Select File");
fileDialogBtn.AddEventListner(new ActionListner(){
fileChooser.showOpenDialog();
})
var selectedFile = fileChooser.getSelectedFile();
```

The above code creates a file chooser dialog and attaches it to the button. The button click would open the file chooser dialog. The selected file is returned through the getSelectedFile method.

11. JTabbedPane

JTabbedPane is another very useful component that lets the user switch between tabs in an application. This is a highly useful utility as it lets the user browse more content without navigating to different pages.

Example:

```
JTabbedPane tabbedPane = new JTabbedPane();
tabbedPane.addTab("Tab 1", new JPanel());
tabbedPane.addTab("Tab 2", new JPanel());
```

The above code creates a two tabbed panel with headings Tab 1 and Tab 2.

12. JSlider

JSlider component displays a slider which the user can drag to change its value. The constructor takes three arguments – minimum value, maximum value, and initial value.

Example:

```
JSlider volumeSlider = new JSlider(0, 100, 50);
var volumeLevel = volumeSlider.getValue();
```

The above code creates a slider from 0 to 100 with an initial value set to 50. The value selected by the user is returned by the getValue method.

4.4.7 Layout Managers

Q27. Discuss about Layout Managers in swing.

Ans : **(Imp.)**

A swing contents two different layout managers. They are :

1. Box Layout
2. Spring Layout

1. BoxLayout

The Java BoxLayout class is used to arrange the components either vertically or horizontally. For this purpose, the BoxLayout class provides four constants. They are as follows:

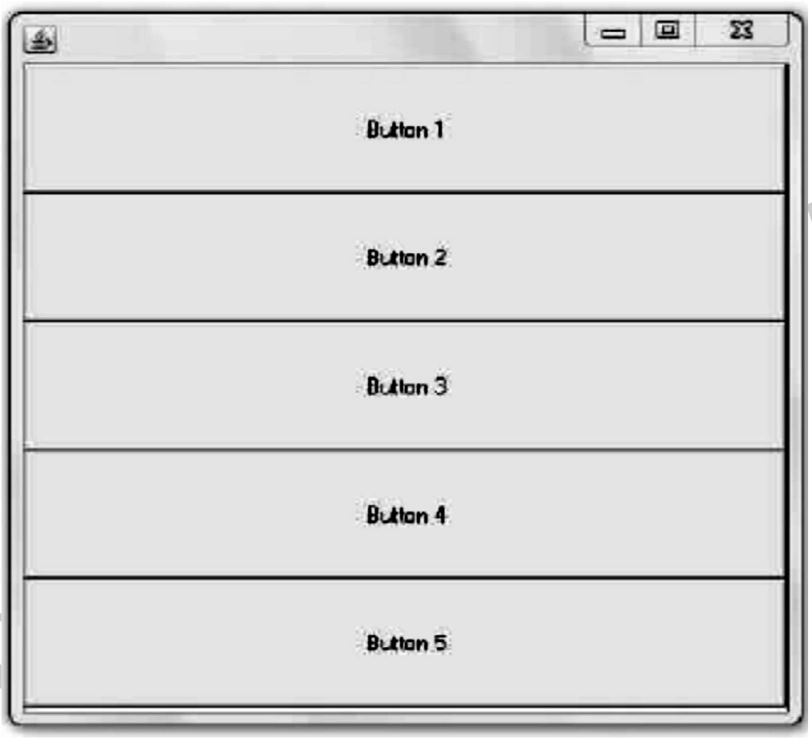
Fields of BoxLayout Class

(i) public static final int X_AXIS: Alignment of the components are horizontal from left to right.

Example of BoxLayout class with Y-AXIS:

```
import java.awt.*;
import javax.swing.*;
public class BoxLayoutExample1 extends Frame {
Button buttons[];
public BoxLayoutExample1 () {
buttons = new Button [5];
for (int i = 0;i<5;i++) {
    buttons[i] = new Button ("Button " + (i + 1));
    // adding the buttons so that it can be displayed
    add (buttons[i]);
}
// the buttons will be placed horizontally
setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
setSize(400,400);
setVisible(true);
}
    // main method
    public static void main(String args[]){
    BoxLayoutExample1 b=new BoxLayoutExample1();
}
}
download this example
```

Output:

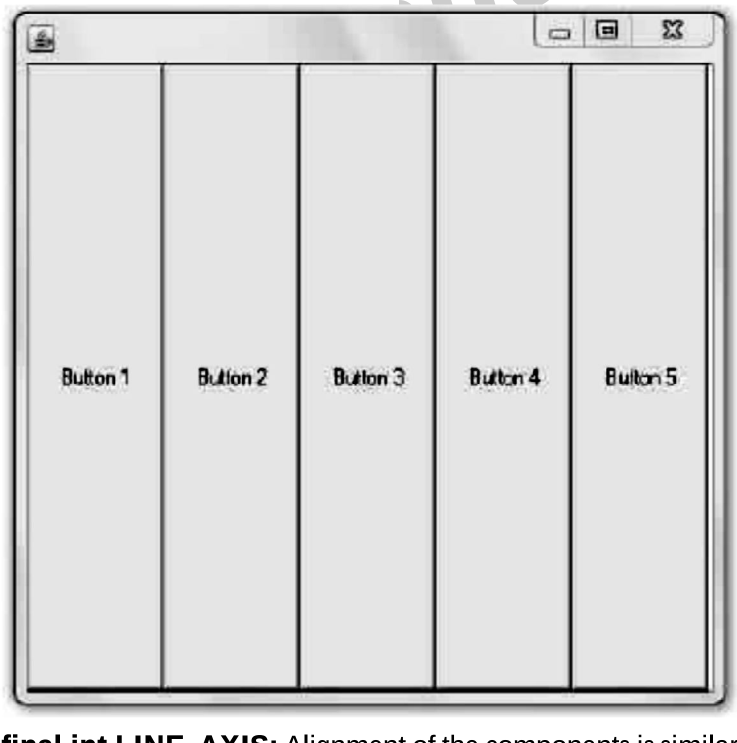


(ii) public static final int Y_AXIS: Alignment of the components are vertical from top to bottom.

Example of BoxLayout class with X-AXIS

```
import java.awt.*;
import javax.swing.*;
public class BoxLayoutExample2 extends Frame {
 Button buttons[];
public BoxLayoutExample2() {
buttons = new Button [5];
for (int i = 0;i<5;i++) {
buttons[i] = new Button ("Button " + (i + 1));
// adding the buttons so that it can be displayed
add (buttons[i]);
}
// the buttons in the output will be aligned vertically
setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
setSize(400,400);
setVisible(true);
}
 // main method
public static void main(String args[]){
BoxLayoutExample2 b=new BoxLayoutExample2();
}
}
download this example
```

Output:

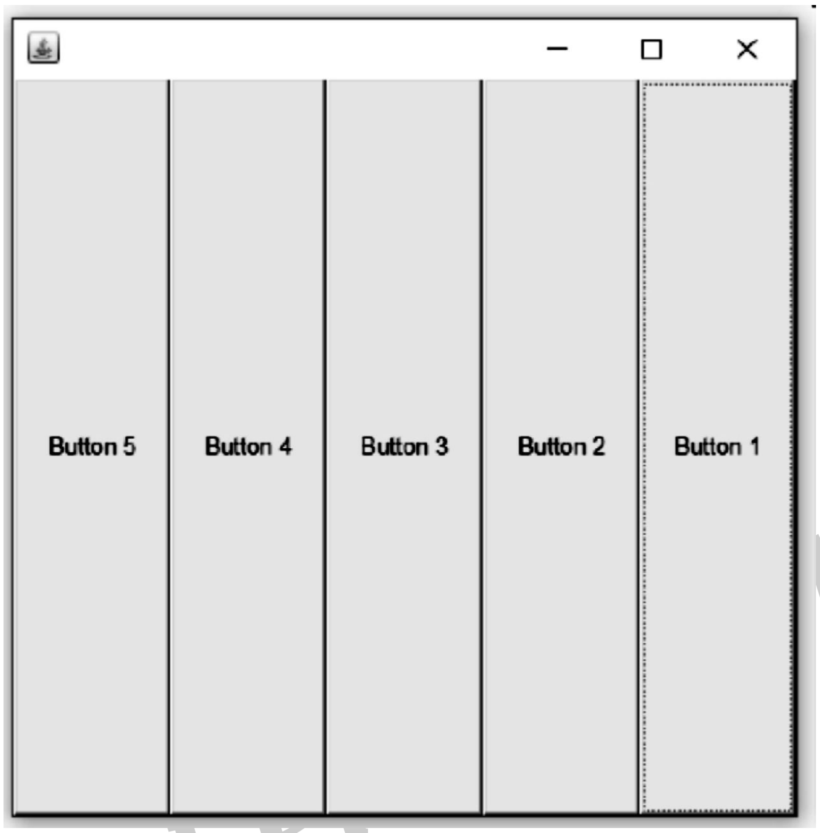


(iii) public static final int LINE_AXIS: Alignment of the components is similar to the way words are aligned in a line, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then the components are aligned horizontally; otherwise, the components are aligned vertically. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is also from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

Example

```
import java.awt.*;
import javax.swing.*;
public class BoxLayoutExample3 extends Frame
{
Button buttons[];
// constructor of the class
public BoxLayoutExample3()
{
buttons = new Button[5];
setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);  // line 11
for (int i = 0; i < 5; i++)
{
buttons[i] = new Button ("Button " + (i + 1));
// adding the buttons so that it can be displayed
add (buttons[i]);
}
// the ComponentOrientation is set to RIGHT_TO_LEFT. Therefore,
// the added buttons will be rendered from right to left
setLayout (new BoxLayout(this, BoxLayout.LINE_AXIS));
setSize(400, 400);
setVisible(true);
}
// main method
public static void main(String argvs[])
{
// creating an object of the class BoxLayoutExample3
BoxLayoutExample3 obj = new BoxLayoutExample3();
}
}
```

Output:

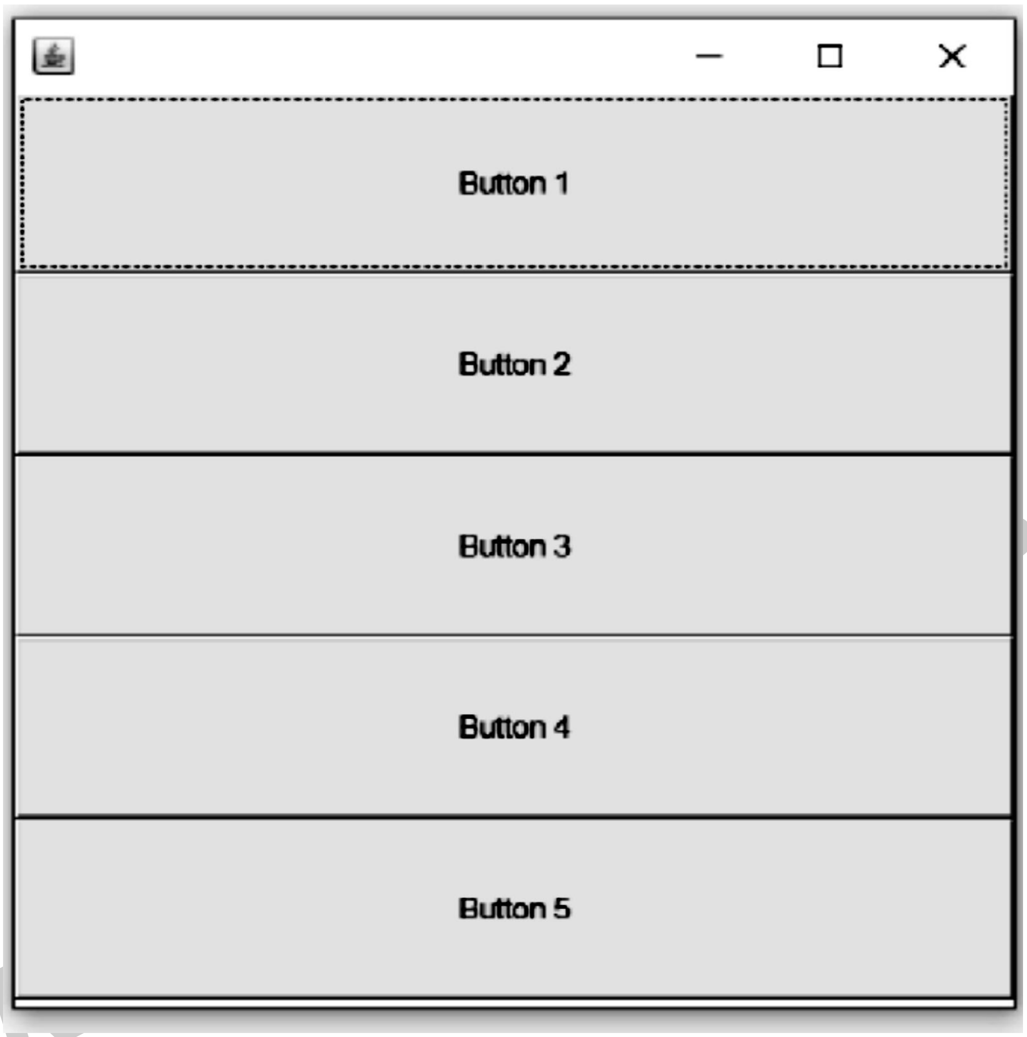


(iv) public static final int PAGE_AXIS: Alignment of the components is similar to the way text lines are put on a page, which is based on the ComponentOrientation property of the container. If the ComponentOrientation property of the container is horizontal, then components are aligned vertically; otherwise, the components are aligned horizontally. For horizontal orientations, we have two cases: left to right, and right to left. If the value ComponentOrientation property of the container is also from left to right, then the components are rendered from left to right, and for right to left, the rendering of components is from right to left. In the case of vertical orientations, the components are always rendered from top to bottom.

Example

```
import java.awt.*;

import javax.swing.*;

public class BoxLayoutExample4 extends Frame

{

Button buttons[];

// constructor of the class
public BoxLayoutExample4()
{
JFrame f = new JFrame();
JPanel pnl = new JPanel();
buttons = new Button[5];
GridBagConstraints constrntObj = new GridBagConstraints();
constrntObj.fill = GridBagConstraints.VERTICAL;
for (int i = 0; i < 5; i++)
{
buttons[i] = new Button ("Button " + (i + 1));
// adding the buttons so that it can be displayed
add(buttons[i]);
}
// the components will be displayed just like the line is present on a page
setLayout (new BoxLayout(this, BoxLayout.PAGE_AXIS));
setSize(400, 400);
setVisible(true);
}
// main method
public static void main(String argvs[])
{
// creating an object of the class BoxLayoutExample4
BoxLayoutExample4 obj = new BoxLayoutExample4();
}
}
```

Output:



2. Spring Layout

A SpringLayout arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout.Constraint object.

Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.

Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

Constructor

SpringLayout(): The default constructor of the class is used to instantiate the SpringLayout class.

Nested Classes

Modifier and Type	Class	Description
static class	SpringLayout.Constraints	It is a Constraints object helps to govern component's size and position change in a container that is controlled by Spring Layout

SpringLayout Fields

Modifier and Type	Field	Description
static String	BASELINE	It specifies the baseline of a component.
static String	EAST	It specifies the right edge of a component's bounding rectangle.
static String	HEIGHT	It specifies the height of a component's bounding rectangle.
static String	HORIZONTAL_CENTER	It specifies the horizontal center of a component's bounding rectangle.
static String	NORTH	It specifies the top edge of a component's bounding rectangle.
static String	SOUTH	It specifies the bottom edge of a component's bounding rectangle.
static String	VERTICAL_CENTER	It specifies the vertical center of a component's bounding rectangle.
static String	WEST	It specifies the left edge of a component's bounding rectangle.
static String	WIDTH	It specifies the width of a component's bounding rectangle.

SpringLayout Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component component, Object constraints)	If constraints is an instance of SpringLayout. Constraints, associates the constraints with the specified component.
void	addLayoutComponent(String name, Component c)	Has no effect, since this layout manager does not use a per-component string.

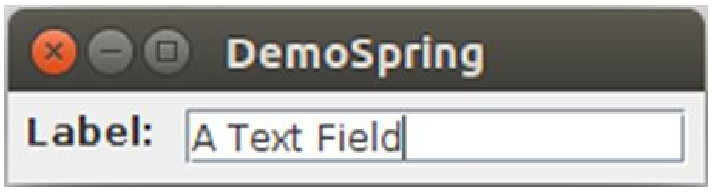
Spring	getConstraint(String edgeName, Component c)	It returns the spring controlling the distance between the specified edge of the component and the top or left edge of its parent.
SpringLayout.	getConstraints(Component c)	It returns the constraints for the
Constraints		specified component.
float	getLayoutAlignmentX(Container p)	It returns 0.5f (centered).
float	getLayoutAlignmentY(Container p)	It returns 0.5f (centered).
void	invalidateLayout(Container p)	It Invalidates the layout, indicating that if the layout manager has cached information it should be discarded.
void	layoutContainer(Container parent)	It lays out the specified container.
Dimension	maximumLayoutSize(Container parent)	It is used to calculates the maximum size dimensions for the specified container, given the components it contains.
Dimension	minimumLayoutSize(Container parent)	It is used to calculates the minimum size dimensions for the specified container, given the components it contains.
Dimension	preferredLayoutSize(Container parent)	It is used to calculates the preferred size dimensions for the specified container, given the components it contains.

Example

```
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SpringLayout;
public class MySpringDemo {
    private static void createAndShowGUI() {
          JFrame frame = new JFrame("MySpringDemp");
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          Container contentPane = frame.getContentPane();
```

```
        SpringLayout layout = new SpringLayout();
        contentPane.setLayout(layout);
        JLabel label = new JLabel("Label: ");
        JTextField textField = new JTextField("My Text Field", 15);
        contentPane.add(label);
        contentPane.add(textField);
        layout.putConstraint(SpringLayout.WEST, label,6,SpringLayout.WEST, contentPane);
        layout.putConstraint(SpringLayout.NORTH, label,6,SpringLayout.NORTH, contentPane);
        layout.putConstraint(SpringLayout.WEST, textField,6,SpringLayout.EAST, label);
        layout.putConstraint(SpringLayout.NORTH, textField,6,SpringLayout.NORTH, contentPane);
        layout.putConstraint(SpringLayout.EAST, contentPane,6,SpringLayout.EAST, textField);
        layout.putConstraint(SpringLayout.SOUTH, contentPane,6,SpringLayout.SOUTH, textField);

        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Output:



4.4.8 JTable

Q28. Write a short notes on JTable.

Ans : **(July-19)**

JTable is a flexible Swing component which is very well-suited at displaying data in a tabular format. Sorting rows by columns is a nice feature provided by the JTable class. In this tutorial, you will learn some fundamental techniques for sorting rows in JTable, from enable sorting, sort a column programmatically to listen to sorting event.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

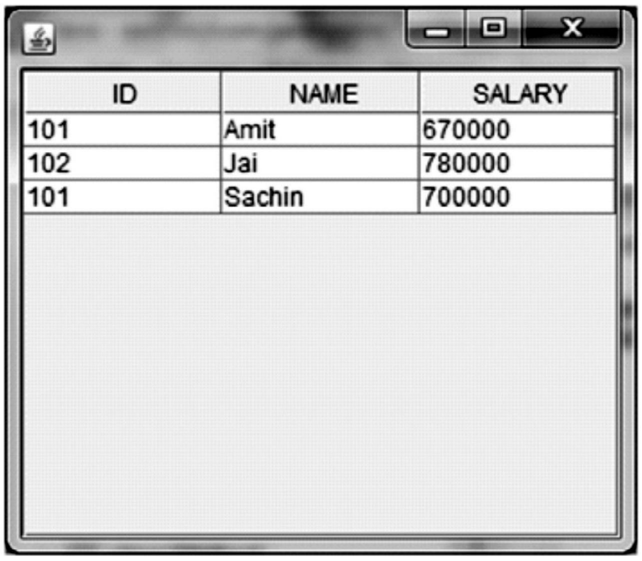
Commonly used Constructors

Constructor	Description
JTable()	Creates a table with empty cells.
JTable (Object[][] rows,Object [] columns)	Creates a table with the specified data.

Program

```
import javax.swing.*;
public class TableExample
{
      JFrame f;
      TableExample()
{
      f=new JFrame();
String data[][]={{"101","Amit","670000"}, { "102","Jai","780000"}, { "101","Sachin","700000"}
};
      String column[]={"ID","NAME","SALARY"};
      JTable jt=new JTable(data,column);
      jt.setBounds(30,40,200,300);
      JScrollPane sp=new JScrollPane(jt);
      f.add(sp);
      f.setSize(300,400);
      f.setVisible(true);
      }
      public static void main(String[] args)
      {
      new TableExample();
      }
}
```

Output



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

Short Question and Answers

1. What is purpose of layouts?

Ans :

The layout manager automatically positions all the components within the container. If we do not use layout manager then also the components are positioned by the default layout manager. It is possible to layout the controls by hand but it becomes very difficult because of the following two reasons.

➢ It is very tedious to handle a large number of controls within the container.

➢ Oftenly the width and height information of a component is not given when we need to arrange them.

Java provide us with various layout manager to position the controls. The properties like size,shape and arrangement varies from one layout manager to other layout manager. When the size of the applet or the application window changes the size, shape and arrangement of the components also changes in response i.e. the layout managers adapt to the dimensions of appletviewer or the application window.

The layout manager is associated with every Container object. Each layout manager is an object of the class that implements the LayoutManager interface.

2. Write about applet life cycle.

Ans :

Four methods in the Applet class gives you the framework on which you build any serious applet.

➢ **init** - This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.

➢ **start** - This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.

➢ **stop** - This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.

➢ **destroy** - This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

➢ **paint** - Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.

3. Java applet.

Ans :

An applet is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

There are some important differences between an applet and a standalone Java application, including the following.

➢ An applet is a Java class that extends the java.applet.Applet class.

➢ A main() method is not invoked on an applet, and an applet class will not define main().

➢ Applets are designed to be embedded within an HTML page.

➢ When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.

➢ A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.

4. Write short notes on Dialog box.

Ans :

(i) showConfirmationDialog(): It asks for the user confirmation.

(ii) showMessageDialog() : It is used to display a message and then allow the user to press OK button.

(iii) showInputDialog() : It is used to accept the input from user.

(iv) showOptionDialog(): It is used to create a dialog that contains the elements specified by the user.

5. What is the purpose of AWT?

Ans :

Unlike C/C++, Java also supports GUI environment. Through the use of GUI (Graphical User Interface) environments it is possible to take user input into a running program or it can display some message to the user dynamically. The user interfaces can either be command line or GUI. All the classes and interfaces required for developing GUI components and drawing graphics are provided by two packages i.e., java, awt and java.awt.event.

6. Compare and contrast AWT and Swing.

Ans :

S.No	AWT	Swing
1.	Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
2.	The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
3.	Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
4.	The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
5.	The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
6.	MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
7.	AWT provides comparatively less powerful components.	Swing provides more powerful components.

7. Write a short notes on JTable.

Ans :

JTable is a flexible Swing component which is very well-suited at displaying data in a tabular format. Sorting rows by columns is a nice feature provided by the JTable class. In this tutorial, you will learn some fundamental techniques for sorting rows in JTable, from enable sorting, sort a column programmatically to listen to sorting event.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors

Constructor	Description
JTable()	Creates a table with empty cells.
JTable (Object[][] rows,Object [] columns)	Creates a table with the specified data.

Program

```
import javax.swing.*;
public class TableExample
{
    JFrame f;
    TableExample()
{
    f=new JFrame();
String data[][]={{“101”,”Amit”,”670000"}, { “102”,”Jai”,”780000"}, { “101”,”Sachin”,”700000"}
};
    String column[]={“ID”,”NAME”,”SALARY”};
    JTable jt=new JTable(data,column);
    jt.setBounds(30,40,200,300);
    JScrollPane sp=new JScrollPane(jt);
    f.add(sp);
    f.setSize(300,400);
    f.setVisible(true);
    }
    public static void main(String[] args)
    {
    new TableExample();
    }
}
```

8. Event.

Ans :

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. For example, clicking on a button, moving the mouse, entering a character through keyboard,selecting an item from list, scrolling the page are the activities that causes an event to happen.

9. Event Handling.

Ans :

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.Let's have a brief introduction to this model.

The Delegation Event Model has the following key participants namely:

- **Source** - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to it's handler. Java provide as with classes for source object.
- **Listener** - It is also known as event handler.Listener is responsible for generating response to an event. From java implemen- tation point of view the listener is also an object. Listener waits until it receives an event. Once the event is received , the listener process the event an then returns.

10. What are swings?

Ans :

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Features

(i) Light Weight

Swing components 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.

(ii) Rich Controls

Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.

(iii) Highly Customizable

Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.

(iv) Pluggable look-and-feel

SWING based GUI Application look and feel can be changed at run-time, based on available values.

11. Explain about JPanel by giving an example.

Ans :

The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.

It doesn't have title bar.

Class Declaration

Following is the declaration

for **javax.swing.JPanel** class

public class JPanel extends JComponent implements Accessible

Program

```
import java.awt.*;
import javax.swing.*;
public class PanelExample
{
        PanelExample()
        {
        JFrame f= new JFrame("Panel Example");
        JPanel panel=new JPanel();
        panel.setBounds(40,80,200,200);
        panel.setBackground(Color.gray);
        JButton b1=new JButton("Button 1");
        b1.setBounds(50,100,80,30);
        b1.setBackground(Color.yellow);
        JButton b2=new JButton("Button 2");
        b2.setBounds(100,100,80,30);
        b2.setBackground(Color.green);
        panel.add(b1); panel.add(b2);
        f.add(panel);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
        public static void main(String args[])
{
        new PanelExample();
}
}
```

12. Spring Layout

Ans :

A SpringLayout arranges the children of its associated container according to a set of constraints. Constraints are nothing but horizontal and vertical distance between two-component edges. Every constraint is represented by a SpringLayout. Constraint object.

Each child of a SpringLayout container, as well as the container itself, has exactly one set of constraints associated with them.

Each edge position is dependent on the position of the other edge. If a constraint is added to create a new edge, than the previous binding is discarded. SpringLayout doesn't automatically set the location of the components it manages.

Choose the Correct Answers

1. Which is used to run an applet [c]
   (a) An HTML File (b) Applet viewer Tool
   (c) Both a& b (d) None
2. AWT Stands for ________. [a]
   (a) Abstract Window Tool Kit (b) All window tools
   (c) Abstract Writing Tools (d) All Writing Tools
3. Which of these functions is called to display the output in applets [a]
   (a) display() (b) print()
   (c) displayapplet() (d) printapplet()
4. Event class is defined in which of these package [d]
   (a) io (b) lang
   (c) net (d) util
5. Which of these events is generated when button is pressed? [a]
   (a) Action Event (b) Key Event
   (c) Window Event (d) None
6. Which is the container containing title bar [a]
   (a) panel (b) frame
   (c) window (d) container
7. AWT is used for GUI Programming in java [a]
   (a) True (b) False
   (c) maybe (d) none
8. Which of these functions is called to display the output of an applet? [b]
   (a) display() (b) paint()
   (c) displayApplet() (d) PrintApplet()
9. Which of these methods can be used to output a string in an applet? [c]
   (a) display() (b) print()
   (c) drawString() (d) transient()
10. Which of these methods is a part of Abstract Window Toolkit (AWT)? [b]
   (a) display() (b) paint()
   (c) drawString() (d) transient()

Fill in the Blanks

1. ________ is a java program that runs in a web browser.
2. The applet is started with a method ________.
3. AWT stands for________.
4. ________ is used to turn an option on/off.
5. ________display a single line of read only text.
6. A border layout arranges/ fits the components in ________ regions.
7. Data is arranged in left to right in ________ lay out.
8. ________ lay out is used to arrange the data in vertical and horizontal positions.
9. ________ are light weighted components in java.
10. ________ lay out is default layout for dialog box.

ANSWERS

1. Applet
2. init()
3. Abstract Window Toolkit
4. Checkbox
5. Label
6. Five(5)
7. Flow
8. Gridbag
9. Swing Components
10. Border

One Mark Answers

1. Applets

Ans :

An applet is a Java class that extends the java.applet.Applet class.

- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.

2. AWT

Ans :

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.

3. Swings.

Ans :

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.

4. JApplet.

Ans :

JApplet is a class that represents the Swing applet. It is a subclass of Applet class and must be extended by all the applets that use Swing.

5. BoxLayout.

Ans :

The Java BoxLayout class is used to arrange the components either vertically or horizontally.

Lab Practicals

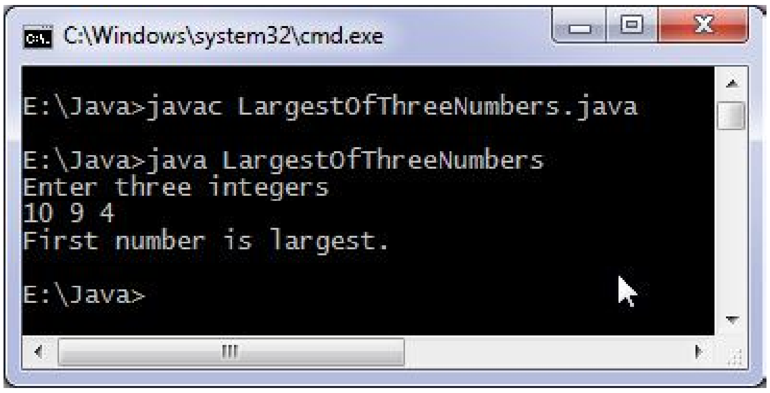
1. Write a program to find the largest of n natural numbers.

Ans :

Largest of given three numbers

```
import java.util.Scanner;
classLargestOfThreeNumbers
{
public static void main(String args[])
{
      int x, y, z;
      System.out.println("Enter three integers ");
          Scanner in = new Scanner(System.in);
          x = in.nextInt();
          y = in.nextInt();
          z = in.nextInt();
      if ( x > y && x > z )
      System.out.println("First number is largest.");
      else if ( y > x && y > z )
      System.out.println("Second number is largest.");
      else if ( z > x && z > y )
      System.out.println("Third number is largest.");
      else
      System.out.println("Entered numbers are not distinct.");
      }
}
```

Output



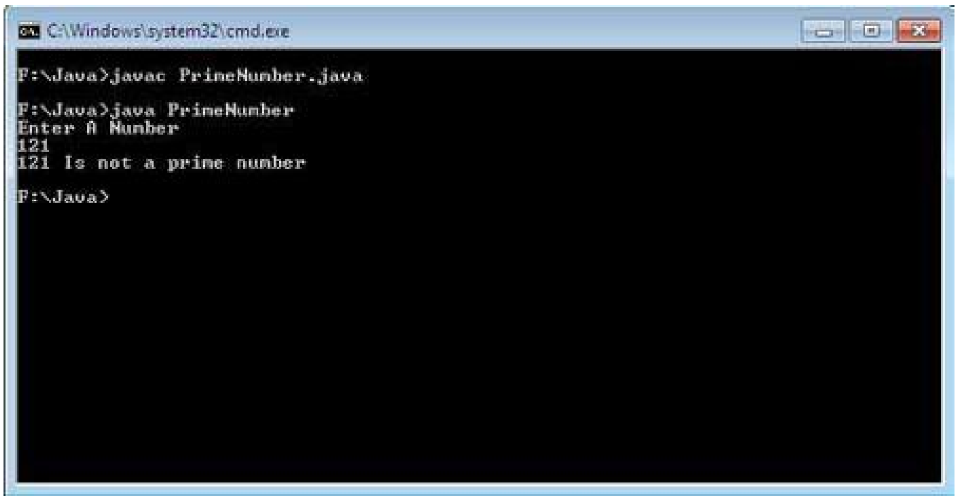
```
E:\Java>javac LargestOfThreeNumbers.java

E:\Java>java LargestOfThreeNumbers
Enter three integers
10 9 4
First number is largest.

E:\Java>
```

2. Write a program to find whether a given number is prime or not.

Ans :

```
import java.util.Scanner;
public class PrimeNumber
{
        public static void main(String args[])
                {
                        int num,b,c;
                        Scanner s=new Scanner(System.in);
                        System.out.println("Enter A Number");
                        num =s.nextInt();
                        b=1;
                        c=0;
                        while(b<= num)
                                {
                                        if((num%b)==0)
                                        c=c+1;
                                        b++;
                                }
                                        if(c==2)
                                        System.out.println(num +" is a prime number");
                                        else
                                        System.out.println(num +" is not a prime number");
                        }
        }
```

Output



C:\Windows\system32\cmd.exe

```
F:\Java>javac PrimeNumber.java

F:\Java>java PrimeNumber
Enter A Number
121
121 Is not a prime number

F:\Java>
```

3. Write a menu driven program for following:

(a) Display a Fibonacci series

(b) Compute Factorial of a number

Ans :

(a) Display a Fibonacci series

For answer refer to Unit-I, Q.No. 12 (While loop)

(b) Compute Factorial of a number

For answer refer to Unit-I, Q.No. 12 (Do.While loop)

4. Write a program to check whether a given number is odd or even.

Ans :

```
        import java.util.Scanner;
        public class Odd_Even
        {
                public static void main(String[] args)
        {
                int n;
                Scanner s = new Scanner(System.in);
                System.out.print("Enter the number you want to check:");
                n = s.nextInt();
                if(n % 2 == 0)
        {
                System.out.println("The given number "+n+" is Even ");
        }
        else
        {
                System.out.println("The given number "+n+" is Odd ");
        }
    }
}
```

Output

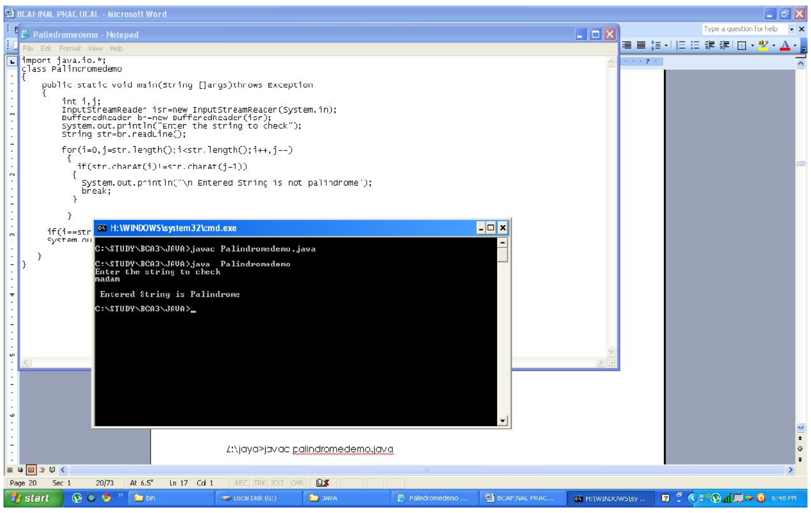
```
javac Odd_Even.java
java Odd_Even
Enter the number you want to check:15
The given number 15 is Odd
```

5. Write a program to check whether a given string is palindrome or not.

Ans :

```
import java.io.*;
class Palindromedemo
{
public static void main(String []args) throws Exception
{
      int i,j;
      InputStreamReaderisr=new InputStreamReader(System.in);
      BufferedReaderbr=new BufferedReader(isr);
      System.out.println("Enter the string to check");
      String str=br.readLine();
      for(i=0,j=str.length();i<str.length();i++,j—)
{
      if(str.charAt(i)!=str.charAt(j-1))
 {
      System.out.println("\n Entered String is not palindrome");
      break;
      }
}
if(i==str.length())
System.out.println("\n Entered String is Palindrome");
  }
}
```

Output



```
C:\STUDY\BCA3\JAVA>javac Palindromedemo.java

C:\STUDY\BCA3\JAVA>java  Palindromedemo
Enter the string to check
madam

 Entered String is Palindrome

C:\STUDY\BCA3\JAVA>_
```

6. Write a program to print the sum and product of digits of an Integer and reverse the Integer

Ans :

```
import java.util.Scanner;
public class Use_Do_While
{
   public static void main(String[] args)
   {
      int n, a, m = 0, sum = 0;
      Scanner s = new Scanner(System.in);
       System.out.print("Enter any number:");
      n = s.nextInt();
      do
      {
            a = n % 10;
            m = m * 10 + a;
            sum = sum + a;
            n = n / 10;
       }
       while( n > 0);
            System.out.println("Reverse:"+m);
            System.out.println("Sum of digits:"+sum);
       }
}
```

Output

```
javac Use_Do_While.java
java Use_Do_While
Enter any number:456
Reverse:654
Sum of digits:15
```

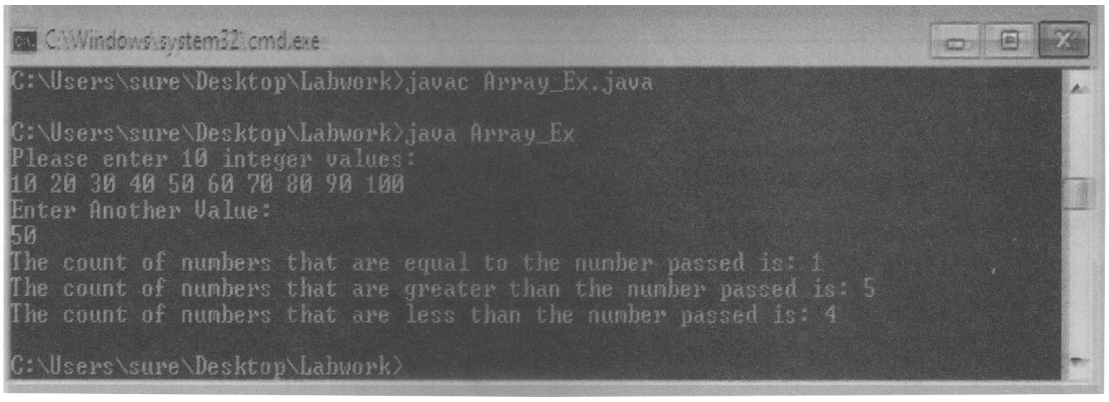
7. Write a program to create an array of 10 integers. Accept values from the user in that Array. Input another number from the user and find out how many numbers are equal to the number passed, how many are greater and how many are less than the number passed.

Ans :

```
       import j ava.util. Scanner;
             public class Array_Ex
             {
                    public static void main(String[] args)
             {
                    int i,n;
```

```
                int equal_count=0, greater_count=0, less_count=0;
                Scanner sc=new Scanner(System.in);
                int[] arr = new int[10];
                System.out.println("Please enter 10 integer values: ");
                for(i=0; i<10; i++)
                {
                        arr[i]=sc.nextInt();
                }
                System.out.println("Enter Another Value: ");
                n=sc.nextInt();
                for(i=0; i<10; i++)
                {
                        if(arr[i]==n)
                        {
                                equal_count++;
                        }
                        else if(arr[i]>n)
                        {
                                greater_count++;
                        {
                        else
                        {
                                less_count++;
                        }
                }
                System.out.println("The count of numbers that are equal to the number passed is:
                "+equal_count);
                System.out.println("The count of numbers that are greater than the number passed is:
                "+greater_count);
                System.out.println("The count of numbers that are less than the number passed is:
                "+less_count);
        }
}
```

Output



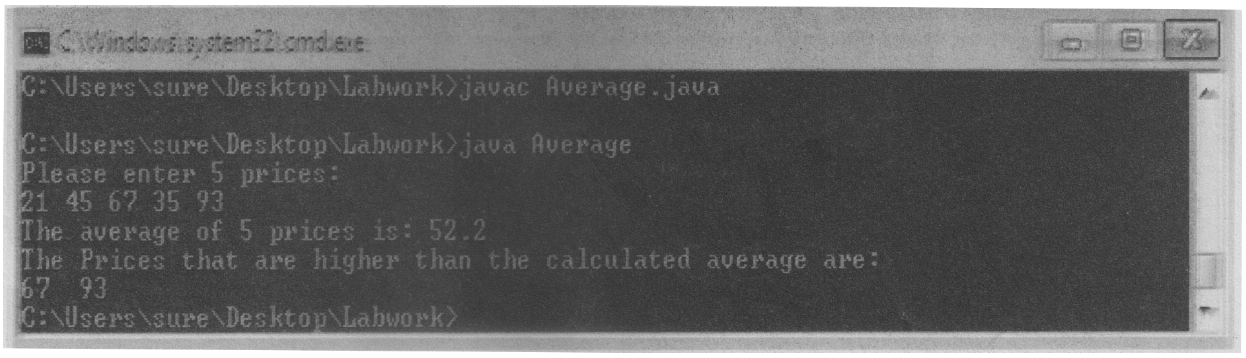
```
C:\Users\sure\Desktop\Labwork>javac Array_Ex.java

C:\Users\sure\Desktop\Labwork>java Array_Ex
Please enter 10 integer values:
10 20 30 40 50 60 70 80 90 100
Enter Another Value:
50
The count of numbers that are equal to the number passed is: 1
The count of numbers that are greater than the number passed is: 5
The count of numbers that are less than the number passed is: 4

C:\Users\sure\Desktop\Labwork>
```

8. Write a program that will prompt the user for a list of 5 prices. Compute the average of the prices and find out all the prices that are higher than the calculated average.

Ans :

```
import java.util.Scanner;
public class Average
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        int[] arr = new int[5];
        float total = 0, avg;
        System.out.println("Please enter 5 prices: ");
        for(int i=0; i<5; i++)
        {
            arr[i]=sc.nextInt();
        }
        for (int i=0; i<5; i++)
        {
            total = total+ arr[i];
        }
        avg = total/5;
        System.out.println("The average of 5 prices is: "+ avg);
        System.out.println("The Prices that are higher than the calculated average are: ");
        for (int i=0; i<5; i++)
        {
            if(arr[i]>avg)
                System.out.print(arr[i]+" ");
            else
                continue;
        }
    }
}
```

Output



```
C:\Users\sure\Desktop\Labwork>javac Average.java

C:\Users\sure\Desktop\Labwork>java Average
Please enter 5 prices:
21 45 67 35 93
The average of 5 prices is: 52.2
The Prices that are higher than the calculated average are:
67 93
C:\Users\sure\Desktop\Labwork>
```

9. Write a program in java to input N numbers in an array and print out the Armstrong numbers from the set.

Ans :

```
public class Armstrong
{
     public static void main(String[] args)
     {
          int n, count = 0, a, b, c, sum = 0;
          System.out.print("Armstrong numbers from 1 to 1000:");
          for(int i = 1; i <= 1000; i++)
     {
          n = i;
          while(n > 0)
     {
          b = n % 10;
          sum = sum + (b * b * b);
          n = n / 10;
     }
          if(sum == i)
     {
          System.out.print(i+" ");
     }
     sum = 0;
     }
  }
}
```

Output

```
javac Armstrong.java
java  Armstrong
Armstrong numbers from 1 to 1000:1 153 370 371 407
```

10. Write java program for the following matrix operations:

(a) Addition of two matrices

(b) Transpose of a matrix

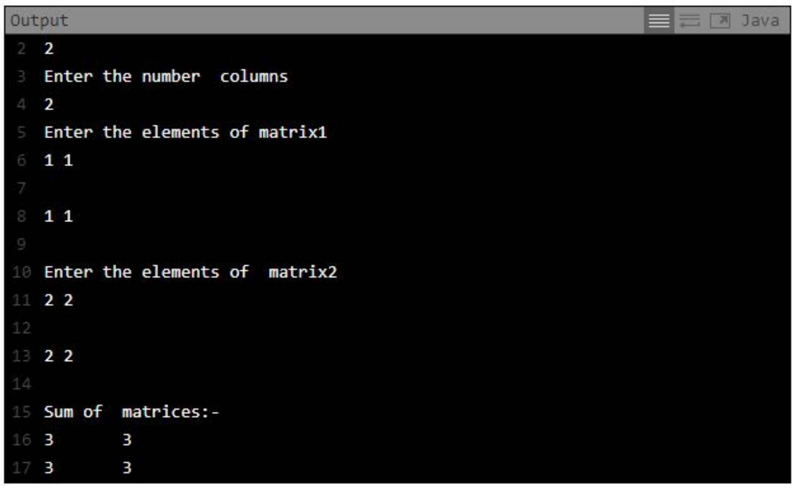
Ans :

(a) Addition of two matrices

```
class AddMatrix
{
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 mat1[][] = new int[row][col];
        int mat2[][] = new int[row][col];
        int res[][] = new int[row][col];
        System.out.println("Enter the elements of matrix1");
        for ( i= 0 ; i < row ; i++ )
        {
              for ( j= 0 ; j < col ;j++ )
              mat1[i][j] = in.nextInt();
              System.out.println();
        }
              System.out.println("Enter the elements of matrix2");

        for ( i= 0 ; i < row ; i++ )
        {
        for ( j= 0 ; j < col ;j++ )
        mat2[i][j] = in.nextInt();
        System.out.println();
        }
              for ( i= 0 ; i < row ; i++ )
              for ( j= 0 ; j < col ;j++ )
              res[i][j] = mat1[i][j] + mat2[i][j] ;
              System.out.println("Sum of matrices:-");
              for ( i= 0 ; i < row ; i++ )
        {
              for ( j= 0 ; j < col ;j++ )
              System.out.print(res[i][j]+"\t");
              System.out.println();
        }

    }
}
```

Output



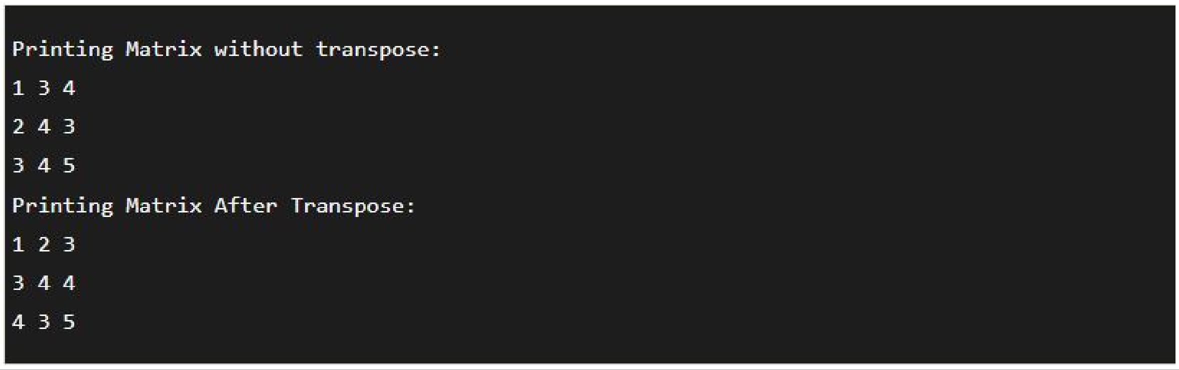
```
Output
2
Enter the number  columns
2
Enter the elements of matrix1
1 1

1 1

Enter the elements of  matrix2
2 2

2 2

Sum of  matrices:-
3       3
3       3
```

(b) Transpose of a matrix

```
public static void main(String args[]){
//creating a matrix
int original[][]={{1,3,4},{2,4,3},{3,4,5}};
//creating another matrix to store transpose of a matrix
int transpose[][]=new int[3][3];  //3 rows and 3 columns
//Code to transpose a matrix
for(int i=0;i<3;i++){
for(int j=0;j<3;j++){
transpose[i][j]=original[j][i];
}
    }
        System.out.println("Printing Matrix without transpose:");
        for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
        System.out.print(original[i][j]+" ");
    }
    System.out.println();//new line
    }
        System.out.println("Printing Matrix After Transpose:");
        for(int i=0;i<3;i++){
        for(int j=0;j<3;j++){
        System.out.print(transpose[i][j]+" ");
    }
        System.out.println();//new line
    }
  }
}
```

Test it Now

Output



```
Printing Matrix without transpose:
1 3 4
2 4 3
3 4 5
Printing Matrix After Transpose:
1 2 3
3 4 4
4 3 5
```

11. Write a java program that computes the area of a circle, rectangle and a Cylinder using function overloading.

Ans :

```
        class OverloadDemo
        {
                void area(float x)
                {
                        System.out.println("the area of the square is "+Math.pow(x, 2)+" sq units");
                }
                void area(float x, float y)
                {
                        System.out.println("the area of the rectangle is "+x*y+" sq units");
        }
        void area(double x)
        {
                double z = 3.14 * x * x;
                System.out.println("the area of the circle is "+z+" sq units");
                }
        }
        class Overload
        {
                public static void main(String args[])
        {
                OverloadDemo ob = new OverloadDemo();
                ob.area(5);
                ob.area(11,12);
                ob.area(2.5);
        }
}
```

Output

```
javac OverloadDemo.java
java OverloadDemo
the area of the square is 25.0 sq units
the area of the rectangle is 132.0 sq units
the area of the circle is 19.625 sq units
```

12. Write a Java program for the implementation of multiple inheritance using interfaces to calculate the area of a rectangle and triangle.

Ans :

For answer refer to Unit-II, Q.No. 21 (Multiple Inheritance)

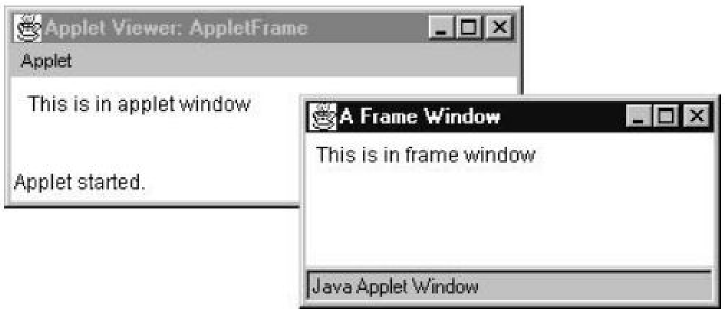
13. Write a java program to create a frame window in an Applet.

Ans :

```
// Create a child frame window from within an applet.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
    <applet code="AppletFrame" width=400 height=60>
    </applet>
*/
// Create a subclass of Frame.
    class SampleFrame extends Frame {
    SampleFrame(String title) {
    super(title);
// create an object to handle window events
    MyWindowAdapter adapter = new MyWindowAdapter(this);
// register it to receive those events
    addWindowListener(adapter);
}
public void paint(Graphics g) {
g.drawString("This is in frame window", 10, 40);
}
}
class MyWindowAdapter extends WindowAdapter
{
SampleFrame sampleFrame;
public MyWindowAdapter(SampleFrame sampleFrame)
{
this.sampleFrame = sampleFrame;
}
```

```
              public void windowClosing(WindowEvent we) {
              sampleFrame.setVisible(false);
              }
       }
       // Create frame window.
              public class AppletFrame extends Applet {
              Frame f;
              public void init() {
              f = new SampleFrame("A Frame Window");
              f.setSize(150, 150);
              f.setVisible(true);
       }
       public void start() {
       f.setVisible(true);
       }
       public void stop() {
       f.setVisible(false);
       }
              public void paint(Graphics g) {
              g.drawString("This is in applet window", 15, 30);
       }
}
```

OUTPUT



```
Applet Viewer: AppletFrame
Applet
This is in applet window
Applet started.

A Frame Window
This is in frame window
Java Applet Window
```

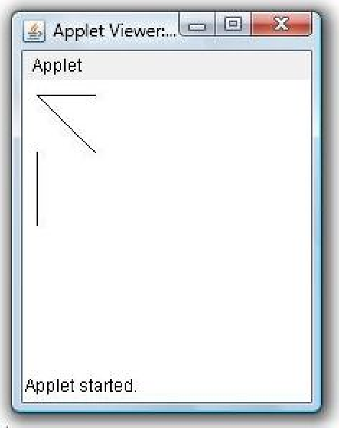
14. Write a java program to draw a line between two coordinates in a window.

Ans :

```
       import java.applet.Applet;
       import java.awt.Graphics;
              public class DrawLineExample extends Applet
              {
              public void paint(Graphics g)
              {
        /*
```

```
 * to draw line in an applet window use,
 * void drawLine(int x1,int y1, int x2, int y2)
 * method.
 *
 * This method draws a line between (x1,y1) and (x2,y2)
 * coordinates in a current color.
*/
 //this will draw a line between (10,10) and (50,50) coordinates.
g.drawLine(10,10,50,50);
//draw vertical line
g.drawLine(10,50,10,100);
//draw horizontal line
g.drawLine(10,10,50,10);
}
}
```

Example Output



15. Write a java program to display the following graphics in an applet window.

(a) Rectangles

(b) Circles

(c) Ellipses

(d) Arcs

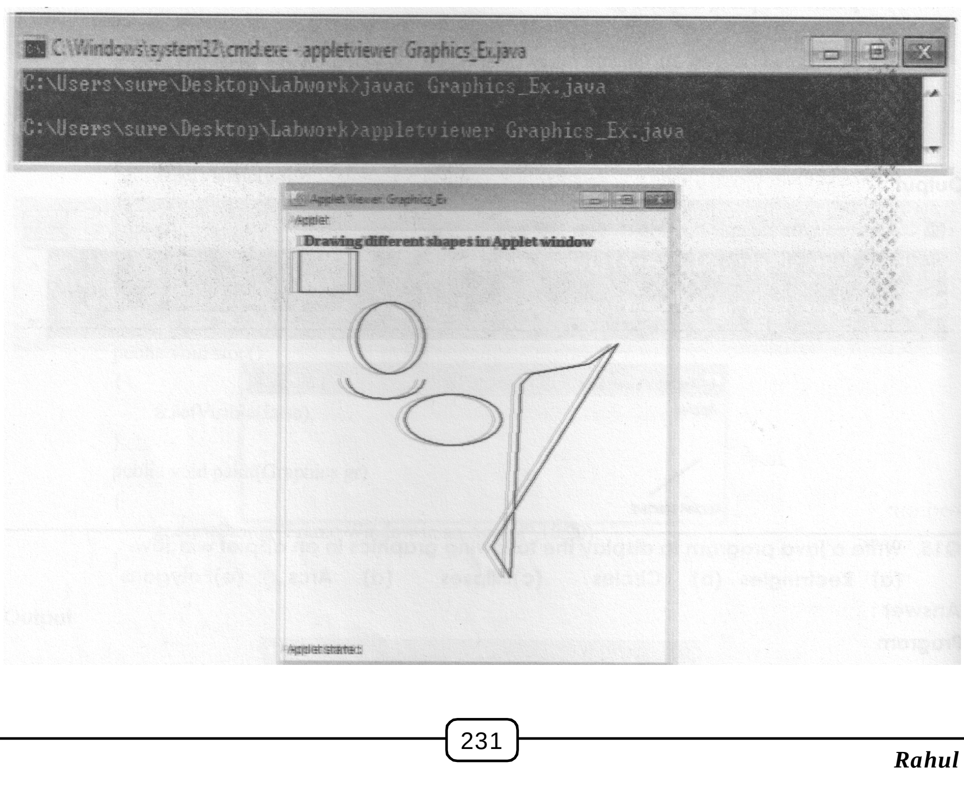
(e) Polygons

Ans :

```
import java.applet. Applet;
import java.awt.Graphics;
/*
<applet code="Graphics_Ex" width=400 height=400>
```

```
</applet>
*/
import java.awt.*;
import j ava. applet. *;
public class GraphicsEx extends Applet
{
    public void paint(Graphics g)
    {
        g.setFont(new Font("Cambria", Font.BOLD,15));
        g.drawString("Drawing different shapes in Applet window", 15, 15);
        g.drawRect( 10,20,60,40); //drawing rectangle
        g.drawOval(70, 70, 70, 70); //drawing circle
        g.drawOval(120, 160, 100, 50);//drawing ellipse
        g.drawArc(60, 125, 80, 40, 180,180); //drawing arc
        int x[] = { 210,230, 240, 250, 310, 340 };
        int y[] = { 310, 340, 150, 140, 130, 110 };
        int n = 6;
        Polygon pg = new Polygon(x, y, n);
        g.drawPolygon(pg); //drawing polygon
    }
}
```

Output:



```
C:\Windows\system32\cmd.exe - appletviewer Graphics_Ex.java
C:\Users\sure\Desktop\Labwork>javac Graphics_Ex.java

C:\Users\sure\Desktop\Labwork>appletviewer Graphics_Ex.java
```

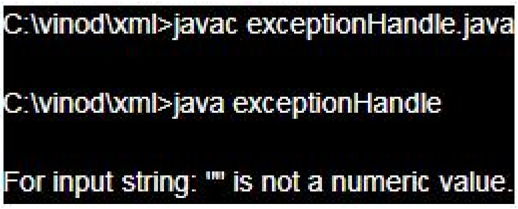
Applet Viewer: Graphics_Ex
Applet
Drawing different shapes in Applet window
Applet started.

16. Write a program that reads two integer numbers for the variables a and b. If any other character except number (0-9) is entered then the error is caught by NumberFormatException object. After that ex.getMessage () prints the information about the error occurring causes.

Ans :

```
import java.io.*;
public class exceptionHandle
        {
        public static void main(String[] args) throws Exception{
        try
        {
        int a, b;
        BufferedReader in =
        new BufferedReader(new InputStreamReader(System.in));
        a = Integer.parseInt(in.readLine());
        b = Integer.parseInt(in.readLine());
        }
            catch (NumberFormatException ex){
            System.out.println(ex.getMessage() + " is not a numeric value.");
            System.exit(0);
        }
      }
}
```

Output



```
C:\vinod\xml>javac exceptionHandle.java

C:\vinod\xml>java exceptionHandle

For input string: "" is not a numeric value.
```

17. Write a program for the following string operations:

(a) Compare two strings

(b) Concatenate two strings

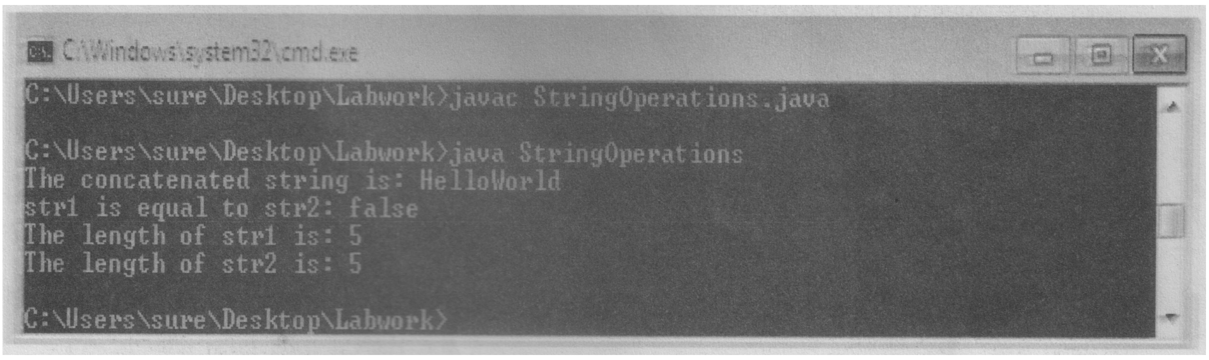
(c) Compute length of a string

Ans :

```
        import java.io.*;
        import java.util.*;
        class StringOperations
        {
```

```
        public static void main (String[] args)
        {
                boolean flag;
                String strl= "Hello";
                String str2 = "World";
                //String Concatenation
                System.out.println("The concatenated string is: " +strl .concat(str2));
                //String Comparison
                flag = strl.equals(str2);
                System.out.println("strl is equal to str2: "+flag);
                //String Length
                System.out.println("The length of strl is: " + strl.length/));
                System.out.println("The length of str2 is: " + str2.1ength());
        }
}
```

Output



```
C:\Users\sure\Desktop\Labwork>javac StringOperations.java

C:\Users\sure\Desktop\Labwork>java StringOperations
The concatenated string is: HelloWorld
str1 is equal to str2: false
The length of str1 is: 5
The length of str2 is: 5

C:\Users\sure\Desktop\Labwork>
```

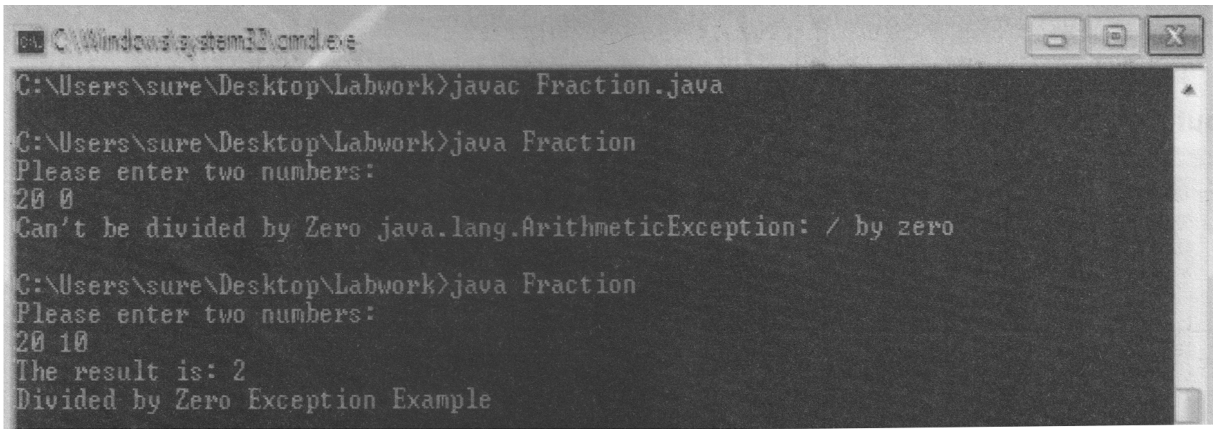
18. Create a class called Fraction that can be used to represent the ratio of two integers. Include appropriate constructors and methods. If the denominator becomes zero, throw and handle an exception.

Ans :

```
import java.util.Scanner; class Fraction
{
        public Fraction() throws ArithmeticException
{
        Scanner sc=new Scanner(System.in);
        System.out.println("Please enter two numbers: ");
        int numerator = sc.nextlnt();
```

```
            int denominator = sc.nextInt();
            int result = numerator/denominator;
            System.out.println("The result is: " +result);
        }
        void display()
        {
        System.out.println("Divided by Zero Exception Example");
        }
        public static void main(String args[])
        {
        try
        {
        Fraction f = new Fraction(); f.display();
        }
        catch(ArithmeticException e)
        {
        System.out.println ("Can't be divided by Zero " + e);
        }
      }
}
```

Output



```
C:\Users\sure\Desktop\Labwork>javac Fraction.java

C:\Users\sure\Desktop\Labwork>java Fraction
Please enter two numbers:
20 0
Can't be divided by Zero java.lang.ArithmeticException: / by zero

C:\Users\sure\Desktop\Labwork>java Fraction
Please enter two numbers:
20 10
The result is: 2
Divided by Zero Exception Example
```

FACULTY OF SCIENCE

B.Sc. III Year V-Semester(CBCS) Examination

Model Paper - I

Paper - V : PROGRAMMING IN JAVA

Time : 3 Hours] [Max. Marks : 80

PART- A (8 × 4 = 32 Marks)

ANSWERS

Note : Answer any **Eight** of the following questions

1.	Type conversion in java.	**(Unit-I, SQA-1)**
2.	JVM	**(Unit-I, SQA-6)**
3.	Type casting.	**(Unit-I, SQA-7)**
4.	What are wrapper class?	**(Unit-II, SQA-9)**
5.	What is the major difference between an interface and a class?	**(Unit-II, SQA-7)**
6.	Discuss constructors overloading with super keyword.	**(Unit-II, SQA-2)**
7.	FileOutputStream Class.	**(Unit-III, SQA-5)**
8.	Multitasking.	**(Unit-III, SQA-11)**
9.	Exception Handling.	**(Unit-III, SQA-10)**
10.	Write about applet life cycle.	**(Unit-IV, SQA-2)**
11.	Write short notes on Dialog box.	**(Unit-IV, SQA-4)**
12.	What is purpose of layouts?	**(Unit-IV, SQA-1)**

SECTION - B (4 × 12 = 48 Marks)

Note : Answer **ALL** the following questions

13.	Explain the architecture of JVM.	**(Unit-I, Q.No.4)**
	(OR)	
14.	Explain about the type casting and type conversion in java.	**(Unit-I, Q.No.9)**
15.	Define Constructors. Explain different types of constructors in java.	**(Unit-II, Q.No.6)**
	(OR)	
16.	What are the different types of Inheritance?	**(Unit-II, Q.No.19)**

17. Define Threads. List different ways of creating threads. Explain with examples. **(Unit-III, Q.No.11)**

(OR)

18. What is java.io Package? Explain the different ways of input output methods. **(Unit-III, Q.No.15)**

19. Explain the sequence of applet's life cycle. **(Unit-IV, Q.No.2)**

(OR)

20. Explain different layout managers in java with examples. **(Unit-IV, Q.No.20)**

FACULTY OF SCIENCE

B.Sc. III Year V-Semester(CBCS) Examination

Model Paper - II

Paper - V : PROGRAMMING IN JAVA

Time : 3 Hours] [Max. Marks : 80

PART- A (8 × 4 = 32 Marks)

ANSWERS

Note : Answer any **Eight** of the following questions

1. Explain the features of Java. **(Unit-I, SQA-2)**
2. Object in Java. **(Unit-I, SQA-10)**
3. Garbage Collector. **(Unit-I, SQA-9)**
4. Define Constructors. **(Unit-II, SQA-1)**
5. What is an abstract class? **(Unit-II, SQA-3)**
6. Protection access. **(Unit-II, SQA-5)**
7. Write about Multithreading in Java. **(Unit-III, SQA-3)**
8. What is exception? **(Unit-III, SQA-1)**
9. Synchronization **(Unit-III, SQA-8)**
10. Java applet. **(Unit-IV, SQA-3)**
11. Event Handling. **(Unit-IV, SQA-9)**
12. Explain about JPanel by giving an example. **(Unit-IV, SQA-11)**

SECTION - B (4 × 12 = 48 Marks)

Note : Answer **ALL** the following questions

13. Explain the features of Java. **(Unit-I, Q.No.5)**

(OR)

14. What are the various types of Control Statements in Java with a suitable examples. **(Unit-I, Q.No.10)**
15. Explain about Command-Line Arguments. How they are useful. **(Unit-II, Q.No.15)**

(OR)

16. Differentiate the use of Abstract class and Interface? **(Unit-II, Q.No.32)**

17. Explain the handling of exception in Java. **(Unit-III, Q.No.3)**

(OR)

18. Explain in detail about FileInputStream Class with an example. **(Unit-III, Q.No.17)**

19. What are AWT's? Explain AWT Hierarchy in JAVA? **(Unit-IV, Q.No.11)**

(OR)

20. What are the differences between AWT and Swing? **(Unit-IV, Q.No.22)**

FACULTY OF SCIENCE

B.Sc. III Year V-Semester(CBCS) Examination

Model Paper - III

Paper - V : PROGRAMMING IN JAVA

Time : 3 Hours] [Max. Marks : 80

PART- A (8 × 4 = 32 Marks)

ANSWERS

Note : Answer any **Eight** of the following questions

1.	Explain the features of Java.	**(Unit-I, SQA-2)**
2.	Define Java.	**(Unit-I, SQA-4)**
3.	Java Essentials.	**(Unit-I, SQA-5)**
4.	Define Interface?	**(Unit-II, SQA-4)**
5.	What is a package in java?	**(Unit-II, SQA-6)**
6.	What is multiple Inheritance?	**(Unit-II, SQA-10)**
7.	Define thread.	**(Unit-III, SQA-6)**
8.	Benefits of Exception Handling.	**(Unit-III, SQA-9)**
9.	What is a scanner class?	**(Unit-III, SQA-12)**
10.	What is the purpose of AWT?	**(Unit-IV, SQA-5)**
11.	Compare and contrast AWT and Swing.	**(Unit-IV, SQA-6)**
12.	What are swings?	**(Unit-IV, SQA-10)**

SECTION - B (4 × 12 = 48 Marks)

Note : Answer **ALL** the following questions

13. Describe the Structure of a Java Program. **(Unit-I, Q.No.8)**

(OR)

14. What are the different looping statements in java? Explain. **(Unit-I, Q.No.12)**

15. What is Single Inheritance in JAVA? Explain with an Example. **(Unit-II, Q.No.20)**

(OR)

16. Explain the usage of string buffers class. **(Unit-II, Q.No.39)**

17. Discuss the Priority of threads used in JAVA? **(Unit-III, Q.No.13)**

(OR)

18. Explain in detail about FileOutputStream Class with an example. **(Unit-III, Q.No.18)**

19. What are the Components of AWT? **(Unit-IV, Q.No.13)**

(OR)

20. Discuss about Layout Managers in swing. **(Unit-IV, Q.No.27)**

FACULTY OF SCIENCE

B.Sc V-Semester (CBCS) Examination

July - 2021

COMPUTER SCIENCE (PROGRAMMING IN JAVA)

Paper - V

Time : 2 Hrs] [Max. Marks : 60

PART - A (4 × 5 = 20 Marks)
Note : Answer any FOUR questions

		Answers
1.	What is constructor? Discuss constructor overloading with super keyword.	**(Unit-II, SQA- 1, 2)**
2.	Write about abstract classes and interfaces.	**(Unit-II, SQA- 3, 4)**
3.	What is exception? Discuss user-defined exception.	**(Unit-III, SQA- 1, 2)**
4.	Write about protection access.	**(Unit-II, SQA- 5)**
5.	What is the purpose of layouts?	**(Unit-IV, SQA- 1)**
6.	Write about applet life cycle.	**(Unit-IV, SQA- 2)**
7.	Define thread. List the methods in thread class.	**(Unit-III, SQA- 6)**
8.	Write a note on type conversion in java.	**(Unit-I, SQA- 1)**

Part - B (2 × 20 = 40 Marks)
Note: Answer any TWO questions.

9.	Compare and contrast method overloading and method overriding in java by giving an example for each.	**(Unit-II, Q.No. 5)**
10.	What is inheritance? Explain types of inheritance by giving an example for each.	**(Unit-II, Q.No. 19, 20, 21, 22, 23, 24)**
11.	Define thread. List different ways of creating threads explain them with example.	**(Unit-III, Q.No. 11)**
12.	What is the purpose of java to package? Explain any two different ways of input output methods with example.	**(Unit-IV, Q.No. 15)**
13.	What is the purpose of AWT? Explain the controls supported by AWT by giving an example.	**(Unit-IV, Q.No. 11, 12)**
14.	Explain the different steps for database handling using JDBC.	**(Out of Syllabus)**

FACULTY OF SCIENCE

B.Sc V-Semester (CBCS) Examination

October/November - 2020

COMPUTER SCIENCE (PROGRAMMING IN JAVA)

Paper - V

Time : 2 Hours] [Max. Marks : 60

PART - A - (4 × 5 = 20 Marks)

Note : Answer any four questions.

		ANSWERS
1.	Define class. Write an example for class in Java	**(Unit - I, Q.No. 14)**
2.	Briefly explain about abstract class.	**(Unit - II, SQA-3)**
3.	Write briefly about exception handling.	**(Unit - III, SQA-10)**
4.	Write short notes on output streams.	**(Unit - III, Q.No. 22)**
5.	Write about any three types of events.	**(Unit - IV, Q.No. 9)**
6.	Write briefly about types of JDBC drivers.	**(Out of Syllabus)**
7.	Write short note on one dimensional array.	**(Unit - II, Q.No. 13)**
8.	Write briefly about synchronization.	**(Unit - III, SQA-8)**

PART - B (2 × 20 = 40 Marks)

Note : Answer any two questions

9.	Explain in detail the various date types in Java.	**(Unit - I, Q.No. 7)**
10.	Explain the various types of inheritance with examples.	**(Unit - II, Q.No. 20, 21, 22, 23, 24, 25)**
11.	Explain in detail about packages. Write a program to implement user defined package.	**(Unit - II, Q.No. 33, 34)**
12.	Explain with an example Buffered Input stream and Buffered Output stream.	**(Unit - III, Q.No. 21, 22)**
13.	Define Applet. Explain life cycle of an applet with an example program.	**(Unit - IV, Q.No. 1, 2)**
14.	Explain JDBC in detail. Write the steps to establish JDBC-ODBC connection.	**(Out of Syllabus)**

FACULTY OF SCIENCE

B.Sc V-Semester (CBCS) Examination

November / December - 2019

COMPUTER SCIENCE (PROGRAMMING IN JAVA)

Paper - V

Time : 3 Hrs] [Max. Marks : 60

PART - A (5 × 3 = 15 Marks)

Note : Answer any FIVE of the following questions

Answers

1. Write about different features of Java. **(Unit-I, SQA- 5)**
2. Write short notes on Interfaces. **(Unit-II, SQA- 4)**
3. Write briefly on multi threading. **(Unit-III, SQA- 3)**
4. What is a package? Write few inbuilt packages of Java. **(Unit-II, SQA- 6)**
5. Write short notes on Java Applet. **(Unit-IV, SQA- 3)**
6. Write short notes on Dialog Box. **(Unit-IV, SQA- 4)**
7. Write the structure of a Java program. **(Unit-I, SQA- 3)**
8. Write any three differences between Swing and AWT. **(Unit-IV, SQA- 6)**

Part – B (3 ×15 = 45 Marks)

Note: Answer ALL the following questions.

9. (a) Explain the various types of control statements in Java with an example for each. **(Unit-I, Q.No. 11, 12, 13)**

OR

(b) Explain different types of method overloading with examples. **(Unit-II, Q.No. 4)**

10. (a) Explain wrapper classes in detail. Write a Java program to demonstrate wrapper class in Java. **(Unit-II, Q.No. 36)**

OR

(b) Explain in detail about File Input Stream and File Output Stream class with an example. **(Unit-IV, Q.No. 17, 18)**

11. (a) Explain about various AWT classes used in Java. Write a program using checkboxes. **(Unit-IV, Q.No. 12, 16)**

OR

(b) Define ResultSet. Explain in detail about various ResultSet objects in Java. **(Out of Syllabus)**

FACULTY OF SCIENCE

B.Sc V-Semester (CBCS) Examination

June / July - 2019

COMPUTER SCIENCE (PROGRAMMING IN JAVA)

Paper - V

Time : 3 Hrs] [Max. Marks : 60

PART - A (5 × 3 = 15 Marks)
Note : Answer any FIVE of the following questions

		Answers
1.	What is the major difference between an interface and a class?	**(Unit-II, SQA- 7)**
2.	When do we declare a method or class abstract?	**(Unit-II, SQA- 8)**
3.	How do we set priorities for threads?	**(Unit-III, SQA- 7)**
4.	Define package. Write the syntax to create and import a package.	**(Unit-II, SQA- 6)**
5.	Explain JTable in swings with an example program.	**(Unit-IV, SQA- 7)**
6.	What are the types of JDBC drivers?	**(Out of Syllabus)**
7.	Define thread. How do we start a thread?	**(Unit-III, SQA- 6)**
8.	Write about File Input Stream and File Output Stream class.	**(Unit-III, SQA- 4, 5)**

Part – B (3 × 15 = 45 Marks)
Note: Answer ALL the questions.

9. (a) Define class. Explain about class declaration, creating objects, methods declaration and invocation with syntax and an example. **(Unit-I, Q.No. 14, 15)**

OR

(b) Define inheritance. Explain the different types of inheritance. Write a program to demonstrate multiple inheritance in java. **(Unit-II, Q.No. 19, 20, 21, 22, 23, 24)**

10. (a) Define package. How do we add a class to package? Discuss the various levels of access protection available with an example. **(Unit-II, Q.No. 33, 34, 35)**

OR

(b) What is a random access file? Why do we need a random access file? Write a program for reading/writing using random access file. **(Unit-III, Q.No. 23)**

11. (a) Explain about JFrame, JApplet and JPanel by giving an example for each. **(Unit-III, Q.No. 23, 24, 25)**

OR

(b) Explain the basic steps in developing a JDBC application. **(Out of Syllabus)**

FACULTY OF SCIENCE

B.Sc V-Semester (CBCS) Examination

November / December - 2018

COMPUTER SCIENCE (PROGRAMMING IN JAVA)

Paper - V

Time : 3 Hrs] [Max. Marks : 60

PART - A (5 × 3 = 15 Marks)
Note : Answer any FIVE of the following questions

		Answers
1.	Write about constructors.	**(Unit-II, SQA-1)**
2.	Write a note on type conversion in java.	**(Unit-I, SQA-1)**
3.	Define thread. List the methods in thread class.	**(Unit-III, SQA-6)**
4.	What are wrapper classes?	**(Unit-II, SQA-9)**
5.	Why layouts are needed?	**(Unit-IV, SQA-1)**
6.	What is the purpose of AWT? List the controls supported by AWT.	**(Unit-IV, SQA-5)**
7.	How java supports multiple inheritance?	**(Unit-II, SQA-10)**
8.	Write about thread synchronization.	**(Unit-III, SQA-8)**

Part – B (3 ×15 = 45 Marks)
Note: Answer ALL the following questions.

9. (a) Explain about branching mechanisms in java by giving an example. **(Unit-I, Q.No. 13)**

OR

(b) Compare and contrast method overloading and method overriding in java by giving an example for each. **(Unit-II, Q.No. 5)**

10. (a) What is an exception? Explain the handling exception in Java. **(Unit-III, Q.No. 1, 3)**

OR

(b) Define thread. Explain the different ways we can create thread by giving an example for each. **(Unit-III, Q.No. 11)**

11. (a) Explain the sequence of applet's life cycle methods in which they are called with an example program. **(Unit-IV, Q.No. 2)**

OR

(b) What is layout manager? Explain types of layout managers with an example. **(Unit-IV, Q.No. 18, 19)**

MAHATMA GANDHI UNIVERSITY FACULTY OF SCIENCE

B.Sc (CBCS) III-Year (V-Semester) Backlog Examination

May / June - 2019

PROGRAMMING IN JAVA

Paper - V

Time : 2½ Hrs] [Max. Marks : 60

PART - A (5 × 3 = 15 Marks)
Answer all the questions

Answers

1. Explain Command-line arguments in Java. **(Unit-II, SQA- 11)**
2. What is Synchronization? **(Unit-III, SQA- 8)**
3. Explain the differences between Swing and AWT. **(Unit-IV, SQA- 6)**

Part - B (3 × 15 = 45 Marks)
Note: Answer ALL the questions.

4. (a) Explain the Conditional Statements in Java with examples. **(Unit-I, Q.No. 11)**

OR

   (b) Differentiate the use of Abstract Classes and Interfaces. **(Unit-II, Q.No. 32)**

5. (a) Define Exception. Explain the Handling of Exceptions in Java. **(Unit-III, Q.No. 1, 3)**

OR

   (b) What is a Thread? Explain the implementation of Thread with Runnable Interface. **(Unit-III, Q.No. 11)**

6. (a) Explain the Event Handling mechanism in Java. **(Unit-IV, Q.No. 8)**

OR

   (b) Explain different Layout Managers in Java with examples. **(Unit-IV, Q.No. 19)**

MAHATMA GANDHI UNIVERSITY
FACULTY OF SCIENCE

B.Sc (CBCS) III-Year V-Semester Regular Examinations

November / December - 2018

PROGRAMMING IN JAVA

Paper - V

Time : 2½ Hrs] [Max. Marks : 60

PART - A (5 × 3 = 15 Marks)
Answer all the questions

		Answers
1.	Explain Parameterized Constructors with examples.	**(Unit-II, SQA- 12)**
2.	Explain the usage of StringBuffer Class.	**(Unit-II, SQA- 13)**
3.	Explain the differences between Swing and AWT.	**(Unit-IV, SQA- 6)**

Part - B (3 × 15 = 45 Marks)
Answer ALL the questions.

4. (a) Explain the Loops in Java with examples. **(Unit-I, Q.No. 12)**

OR

(b) What is Inheritance? Explain different types of Inheritance with examples. **(Unit-II, Q.No. 19, 20, 21, 22, 23, 24)**

5. (a) Define Exception. Explain the Handling of Exceptions in Java. **(Unit-III, Q.No. 1, 3)**

OR

(b) What is Thread? Explain the implementation of Thread with Runnable Interface. **(Unit-III, Q.No. 11)**

6. (a) Write a program in Java to implement Radio buttons and Container class. **(Unit-IV, Q.No. 16, 13)**

OR

(b) Explain different Layout Managers in Java with examples. **(Unit-IV, Q.No. 19)**