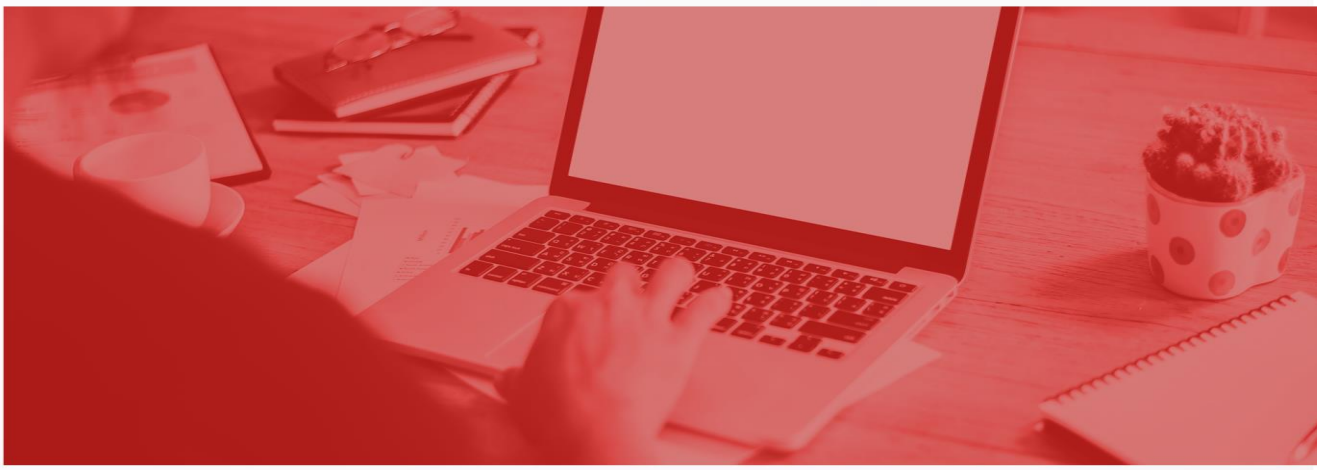
PROGRAMMING IN Java



Compiled by: Er. Ankit Kharel

Nepal College of Information Technology

Balkumari, Lalitpur

CHAPTER: 1

ELEMENTS OF JAVA LANGUAGE

Java, **programming language** and a **platform**, was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

TYPES OF JAVA APPLICATIONS

There are mainly 4 types of applications that can be created using Java programming:

1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

2) Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

JAVA EDITIONS

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, Strings, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

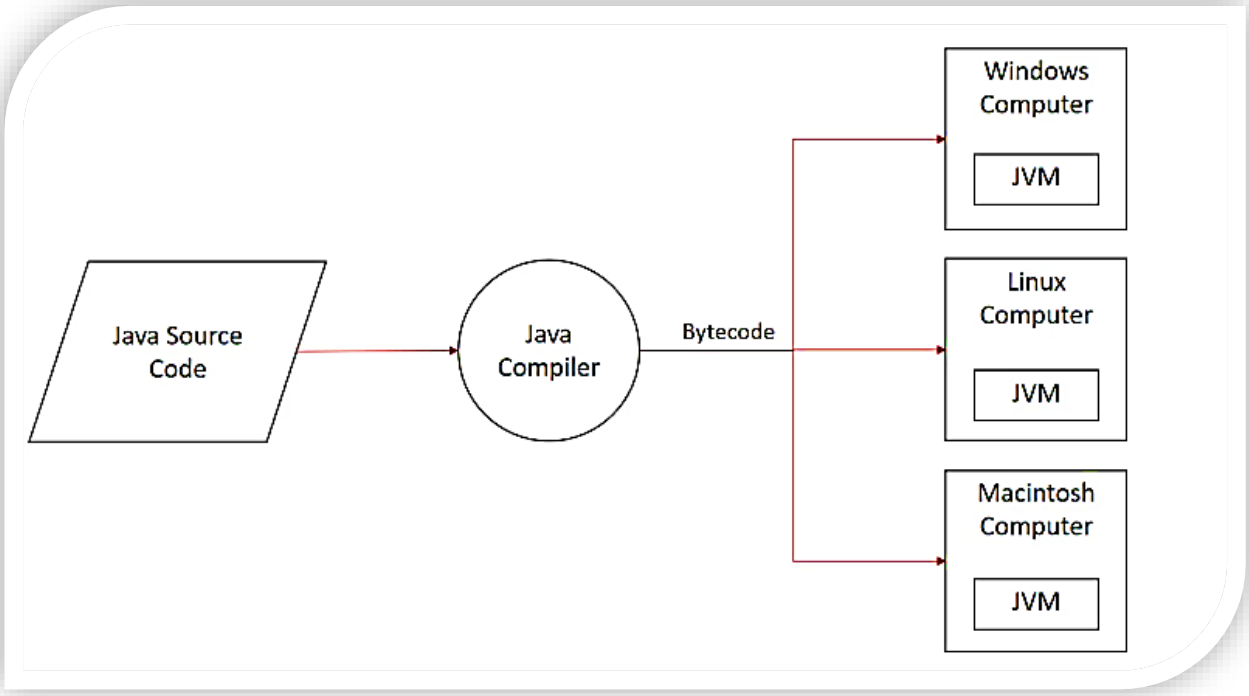
2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA etc.

3) Java ME (Java Micro Edition)

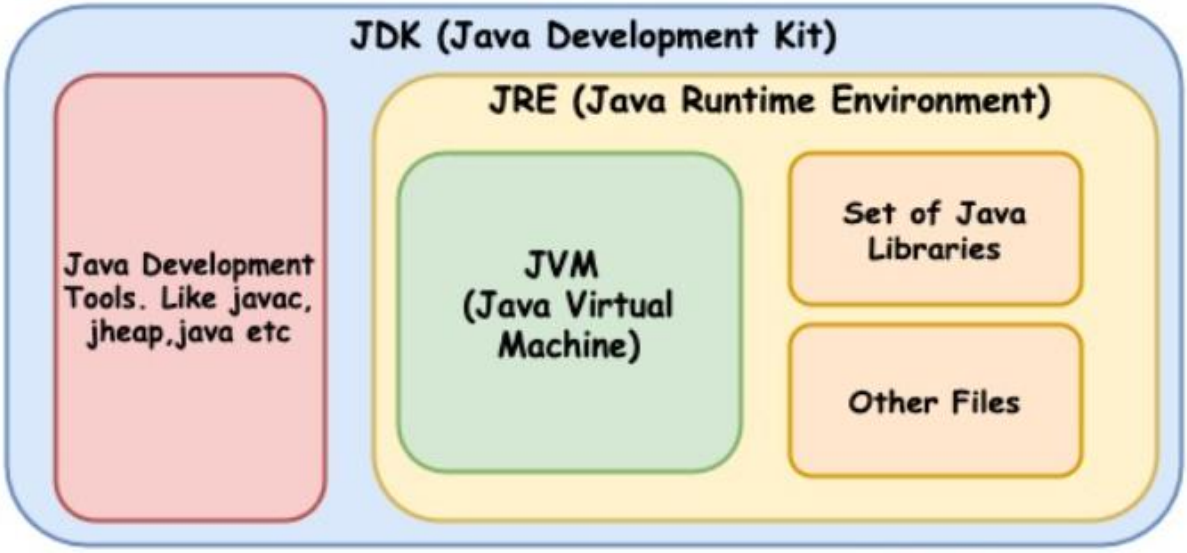
It is a micro platform that is dedicated to mobile applications.

JAVA PROGRAM COMPILATION PROCESS



The source code of a Java code is compiled into an intermediate binary code known as the **Bytecode** during the Java compilation process. The machine cannot directly execute this Bytecode. A virtual machine known as the **Java Virtual Machine**, or JVM, understands it. JVM includes a Java interpreter that converts Bytecode to target computer machine code. JVM is platform-specific, which means that each platform has its own JVM. However, once the proper JVM is installed on the machine, any Java Bytecode code can be run.

JDK, JRE & JVM



JDK

JDK stands for "Java Development Kit." It is a set of software tools and libraries that developers use to develop, compile, and run Java applications and applets. The JDK is essential for Java developers as it contains everything needed to write, compile, test, and run Java programs.

The JDK contains a private Java Runtime Environment (JRE) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JRE

It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime. **JRE = JVM + Library Files.**

CLASS

A Class is a group of similar objects and describes both characteristics (data members) and behavior (member functions) of the object. Classes are user defined data types that binds together data types and function.

Syntax:

```
<access specifier> class class_name
{
        // member variables
        // class methods
}
//A class has either public or default access specifier
```

OBJECT

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory.

Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Syntax:

```
ClassName object = new ClassName();
```

When we create an instance of the class by using the new keyword, it allocates memory (heap) for the newly created **object** and also returns the **reference** of that object to that memory.

Hellow World Program

```
public class Hellow {

    public static void main(String[] args) {

        System.out.println("Hellow World");

    }

}
```

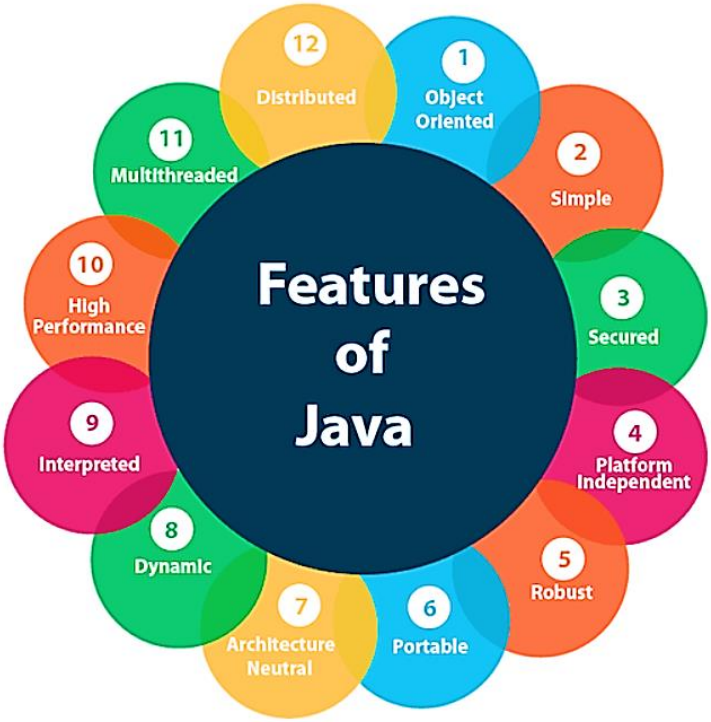
Here's an explanation of each part of this code:

- **public class HelloWorld** : This line declares a class named **HelloWorld**. In Java, every application begins with a class definition, and the class name must match the filename (excluding the **.java** extension) where it is defined. In this case, the filename should be **HelloWorld.java**.
- **public static void main(String[] args)** : This line declares the **main** method. The **main** method is the entry point of a Java application. It is a special method that is executed when you run the program. Here's a breakdown of this line:
    - **public**: This is an access modifier that makes the **main** method accessible from outside the class.
    - **static**: This keyword means that the **main** method belongs to the class itself and can be called without creating an instance of the class. The main method in Java is always static beacus of following reasons:
        - main() method should be declared static so that the JVM (Java Virtual Machine) can call it without having to create an instance of the class containing the main() method.
        - If the main method is not declared static, the JVM has to create an instance of the main Class, and because the constructor might be overloaded and include arguments, there will be no reliable and consistent way for the JVM to find the main method in Java.

- **void**: This specifies that the **main** method doesn't return any value.
- **main**: This is the name of the method.
- **String[] args**: This is the parameter list for the **main** method. It accepts an array of strings called **args**. These are command-line arguments that can be passed to the program when it's run.

- **System.out.println("Hello, World!"):** This line is the actual code that gets executed when the program is run. Here's what it does:
  - **System**: This is a built-in class in Java that provides access to system-related functionality.
  - **out**: This is a static field of the **System** class that represents the standard output stream.
  - **println**: This is a method of the **PrintStream** class associated with the standard output stream. It is used to print text to the console followed by a newline character.
  - **"Hello, World!"**: This is a string literal, enclosed in double quotes, that represents the text you want to print.

FEATURES OF JAVA



1. Simpe

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

2. Object Oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behavior.

Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.

Basic concepts of OOPs are:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

3. Platform Independent

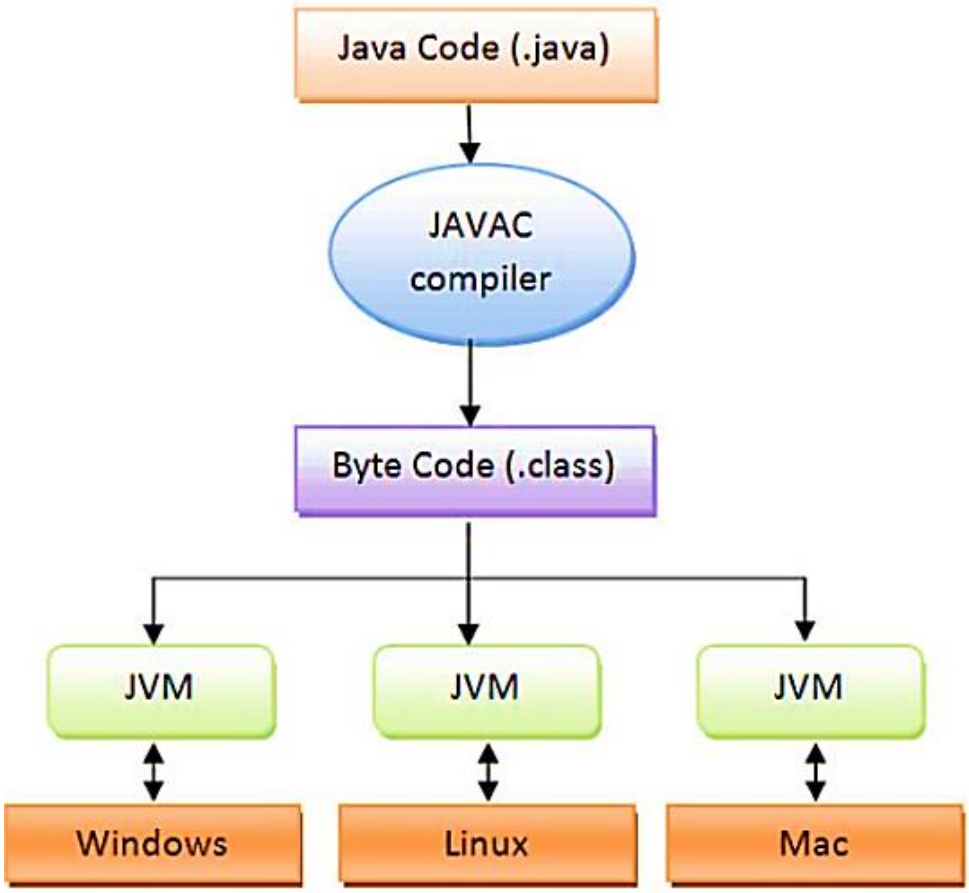
Q.Describe how java is a platform nutral language by a diagram.

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

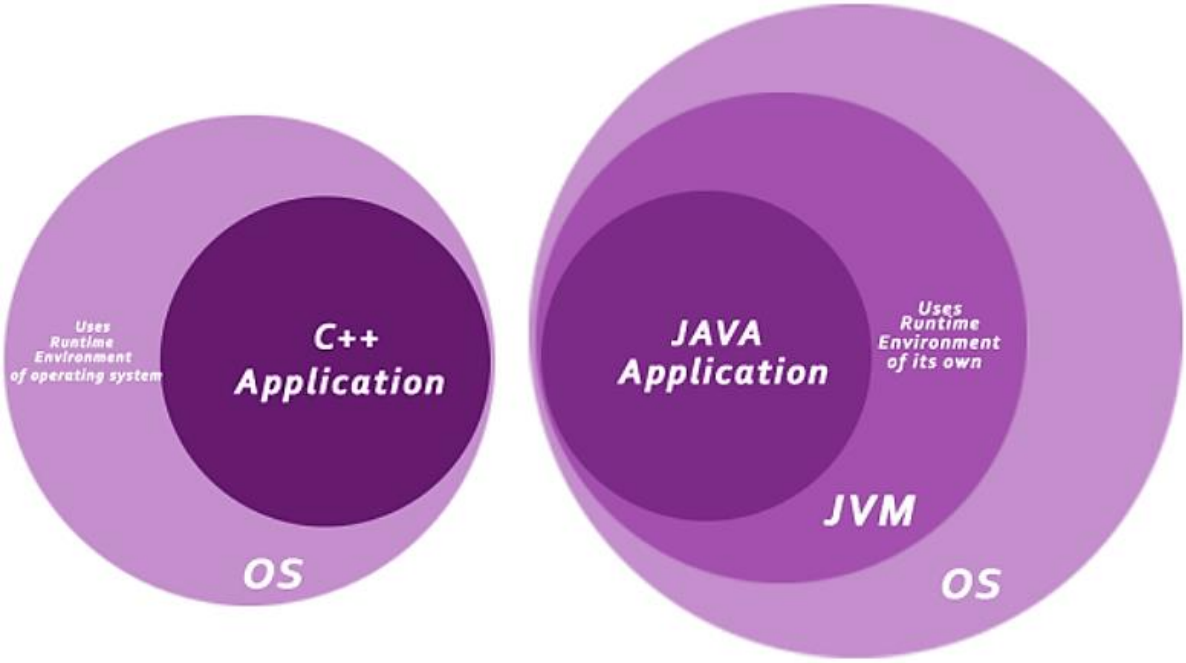


4. Secured

Q. How java is more secure than other programming language?

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox.



- **Classloader:** Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.

- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access rights to objects.

- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

5. Robust

Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

6. Architecture-neutral & Portable

Q.Explain why java program are known as portable and architecture neutral.

Java is architecture-neutral and portable because of following reasons:

- When we compile a Java source code file, it is not compiled directly to machine code for a specific platform. Instead, it is compiled into bytecode, which is a platform-independent, low-level representation of the code. This bytecode is not tied to any particular hardware or operating system.
- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.
- Java offers standardized APIs (Application Programming Interfaces) for various tasks, such as database access, GUI development (Swing and JavaFX), and web development (Servlets and JSP). These APIs provide a common interface to various services and are implemented consistently across different Java platforms.
- Before executing bytecode, the JVM performs bytecode verification to ensure that it adheres to Java's safety and security rules. This verification step helps ensure that Java programs do not violate the platform's architecture.
- Java's garbage collection mechanism automatically manages memory, reducing the risk of memory-related issues that can vary between platforms.

Due to these features and design principles, Java programs can be written once and run on any platform with a compatible JVM, making them portable and architecturally neutral.

7. High Performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

8. Multi Threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

9. Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

10. Dynamic

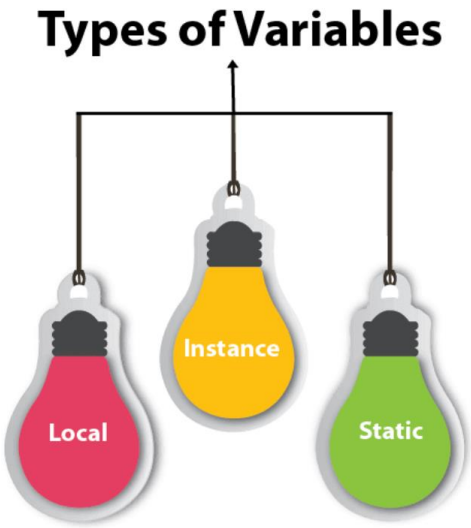
Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

JAVA VARIABLES

Variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

Variable is a name of memory location. There are three types of variables in java: local, instance and static.



1. Local Variable

A variable declared inside the body of the method is called local variable. We can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

2. Instance Variable

Instance variables are declared within a class but outside of any method, constructor, or block. It is called an instance variable because its value is instance-specific and is not shared among instances. Each instance of the class has its own set of instance variables, which are not shared among instances.

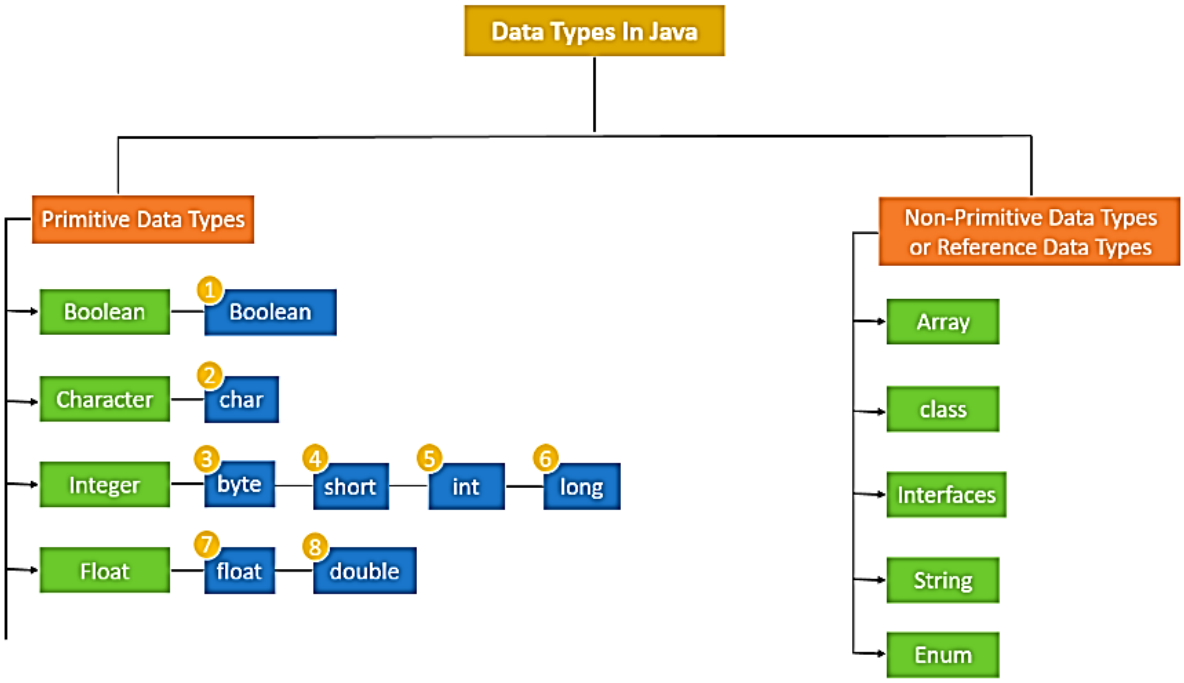
3. Static Variable

Static variables are declared within a class, like instance variables, but they are marked with the **static** keyword. They are associated with the class itself rather than with individual instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

```
public class A
{
     int ins_var = 50;//instance variable
   static int static_var = 100;//static variable
     void method()
     {
        int local_var= 90;//local variable
     System.out.println("Value of local_var:"+ local_var);
     }
     public static void main(String args[])
     {
      int data=230; //local variable
System.out.println("Value of static_var: "+ A.static_var);
      System.out.println("Value of data: "+ data);
      A a = new A();
      System.out.println("Value of ins_var:"+ a.ins_var);
      a.method();

     }
}
```

```
Value of static_var: 100
Value of data: 230
Value of ins_var:50
Value of local_var:90
```

JAVA DATA TYPES



Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Q. WAP to add two integer numbers.

```
import java.util.Scanner;

public class Sample {
    public static void main(String[] args) {
// Create a Scanner object to read input from the user
  Scanner sc = new Scanner(System.in);

// Prompt the user to enter the first number
   System.out.print("Enter the first number: ");
        int num1 = sc.nextInt();

 // Prompt the user to enter the second number
  System.out.print("Enter the second number: ");
        int num2 = sc.nextInt();

 // Calculate the sum of the two numbers
        int sum = num1 + num2;

  // Print the result
System.out.println("The sum of " + num1 + " and " + num2 +
" is: " + sum);

    }
}
```

```
Enter the first number: 3
Enter the second number: 4
The sum of 3 and 4 is: 7
```

*The line **Scanner sc = new Scanner(System.in);** in Java is used to create a new instance of the **Scanner** class, which is a built-in class in Java that allows us to read input from various sources, including the keyboard (in this case, **System.in**). T o use the **Scanner** we need to import the Scanner class from the util package as: **import java.util.Scanner***

*Here's a breakdown of how you can use the **Scanner** object:*

- ***sc.nextInt()**: Reads an integer from the user.*
- ***sc.nextFloat()**: Reads a floating-point number from the user.*
- ***sc.nextBoolean()**: Reads a Boolean value from the user.*
- ***sc.nextDouble()**: Reads a Double value from the user.*
- ***sc.nextByte()**: Reads a Byte value from the user.*
- ***sc.next()**: Reads a string (a sequence of characters without spaces) from the user.*
- ***scr.nextLine()**: Reads a line of text (including spaces) from the user.*

STRINGS

In java's String, is not a simple type. Nor is it simply an array of characters. Rather, String is defined as an object which can be created using two ways:

1. Using **String Literal**.
2. Using **new keyword**.

1. String Literal and String Constant Pool

A literal, in computer science, is a notation used for representing a value. Java String literal can be created and represented using the double-quotes. All of the content/characters can be added in between the double quotes.

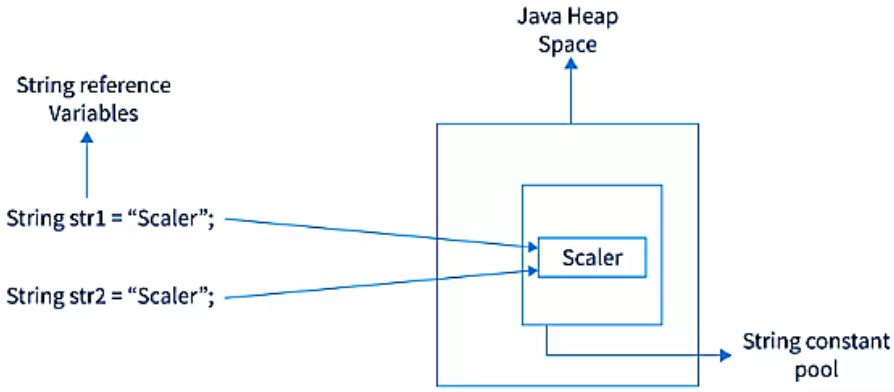
Example:

```
public class Sample {

  public static void main(String args[]) {

    String str = "Hellow World";
  }
}
```

The Strings in Java are stored in a special place in heap called "String Constant Pool" or "String Pool".

String Constant Pool: The string constant pool in Java is a storage area in the heap memory that stores string literals. When a string is created, the JVM checks if the same value exists in the string pool. If it does, the reference to that existing object is returned. Otherwise, a new string object is created and added to the string pool, and its reference is returned.

```
String str1 = "Scaler";

// New String is not created.
// str2 is pointing to the old string value only.
String str2 = "Scaler";
```



2. By new Keyword

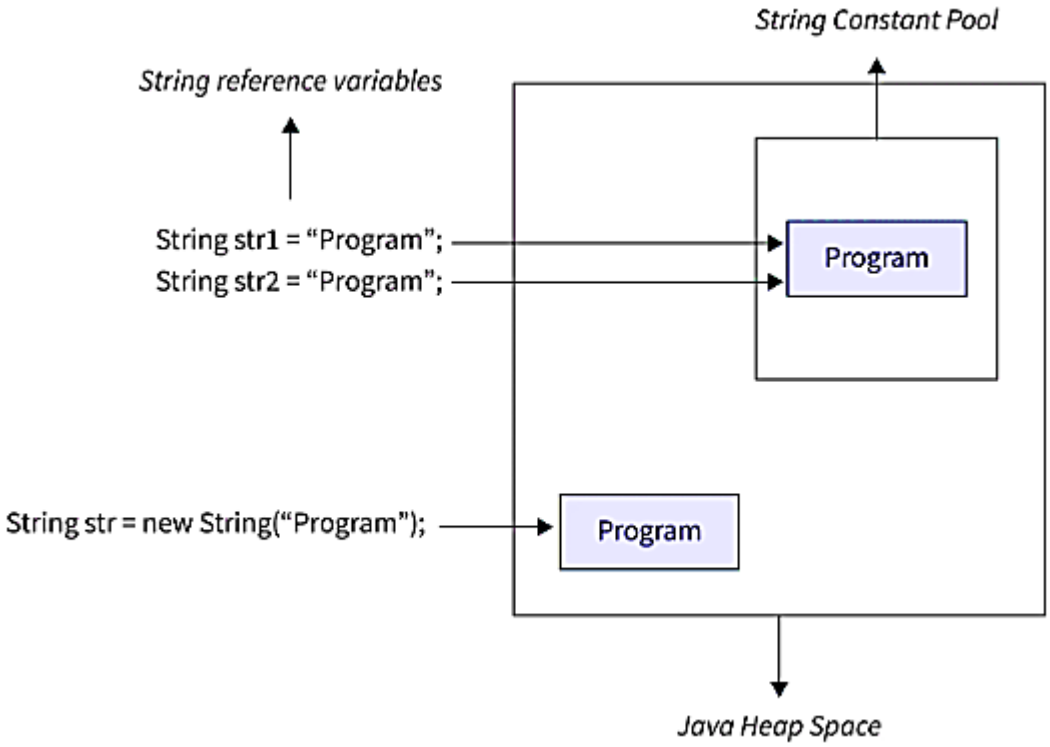
When a string is created with **new**, a new object of the String class is created in the heap memory, outside the string constant pool. Unlike string literals, these objects are allocated separate memory space in the heap, regardless of whether the same value already exists in the heap or not.

Syntax:

```
String stringName = new String("string_value");
```

Example: Creating Java Strings using the new keyword

```
String str = new String("Program");
```



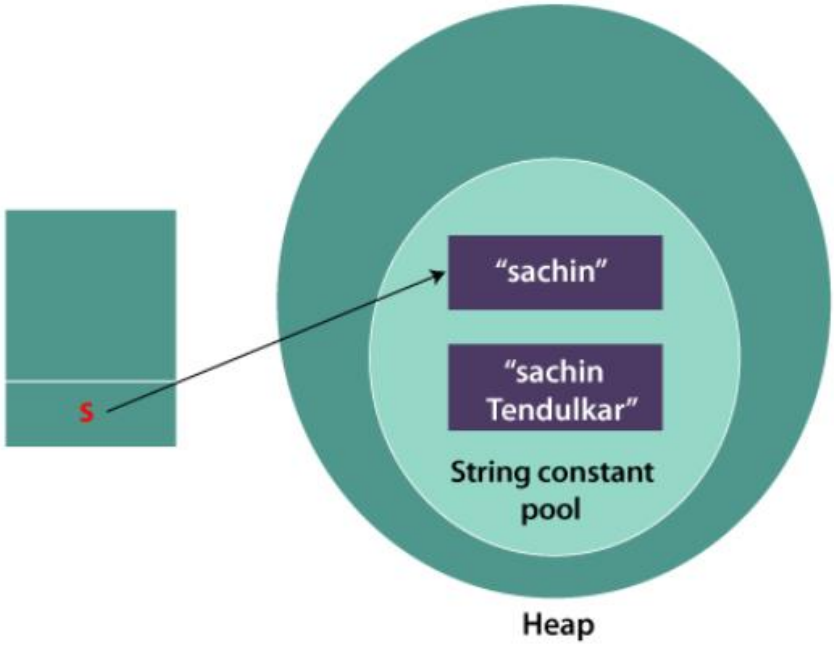
Immutable String in Java

In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a new String object is created.

Let's try to understand the concept of immutability by the example given below:

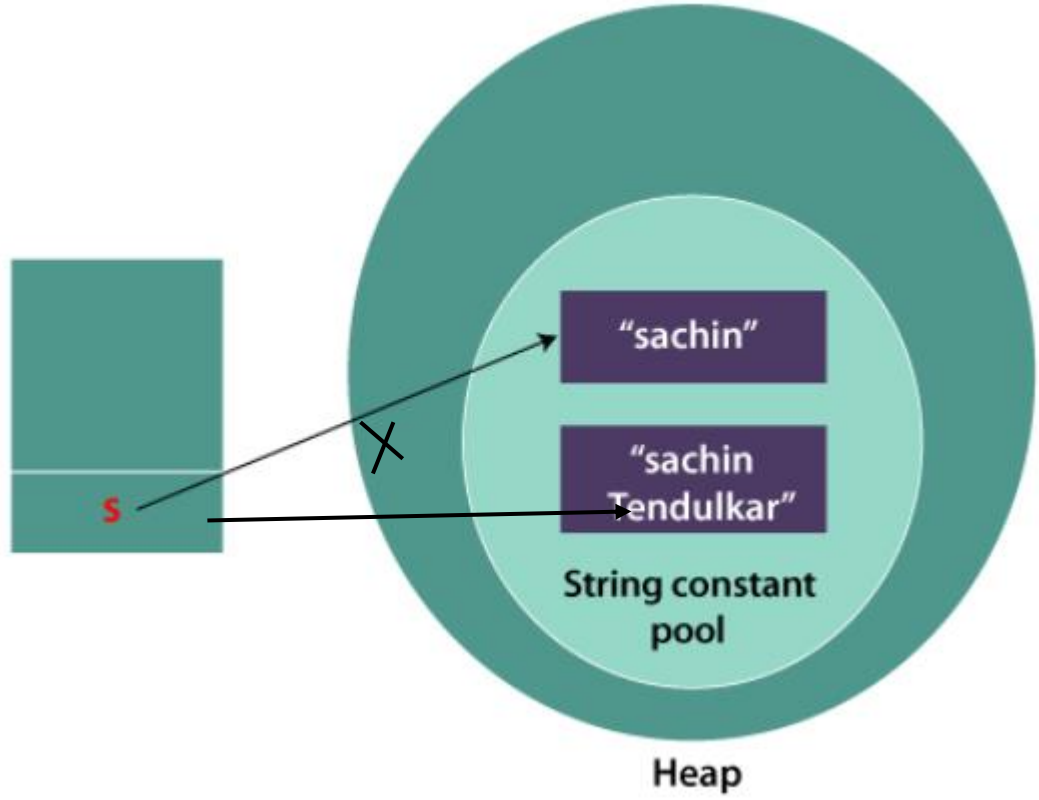
```
class Sample{
 public static void main(String args[]){
   String s="Sachin";
   s.concat(" Tendulkar");//concat() method appends
the string at the end
   System.out.println(s);//will print Sachin
because strings are immutable objects
 }
}
```



Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as immutable.

But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

```
class Sample{
 public static void main(String args[]){
   String s="Sachin";
   s=s.concat(" Tendulkar");
   System.out.println(s); );//will print Sachin
Tendulkar. In such a case, s points to the "Sachin
Tendulkar". Please notice that still Sachin object
is not modified.
 }
}
```



ARRAYS

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.

Syntax(One Dimensional Array):

```
datatype[] arrayName = new datatype[length];
            OR
datatype arrayName[] = new datatype[length];

//[length] = Length of the array
```

Example:

```
// Both will create an int array with a length of 5.

int[] myArray = new int[5];
int myArray1[] = new int[5];
```

Q. WAP to input and output 5 array elements

```
import java.util.Scanner;
public class Arrays{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = new int[5];

        // Input 5 elements into the array
        for (int i = 0; i < 5; i++) {
          System.out.println("Enter arr["+i+"] element:");
           arr[i] = sc.nextInt();
        }
        // Output the 5 elements from the array
        System.out.println("Array elements:");
        for (int i = 0; i < 5; i++) {
            System.out.print(arr[i]+ "  ");
        }
    }
}
```

```
Enter arr[0] element:
6
Enter arr[1] element:
7
Enter arr[2] element:
8
Enter arr[3] element:
9
Enter arr[4] element:
11
Array elements:
6  7  8  9  11
```

For each Loop for Java Array

A for-each loop in Java is also known as an enhanced for loop. It's a simplified way to iterate over elements in an array

Syntax:

```
for (elementType element : iterableObject) {
    // code to be executed for each element
}
```

Q. WAP to input and output 5 array elements using for each loop

```
import java.util.Scanner;
public class Arrays{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = new int[5];

        // Input 5 elements into the array
        for (int i = 0; i < 5; i++) {
          System.out.println("Enter arr["+i+"] element:");
           arr[i] = sc.nextInt();
        }
        // Output the 5 elements from the array
        System.out.println("Array elements:");
        for (int i: arr) {
            System.out.print(i+ "  ");
        }
    }
}
```

Q. WAP to input 10 array elements and put even and odd elements in separate array.

```
import java.util.Scanner;
public class ArrayProgram {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = new int[10];
        int[] even = new int[10];
        int[] odd = new int[10];
        //input array
        for(int i=0; i<10;i++) {
         System.out.print("Enter a["+i+"] element:");
         arr[i]= sc.nextInt();
        }
        //copy elements into even and odd array
        int ec=0, oc=0;
        for (int i = 0; i < 10; i++) {
              if(arr[i]%2==0) {
                   even[ec]=arr[i];
                   ec++;
              }else {
                   odd[oc]=arr[i];
                   oc++;
              }
         }
        //print even array elements
        System.out.println("The Even elemnts are:");
        for (int i = 0; i < ec; i++) {
              System.out.print(even[i]+ "  ");
         }
```

```
        //print odd array elements
        System.out.println("\nThe odd elements are:");
        for (int i = 0; i < oc; i++) {
              System.out.print(odd[i]+ "  ");
         }

    }
}
```

```
Enter a[0] element:1
Enter a[1] element:2
Enter a[2] element:3
Enter a[3] element:4
Enter a[4] element:5
Enter a[5] element:6
Enter a[6] element:7
Enter a[7] element:8
Enter a[8] element:9
Enter a[9] element:10
The Even elemnts are:
2  4  6  8  10
The odd elements are:
1  3  5  7  9
```

Q. WAP to input 10 array elements and delete x^{th} position element from the array.

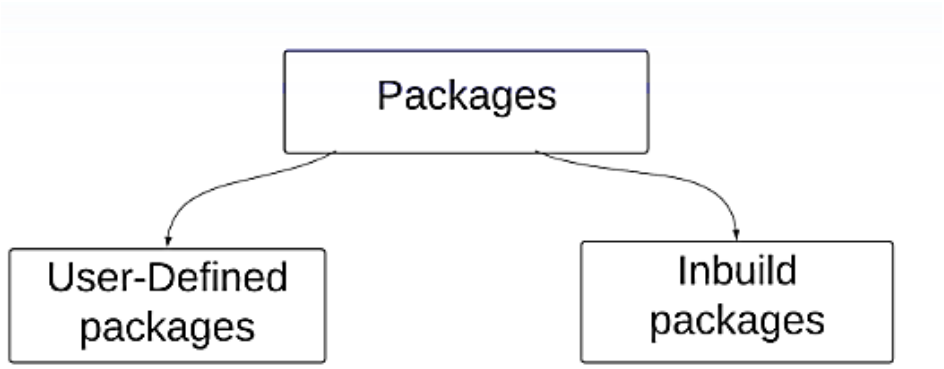
```
import java.util.Scanner;
public class ArrayProgram {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] arr = new int[10];
        //input array
        for(int i=0; i<10;i++) {
         System.out.print("Enter a["+i+"] element:");
         arr[i]= sc.nextInt();
        }
        //get x value from user
        int pos;
       System.out.println("Enter postion to delete(0-9):");
       pos = sc.nextInt();
       //Delete the element at the specified position
        for (int i = pos; i < 9; i++) {
              arr[i] = arr[i+1];
         }
        // Set the last element to 0
        arr[9]=0;
        //print updated array
        System.out.println("The Updated array is:");
        for (int i = 0; i < 10; i++) {
              System.out.print(arr[i]+ "  ");
         }
    }
}
```

```
Enter a[0] element:1
Enter a[1] element:2
Enter a[2] element:3
Enter a[3] element:4
Enter a[4] element:5
Enter a[5] element:6
Enter a[6] element:7
Enter a[7] element:8
Enter a[8] element:9
Enter a[9] element:10
Enter postion to delete(0-9):
5
The Updated array is:
1  2  3  4  5  7  8  9  10  0
```

PACKAGES IN JAVA

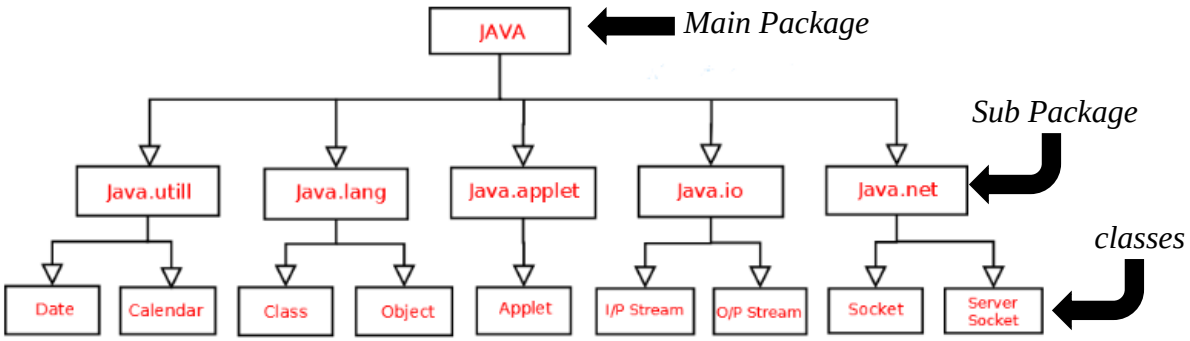
In Java, a package is a way to organize and group related classes and interfaces. Packages help in organizing and structuring code, preventing naming conflicts, and making it easier to manage and maintain large-scale Java applications.

TYPES OF PACKAGES



1. Built-In packages

Several of the frequently used built-in packages include:



a) java.lang: Provides classes for languages in use (e.g classed which defines primitive data types, math operations). This package is imported automatically.

b) java.io: This package contains classes that handle input and output operations.

c) java.util: Contains utility classes which provide data structures like Linked List, Dictionary and support ; for Date / Time operations.

d) java.applet: This package includes classes for building Applets.

e) java.net: This library includes classes that support networking operations.

Syntax for importing packages in java

- *Importing specific Class from package*

```
import packageName.ClassName;
import java.util.Scanner;
```

- *Importing all Class from package*

```
import packageName.*

import java.util.*;
```

PACKAGE NAMING CONVENTION

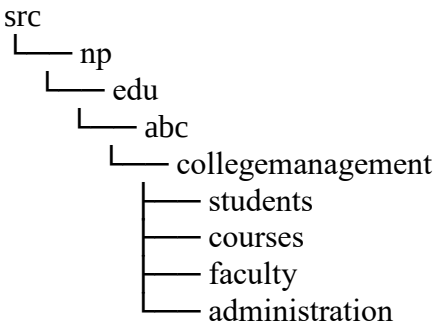
Package names are arranged in the domain names' reverse order.

Let's adapt the package naming convention to our college's domain name, "abc.edu.np" for a Java application related to college management. Here's how you might structure our packages:

Base Package Name (Reversed Domain Name): np.edu.abc.collegemanagement

Subpackages: Organize our code into subpackages based on different functional areas or modules of our college management system:

- **np.edu.abc.collegemanagement.students**: This package could contain classes related to student management.
- **np.edu. abc.collegemanagement.courses**: This package could contain classes related to course management.
- **np.edu. abc.collegemanagement.faculty**: This package could contain classes related to faculty and staff management.
- **np.edu. abc.collegemanagement.administration**: This package could contain classes related to administrative tasks.



2. User defined packages

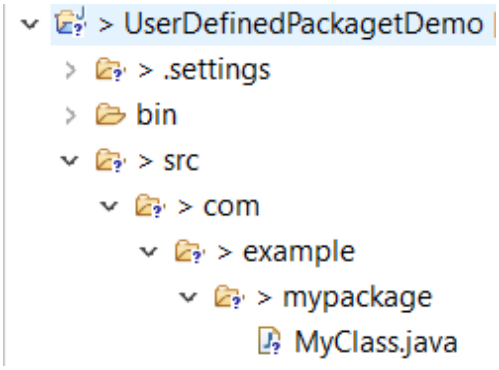
Q. What is user defined package? Illustrate the process of defining and using define package with suitable example.

Java package created by user to categorize their project's classes and interface are known as user-defined packages.

The process of defining and using the user defined packages are as following:

1: Create the Package Directory Structure

*Project Name: **UserDefinedPackageDemo***
*Package Name: **com.example.mypackage***
*Class Name inside package: **MyClass.java***



2. Use Package keyword to create a package and write the class code

```
package com.example.mypackage;
    public class MyClass {
        public void Greeting() {
            System.out.println("Hellow From MyClass");
        }
    }
```

3. Compile the Package(Through command prompt or IDE)

```
javac MyClass.java
```

This will compile the **MyClass** class and generate a **.class** file in the same directory.

4. Import and use the user defined Package

In another Java file outside the package where we want to use the user-defined package, we need to import the package at the beginning of our Java file.

```
import com.example.mypackage.MyClass;
public class Test {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        obj.Greeting();
    }
}
```

5. Run the java program

```
java Test
```

This will execute the **Test** class, which imports and uses the **MyClass** from our user-defined package.

Q. Create a class Myclass in a package Mypack. Import newly created class Myclass from IpmClass.

Step 1: Create the MyClass Class in the Mypack Package

```
package MyPack;
     public class MyClass {
          public void Greeting() {
             System.out.println("Hellow From MyClass");
          }
     }
```

Step 2: Create the IpmClass Class in a Different Package

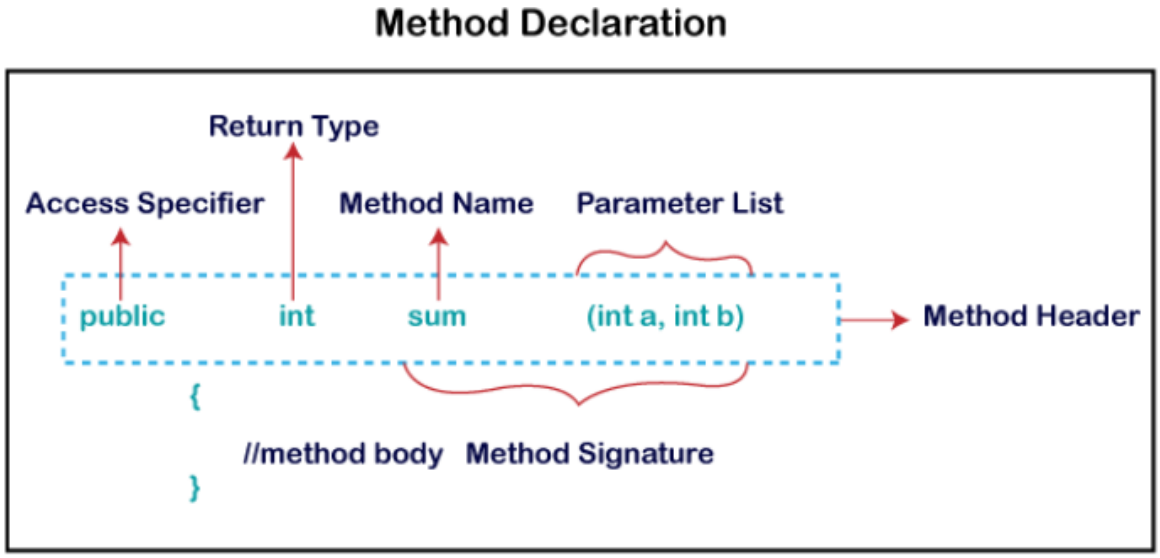
```
package anotherpackage;
import Mypack.MyClass;
public class IpmClass {
     public static void main(String[] args) {
          MyClass obj = new MyClass();
          obj.Greeting();
     }
}
```

FUNCTIONS/ METHODS

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code.

```
Here's the basic syntax for defining a method in Java:

accessModifier returnType methodName(Arguments_List) {

                    // Method body

                    // Code that defines what the method does

                    // Optionally, return a value using the return
                statement

}
```

Example:



Q. WAP to check for a odd/even number using user defined function/method.

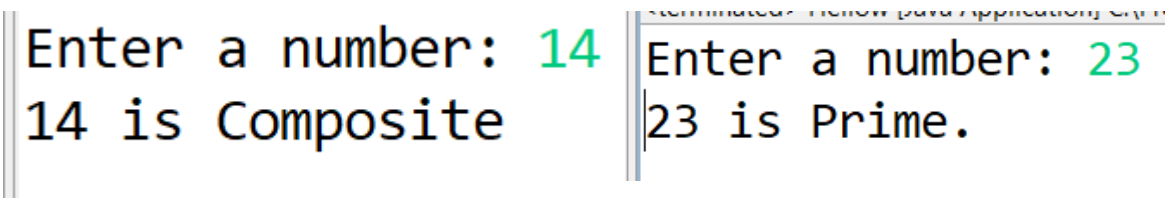
```
import java.util.Scanner;

public class Sample {
 // User-defined static method(static method can be called without
creating object of a class) to check if a number is odd
    public static boolean isOddEven(int number) {
        if(number % 2 == 0){
         return true;
        }else {
         return false;
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        if (isOddEven(num)) {
          System.out.println(num + " is an even number.");
        } else {
          System.out.println(num + " is an odd number.");
        }
    }
}
```

```
Enter a number: 5
5 is an odd number.
```

```
Enter a number: 8
8 is an even number.
```

Q. WAP to check for a Prime number using user defined function/method.

```
import java.util.Scanner;

public class Prime {

    public  void isPrime(int number) {
    int factor =0;
    for(int i=1; i<=number; i++) {
        if(number%i==0) {
            factor++;
        }
    }
    if(factor==2) {
       System.out.println(number + " is Prime.");
    }else {
      System.out.println(number + " is Composite");
    }

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        // Creating object of the class
        Prime p = new Prime();
        p.isPrime(num);
    }
}
```



Q. Write a function that takes an array of integers as an arguments and returns sum of even numbers in that array.

```
import java.util.Scanner;

public class EvenSum {

    public  int sumOfEven(int[] a) {
        int sum = 0;

        for (int i : a) {
            if (i%2 == 0) {
                sum += i;
            }
        }

        return sum;
    }

    public static void main(String[] args) {
        int n, sum;
        Scanner sc = new Scanner(System.in);
     System.out.println("Enter total number of elements:");
        n = sc.nextInt();
        int arr[] = new int[n];
        //input Array
        for (int i = 0; i < n; i++) {
              System.out.print("Enter a["+i+"] element:");
              arr[i]=sc.nextInt();
          }
```

```
        //create object of class
        EvenSum E = new EvenSum ();
        sum = E.sumOfEven(arr);
        System.out.println("Sum of Even Numbers: "+sum);
    }
}
```

```
Enter total number of elements:
5
Enter a[0] element:2
Enter a[1] element:3
Enter a[2] element:4
Enter a[3] element:5
Enter a[4] element:6
Sum of Even Numbers: 12
```

Q. WAP to print following pattern

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

```
import java.util.Scanner;
public class Pattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int n = sc.nextInt();

        for (int i = 1; i <= n; i++) {
         // Print '*' for each column in the current row
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
         // Move to the next line for the next row
            System.out.println();
        }
    }
}
```

Q. WAP to print following pattern

```
1
0 1
1 0 1
0 1 0 1
1 0 1 0 1
```

```java
import java.util.Scanner;
public class Pattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of rows: ");
        int n = sc.nextInt();

        for (int i = 1; i <= n; i++) {
         // Print '*' for each column in the current row
            for (int j = 1; j <= i; j++) {
              if((i+j)%2!=0){
                System.out.print("0 ");
              }else{
                System.out.print("1 ");
              }
            }
         // Move to the next line for the next row
            System.out.println();
        }
    }
}
```

METHOD OVERLOADING(compile time polymorphism)

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**. There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Method overloading is not possible by changing the return type of the method only.

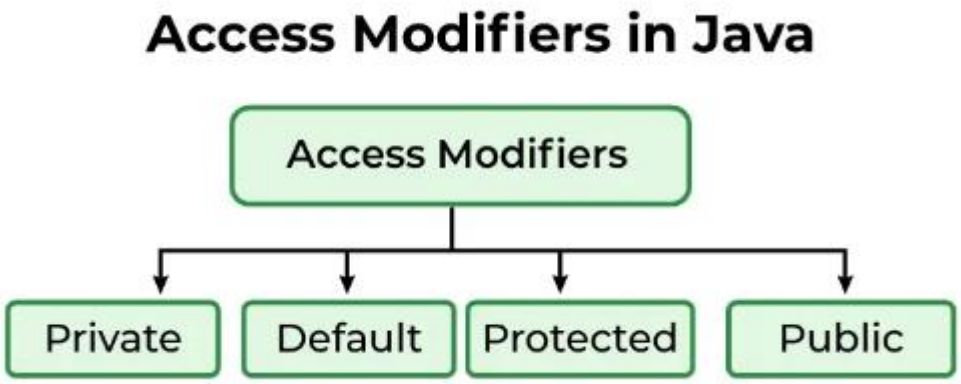
```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double b) {
        return a + b;
    }
    int add(int a, int b, int c) {
        return a + b + c;
    }
    double add(int a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
         Calculator c = new Calculator();
         int r1 = c.add(5, 6);
         System.out.println("Value of r1= "+r1);
         int r2 = c.add(5, 6, 9);
         System.out.println("Value of r1= "+r2);
         double r3 = c.add(5.3, 6.2);
         System.out.println("Value of r1= "+r3);
         double r4 = c.add(5, 6.2);
         System.out.println("Value of r1= "+r4);
     }
}
```

```
<terminated> Calculator [Java Application] C:\Program Fil
Value of r1= 11
Value of r1= 20
Value of r1= 11.5
Value of r1= 11.2
```

ACCESS MODIFIERS/SPECIFIERS IN JAVA

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of Java access modifiers:



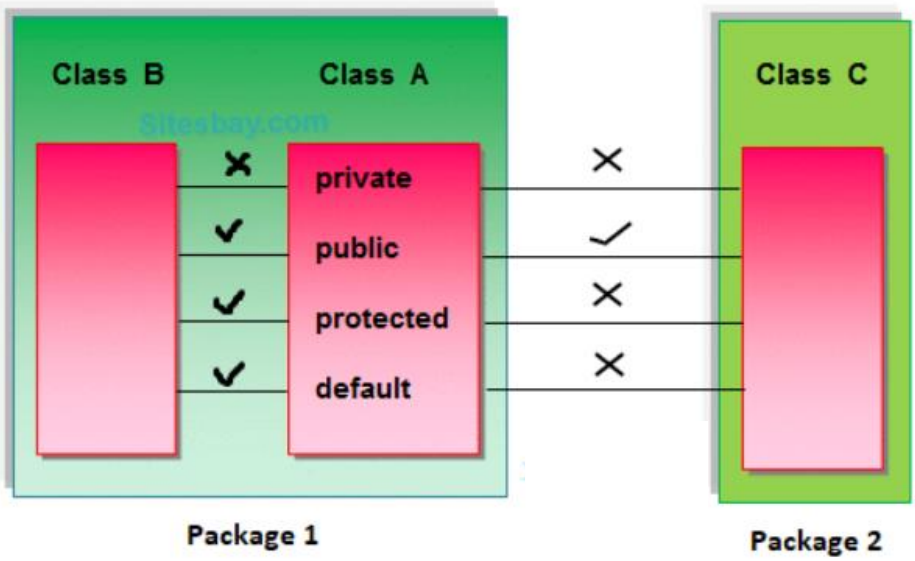
1. Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

2. Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. Protected: The access level of a protected modifier is within the package and outside the package through child class.

4. Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y



WRAPPER CLASS

Wrapper classes in Java are used to represent primitive data types as objects. In Java, we have primitive data types like **int**, **double**, **char**, and **boolean** that store simple values. These primitive data types are not objects, which means we can't perform certain operations on them, like calling methods or using them in data structures that require objects.

Wrapper classes provide a way to work with primitive data types as if they were objects. They "wrap" or "encapsulate" the primitive values within an object. Each primitive data type has a corresponding wrapper class.

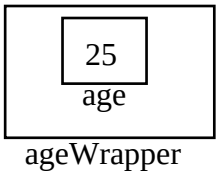
Let's say we have an **int** variable to store an integer value.

```
int age = 25;
```

age is a primitive data type, and it directly stores the integer value 25 in memory. You can perform basic operations with it, but you can't treat it like an object.

In contrast, we can use a wrapper class to create an object that holds the same value. For example, we can use the **Integer** wrapper class:

```
Integer ageWrapper = new Integer(age);
```



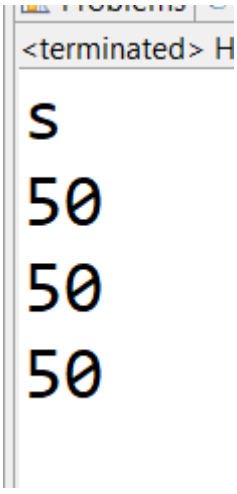
Below are the Primitive Data Types and their corresponding Wrapper classes:

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

1. Auto Boxing

Autoboxing is when the Java compiler performs the automatic conversion of the primitive data types to the object of their corresponding wrapper classes. **For example,** converting an *int* to *Integer*, a *double* to *Double*, etc.

```
//Autoboxing example of int to Integer and char to Character

public class AutoboxingExample{

public static void main(String args[]){
     char ch = 's';
    //Autoboxing- primitive to Character object conversion
     Character s = ch;

     int a=50;
     //Converting int into Integer explicitly
     Integer first=Integer.valueOf(a);
     //Autoboxing, now compiler will write
     Integer.valueOf(a) internally and hence, doesn't
     generate an error
     Integer second=a;
     System.out.println(s);
     System.out.println(a);
     System.out.println(first);
     System.out.println(second);
}
}
```



2. Unboxing

It is just the opposite process of autoboxing. Unboxing is automatically converting an object of a wrapper type (Integer, for example) to its corresponding primitive (int) value.

```
//Unboxing example of Integer to int and Character to char
public class UnboxingExample{
public static void main(String args[]){
     Character ch = 's';
     //Unboxing - Character object to primitive conversion
     char s = ch;
     Integer a = new Integer(5);
     //Converting Integer to int explicitly
     int first=a.intValue();
     //Unboxing,  now  compiler  will  write  a.intValue()
     internally
     int second=a;
    System.out.println(s);
     System.out.println(a);
     System.out.println(first);
     System.out.println(second);
}
}
```

```
s
5
5
5
```

static KEYWORD

Q.Explain with example: static block, static variable and static method. Why main method is always static and public in Java?

Q. Explain the uses of static keyword wirh suitable example.

The ***static*** keyword in Java is mainly used for memory management. **static** keyword is used to declare members (variables and methods) that belong to the class itself rather than to instances of the class.

Here are some characteristics of the static keyword in Java:

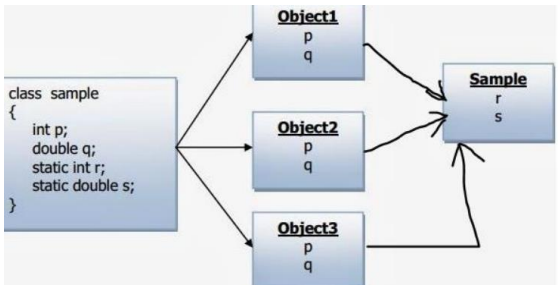
- Static variables and methods are allocated memory space only once during the execution of the program. This memory space is shared among all instances of the class.
- Static members are associated with the class, not with individual objects so static members can be accessed without the need to create an instance of the class.
- Static methods and variables cannot access non-static members of a class, as they are not associated with any particular instance of the class.
- Static methods can be overloaded. However, they cannot be overridden.

The *static* keyword is a non-access modifier in Java that is applicable for the following:

1. Blocks
2. Variables
3. Methods
4. Classes

1. Static Variables

When a variable is declared as static, then a single copy of the variable is created and shared among all objects at the class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.



2. Static Methods

When a method is declared with the *static* keyword, it is known as the static method. A static member function can be called even if no objects of the class exist. A static member function can only access static data members and other static methods.

3. Static Block

We can use a static block in java, which can contain those fields and will execute them only once at the very beginning when the class is loaded into the memory. A static block is executed before the main method during the classloading.

```
public class StaticExample {
    // Static variable (class variable)
    public static int staticVar = 10;
    // Static block
    static {
      System.out.println("This is a static block.");
      staticVar = 20;
    }
    // Static method
    public static void staticMethod() {
        System.out.println("This is a static method.");
        System.out.println("Static variable (staticVar): "
        + staticVar);

    public static void main(String[] args) {
        System.out.println("Static variable (staticVar): "
+ staticVar);
        // Calling the static method
        staticMethod();
    }
}
```

Here's a breakdown of what the code does:

1. **staticVar** is a static variable (class variable) initialized to 10.
2. The static block is executed when the class is first loaded into memory. In the static block:
    - It prints the message "This is a static block."
    - It initializes the **staticVar** variable to 20.
3. **staticMethod()** is a static method:
    - It prints the message "This is a static method."
    - It updates the **staticVar** variable to 30.
    - It then prints the updated value of **staticVar**.
4. In the **main** method:
    - It prints the initial value of **staticVar**, which is 20 (due to the static block's initialization).
    - It calls the **staticMethod()**, which updates the **staticVar** to 30 and prints the updated value.

```
This is a static block.
Static variable (staticVar): 20
This is a static method.
Static variable (staticVar): 30
```

this POINTER

this is a reference variable that refers to the current object. ***this*** pointer stores the address of current calling object. When a member function is called on an object, the "this" pointer is automatically created and initialized with the address of the object that called the function. This allows the member function to access and update the data members of that specific object, rather than the data members of another object.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter)

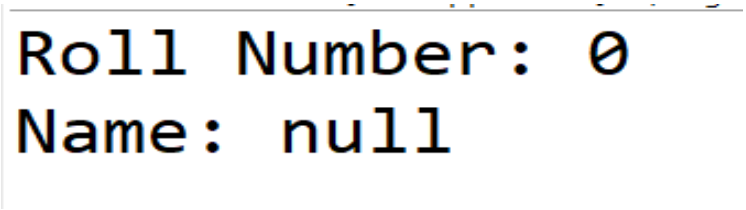
Q. In what condtition is the use of "this" operator mandatory? Explain with suitable example.

When a method or constructor has a parameter with the same name as an instance variable, we can use "this" to explicitly refer to the instance variable. This is especially important when we want to differentiate between the parameter and the instance variable.

```
class Student {
      int roll;
      String name;

     public void setData(int roll, String name) {
          this.roll = roll;
          this.name = name;
     }

     public void displayData() {
          System.out.println("Roll Number: "+roll);
          System.out.println("Name: "+name);
     }

     public static void main(String[] args) {
          Student s = new Student();
          s.setData(95, "Ankit Kharel");
          s.displayData();
     }
}
```

```
Roll Number: 95
Name: Ankit Kharel
```

If we do not use “this” keyword and roll = roll and name = name, then the output will be:



Getter & Setter

getter and setter methods are used to access and modify the private fields of a class. These methods are commonly used in encapsulation.

Getter methods retrieve the value of a private field, while setter methods modify the value of a private field.

```
class Student{
     private int age;
     private String name;

     //getter for age
     public int getAge() {
          return age;
     }
     //setter for age
     public void setAge(int age) {
          this.age = age;
     }
     //getter for name
     public String getName() {
          return name;
     }
```

```
    //setter for age
    public void setName(String name) {
        this.name = name;
    }
    public static void main(String[] args) {
        Student s  = new Student();
        s.setName("Ankit Kharel");
        s.setAge(27);
    System.out.println("Name of Student: "+ s.getName());
    System.out.println("Age: "+ s.getAge());
    }

}
```

```
Name of Student: Ankit Kharel
Age: 27
```

CONSTRUCTOR

A constructor is a special member function of a class that is called when an object of that class is created. It is used to initialize the object's data members and perform any other necessary setup operations.

- Constructor name must be the same as its class name.
- A Constructor must have no explicit return type.
- A Java constructor cannot be abstract, static, final, and synchronized.

TYPES OF CONSTRUCTOR

There are two types of constructors in Java:

- Default constructor (no-argument constructor)
- Parameterized constructor

1. Default Constructor

A constructor that accepts no arguments (parameters) is called the default constructor. If a class does not include any constructor, the compiler supplies a default constructor.

Syntax:

```
class class_name
{
        class_name(){
        }
}
```

Example:

```
class Sample {

  int a;
  boolean b;
```

```
  String str;

   Sample() {
    System.out.println("Defalut Constructor");
    a = 0;
    b = false;
    str = "Ankit Kharel";
  }

  public static void main(String[] args) {
    // call the constructor
    Sample obj = new Sample();
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
    System.out.println("str = " + obj.str);
  }

}
```

```
Defalut Constructor
a = 0
b = false
str = Ankit Kharel
```

2. Parameterized Constructor

A parameterized constructor is a constructor that takes one or more parameters. It is used to initialize the data members of a class with values passed as arguments when an object of that class is created. It is necessary to pass the argument during object creation if the class contains only parametrized constructor.

Syntax:

```
class class_name
{
        class_name( parameter_list ){
        }
}
```

Example:

```
class Sample {

  int a;
  boolean b;
  String str;

   Sample(int a, boolean b, String str) {
     System.out.println("Parameterized Constructor");
     this.a = a;
     this.b = b;
     this.str = str;
   }
```

```
  public static void main(String[] args) {
    // call the constructor
    Sample obj = new Sample(12, true, "Ankita");
    System.out.println("a = " + obj.a);
    System.out.println("b = " + obj.b);
    System.out.println("str = " + obj.str);
  }
}
```

```
Parameterized Constructor
a = 12
b = true
str = Ankita
```

CONSTRUCTOR OVERLOADING

Constructor overloading is a feature that allows a class to have more than one constructor with the same name but with different parameters. When an object is created, the appropriate constructor is called based on the arguments passed in the object creation statement.

```
class Student {
      int roll;
      String name;
     //default constructor
      Student() {
          System.out.println("Default Constructor");
          roll = 1;
          name = "Ankit";
     }
```

```
    //parameterized constructor
    Student(int roll, String name){
        System.out.println("Parameterized Constructor");
        this.roll = roll;
        this.name = name;
    }
    //constructor to initialize another object
    Student( Student s){
        System.out.println("Copying from another object");
        roll = s.roll;
        name = s.name;
    }
    public void display() {
        System.out.println("Name: "+name+" Roll: "+roll);
    }
    public static void main(String[] args) {
        Student s1 = new Student();
        s1.display();
        Student s2 = new Student(95, "Ram");
        s2.display();
        Student s3 = new Student(s2);
        s3.display();
    }
}
```

```
Default Constructor
Name: Ankit Roll: 1
Parameterized Constructor
Name: Ram Roll: 95
Copying from another object
Name: Ram Roll: 95
```

INNER CLASS

An *inner class* in Java is defined as a class that is declared inside another class. The advantages of using inner class are:

- Inner classes can access the private members of their outer classes.
- Inner classes help organize related classes within a single outer class, making the code more readable and structured.
- Inner classes are often used for event handling, such as in graphical user interfaces (GUIs)

The following are some important things to remember when working with Inner classes in Java:

- An inner class can be declared as public, private, or protected.
- An inner class can extend any class and implement any interface.
- It should be noted that if an inner class has been marked as static, it cannot access non-static members of the outer class. It can access static members of the outer class.
- We cannot create an instance of an inner or nested class without an instance of the outer class.

```
public class Student {
     // Variable in the outer class
     private String name;
      // Initialize the outer class variable
     public void setName(String name) {
          this.name = name;
     }
     //member Inner class
     public class Transcript{
          // Variable in the inner class
```

```
        private int roll;
         // Initialize the inner class variable
        public void setRoll(int roll) {
            this.roll = roll;
        }

// Method to display both student name and roll number
        public void displayStudentDetails() {
            System.out.println("Student Name: " + name);
            System.out.println("Roll Number: " + roll);
        }
    }
    public static void main(String[] args) {
        Student s = new Student();
      // Creating an instance of the inner class
      Transcript t = s.new Transcript();
      s.setName("Ankit Kharel");
      t.setRoll(95);
      t.displayStudentDetails();
    }
}
```

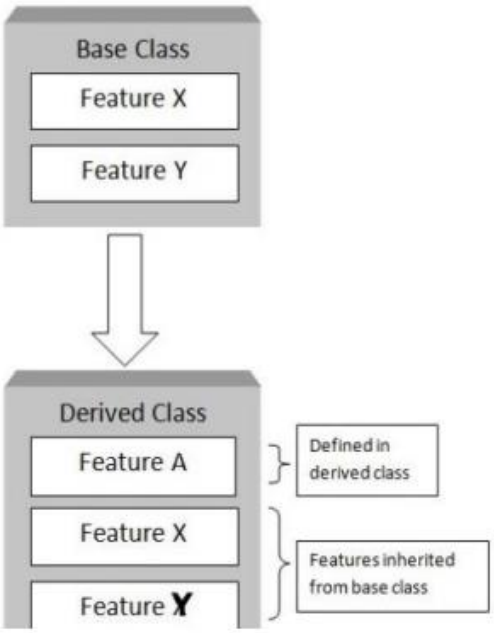
```
Student Name: Ankit Kharel
Roll Number: 95
```

CHAPTER: 2

OOP IN JAVA

INHERITANCE (IS-A Relationship)

Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called **"derived class"** or **"child class"** or **"Sub class"** and the existing class is known as the **"base class"** or **"parent class"** or **"Super class"..** The derived class now is said to be inherited from the base class.



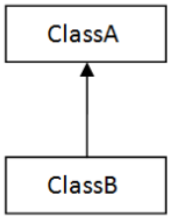
Syntax:

```
class Subclass_name extends Superclass_name  {
   //methods and fields
}
```

TYPES OF INHERITANCE

1. Single Inheritance

In this type of inheritance one derived class inherits from only one base class. It is the simplest form of Inheritance.

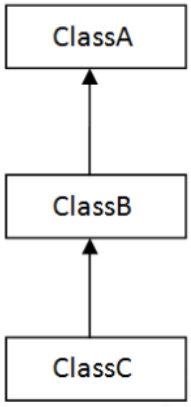



```
class Animal{
     void eat() {
          System.out.println("Eating!!!!!!");
     }
}
class Dog extends Animal{
     void bark() {
          System.out.println("Barking!!!!");
     }
}
class Test{
     public static void main(String[] args) {
          Dog d = new Dog();
          d.eat();
          d.bark();
     }
}
```

```
Eating!!!!!!
Barking!!!!
```

2. Multilevel Inheritance

Multilevel inheritance is a type of inheritance where a derived class inherits from a base class, and then another derived class inherits from the first derived class, forming a "multi-level" inheritance hierarchy.



2) Multilevel

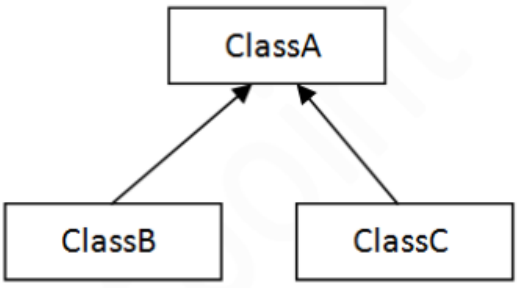
```
class Animal{
     void eat() {
          System.out.println("Eating!!!!!!");
     }
}
class Dog extends Animal{
     void bark() {
          System.out.println("Barking!!!!");
     }
}
```

```
class BabyDog extends Dog{
     void weep() {
          System.out.println("Weeping!!!");
     }
}
class Test{
     public static void main(String[] args) {
          BabyDog b = new BabyDog();
          b.eat();
          b.bark();
          b.weep();
     }
}
```

```
Eating!!!!!!
Barking!!!!
Weeping!!!
```

3. Hierarchical Inheritance

When two or more than two classes are derived from one base class, it is called hierarchical inheritance.



3) Hierarchical

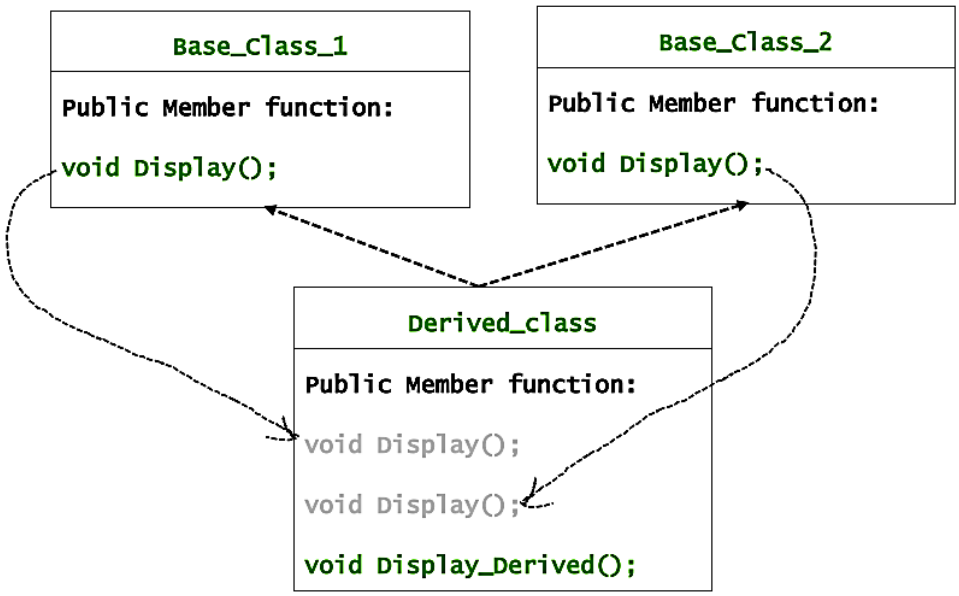
```
class Animal{
     void eat() {
          System.out.println("Eating!!!!!!");
     }
}
class Dog extends Animal{
     void bark() {
          System.out.println("Barking!!!!");
     }
}
class Cat extends Animal{
     void meow() {
          System.out.println("Meowing!!!");
     }
}
class Test{
     public static void main(String[] args) {
          Cat c = new Cat();
          c.eat();
          c.meow();
          Dog d = new Dog();
          d.eat();
          d.bark();
     }
}
```

```
Eating!!!!!!
Meowing!!!
Eating!!!!!!
Barking!!!!
```

Q. Why is Multiple Inheritance not Supported in Java. Provide a sample code to support your answer.

Multiple Inheritance is not supported in Java because of Ambiguity Problem.

In multiple inheritances, when one class is derived from two or more base classes then there may be a possibility that the base classes have functions with the same name, and the derived class may not have functions with that name as those of its base classes. If the derived class object needs to access one of the similarly named member functions of the base classes then it results in ambiguity because the compiler gets confused about which base's class member function should be called.




```
class BaseClass_1{
     void Display() {
          System.out.println("I am Base Class 1");
     }
}
class BaseClass_2{
     void Display() {
          System.out.println("I am Base Class 2");
     }
}
```

```
class DerivedClass extends BaseClass_1, BaseClass_2{
     void Display() {
          System.out.println("I am Derived Class");
     }
}
class Test{
     public static void main(String[] args) {
          DerivedClass d = new DerivedClass();
          d.Display();//Ambiguity Problem
     }
}
```

In this example, both Base_class_1 and Base_class_2 define a member function display (). When we try to access display () through the Derived_class, we get an ambiguity error because the compiler doesn't know which display () to call.

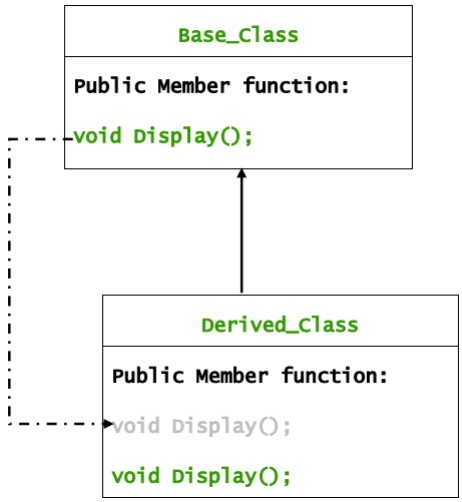
To address this issue, Java introduced the concept of interfaces.

METHOD OVERRIDING(Run-time polymorphism)

Suppose base class and derived class has a member function with exactly same name, return type and arguments. Now, if we call the function with the object of derived class then the derived class function gets called. This is known as function overriding.

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

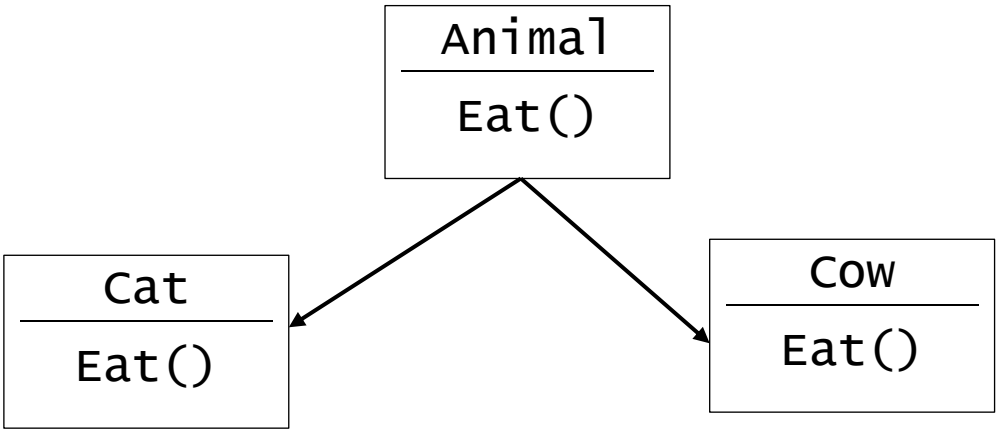



```
class BaseClass{
     void Display() {
          System.out.println("I am Base Class");
     }
}
class DerivedClass extends BaseClass{
     void Display() {
          System.out.println("I am Derived Class");
     }
}
```

```
class Test{
     public static void main(String[] args) {
          DerivedClass d = new DerivedClass();
          d.Display();
     }

}
```

```
I am Derived Class
```

Q. Write a code Animal as Base class, cat(derived class) and Cow(derived class). Write a eat() method in animal class and override method in the derived class.




```
class Animal {
    void eat() {
        System.out.println("Animal is eating.");
    }
}
```

```
class Cat extends Animal {
    @Override
    void eat() {
        System.out.println("Cat is eating.");
    }
}

class Cow extends Animal {
    @Override
    void eat() {
        System.out.println("Cow is eating.");
    }
}

public class Test {
    public static void main(String[] args) {
        Animal a = new Animal();
        Animal b = new Cat();//Up cating
        Animal c = new Cow();//Up casting
        a.eat();
        b.eat();
        c.eat();
    }
}
```

```
Animal is eating.
Cat is eating.
Cow is eating.
```

Upcasting: In Java, upcasting allows you to assign an instance of a subclass to a reference variable of its superclass.

Q. Difference Between Method Overloading and Method Overriding.

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.

Q. Create a class Employee with id, name, post and salary. Create a parameterized constructor to initialize the instance variables . Overide the toString() method to display the employee details.

*The **toString()** method of the Object class returns the string representation of an object in Java. If we print any object, the Java Compiler internally invokes the toString() method on that object. The value returned by the toString() method is then printed.*

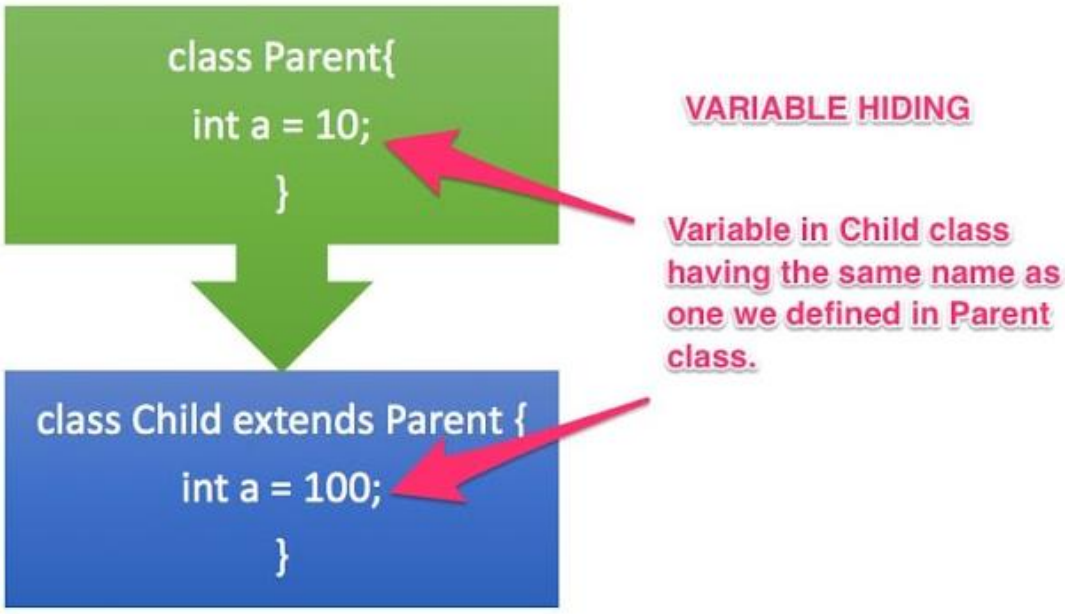
```
public class Employee {
    private int id;
    private String name;
    private String post;
    private double salary;

    // Parameterized constructor to initialize instance variables
    public Employee(int id, String name, String post, double salary)
     {
        this.id = id;
        this.name = name;
        this.post = post;
        this.salary = salary;
    }

    // Override the toString() method to display employee details
    @Override
    public String toString() {
        return "Employee ID: " + id + "\n" +
               "Name: " + name + "\n" +
               "Post: " + post + "\n" +
               "Salary: Rs" + salary;
    }
```

```
    public static void main(String[] args) {
        // Create an Employee object and initialize it using the constructor
  Employee employee = new Employee(12, "Ankit Kharel", "JP", 60000.0);
        // Display employee details using toString()
        System.out.println(employee);
    }
}
```

```
Employee ID: 12
Name: Ankit Kharel
Post: JP
Salary: Rs60000.0
```

Q. Discuss about the concept of instance variable hiding in java. Illustrate with a code .

When a class inherits another class, if the parent class and child class both have variable declarations with the same name then the variable declared inside child class will hide the inherited version.



```
class Parent
{
    int a = 10;
    String str = "Parent Class";
}
class Child extends Parent
{
    int a = 100;
    String str = "Child Class";
}
public class Test
{
    public static void main(String args[])
    {
        Child obj = new Child();
  System.out.println("Value of variable a is: "+obj.a);
  System.out.println("Value of variable str is: "+obj.str);
    }
}
```

```
Value of variable a is: 100
Value of variable str is: Child Class
```

Q. Difference Between Variable shadowing and Variable Hiding.

```
public class Test {
private int x = 5; // Class-level variable

public void doSomething(int x) {
// The parameter x shadows the class-level variable x within this
method(Variable shadowing)
System.out.println(x); // Refers to the method parameter, not the
class variable
}
}
```

Q. Can we override private or static method in Java? Explain with an example.

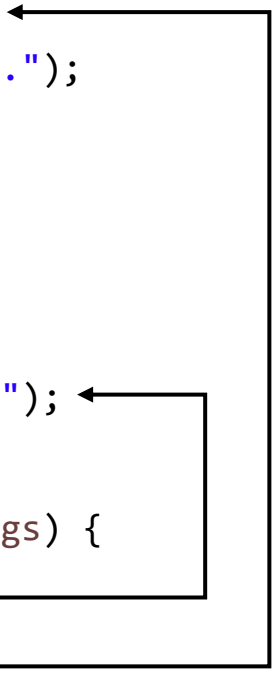
In Java, it's not possible to override private methods, and static methods are not overridden in the traditional sense but can be "hidden" in subclasse.

1. **Private Methods:** Private methods in a class are not accessible outside that class, including in subclasses. Therefore, we cannot override a private method in a subclass. If we define a method in a subclass with the same name and signature as a private method in the superclass, it's treated as a new method in the subclass, not an override.

```
class Parent {
    private void privateMethod() {
        System.out.println("I am Parent.");
    }
}
class Child extends Parent {
    private void privateMethod() {
        System.out.println("I am child.");
    }
    public static void main(String[] args) {
         Child c = new Child();
         c.privateMethod();
         Parent p = new Child();
         p.privateMethod();
// The method privateMethod() from the type Parent is not
visible at  Child.main
     }
}
```

2. **Static Methods:** In Java, static methods are associated with the class itself, not with instances of the class. While we can declare a static method with the same name in a subclass, it's not considered method overriding. Instead, it's method hiding

```
class Parent {
    public static void staticMethod() {
        System.out.println("I am Parent.");
    }
}
class Child extends Parent {
    public static void staticMethod() {
        System.out.println("I am child.");
    }
    public static void main(String[] args) {
         staticMethod();
         Parent.staticMethod();
     }
}
```



super KEYWORD

The ***super*** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever we create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

a. to refer immediate parent class instance variable (use of super with variable).

b. to invoke immediate parent class method. (use of super with methods)

c. to invoke immediate parent class constructor. (use of super with constructors)

a. Use of super with variable.

We can use super keyword to access the data member of parent class if parent class and child class have same name for data member.

```
class Vehicle {
     int maxSpeed = 120;
}
class Car extends Vehicle {
     int maxSpeed = 180;
     void display()
     {
     system.out.println("MaximumSpeed:"+super.maxSpeed);
     }
}
class Test {
     public static void main(String[] args)
     {
          Car c = new Car();
          c.display();
     }
}
```

```
Maximum Speed: 120
```

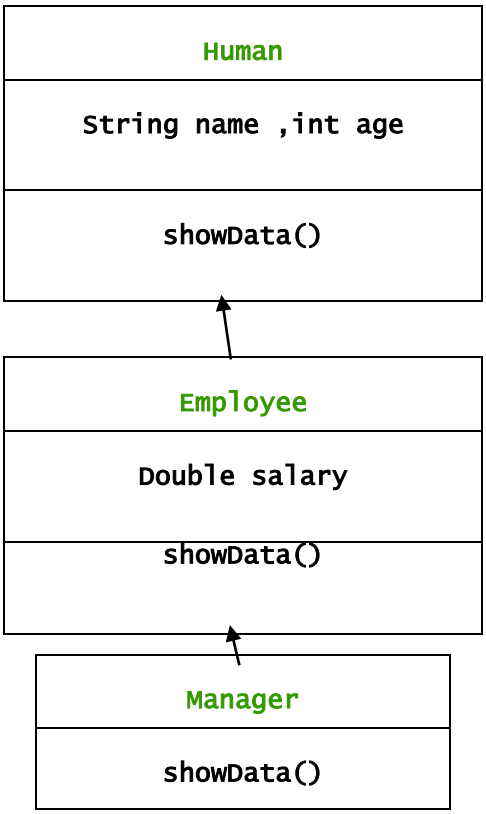
b. Use of super with methods.

The super keyword can also be used to invoke parent class method if subclass contains the same method as parent class.

```
class Person {
     void message() {
          System.out.println("I am Person.");
     }
}
class Child extends Person{
     void message()
     {
          super.message();
          System.out.println("I am Child");
     }
}
class Test {
     public static void main(String[] args)
     {
          Child c = new Child();
          c.message();
     }
}
```

```
I am Child
I am Person.
```

Q. Make a class Human with name and age. Make a class Employee inherit from Human. Add instance variable salary of type double, supply a method showData() that prints the Employee's name, age and salary. Make a class manager inherit from employee. Supply showData() methods for all classes . Provide a tst program that tests these classes and methods




```
class Human{
	String name;
	int age;
	Human(String name, int age){
		this.name = name;
		this.age = age;
	}
	void showData() {
		System.out.println("Name: "+name+"\nAge: "+age);
	}
}
```

```
class Employee extends Human{
     double salary;
     Employee(String name, int age, double salary){
          super(name, age);//calling Base class constructor
          this.salary=salary;
     }
     void showData() {
          super.showData();
          System.out.println("Salary: "+salary);
     }
}
class Manager extends Employee{
     int experience;
Manager(String name, int age, double salary, int experience){
    super(name,age,salary);//calling Base classconstructor
          this.experience = experience;
     }
      void showData() {
          super.showData();
   System.out.println("Years of Experience: "+experience);
     }
}
class Test{
     public static void main(String[] args) {
          Manager m = new Manager("Ankit", 28, 50000, 4);
          m.showData();
     }

}
```

```
Name: Ankit
Age: 28
Salary: 50000.0
Years of Experience: 4
```

final KEYWORD

The **final** keyword is used to denote that a particular element, such as a variable, method, class, or parameter, cannot be further modified, extended, or overridden once it has been declared as "final".

1. final Variable

If we make any variable as **final**, you cannot change the value of final variable(It will be constant).

```
class Bike9{
 final int speedlimit=90; //final variable
 void run(){
  speedlimit=400;  //not allowed

 }
```

2. final Method

When applied to a method, the **final** keyword indicates that the method cannot be overridden by subclasses.

```
class Parent {
    final void display() {
    System.out.println("final method in Parent class.");
    }
}
class Child extends Parent {
    void display() {
      // Error: Cannot override the final method display()
     }
}
```

3. final Class

When applied to a class, the **final** keyword makes the class itself immutable, meaning it cannot be extended by other classes.

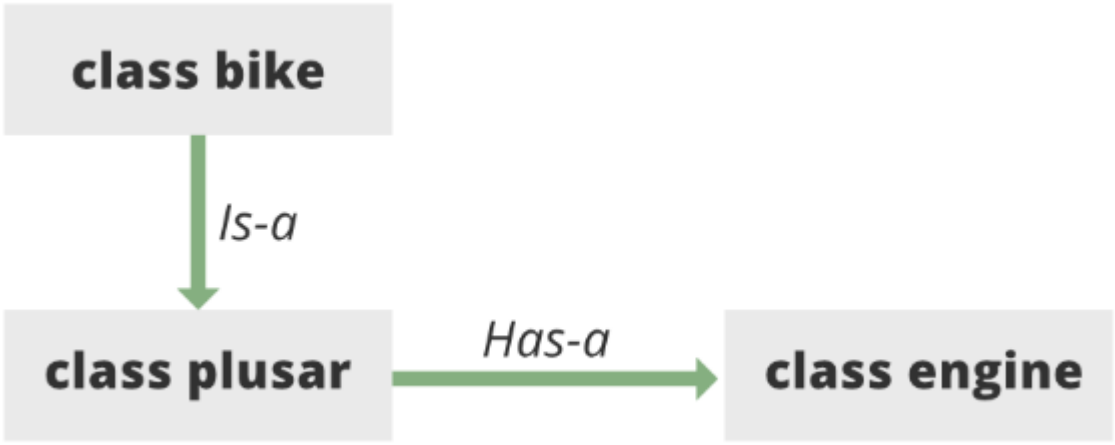
```
final class FinalClass {
    // Class members and methods
}
// Error: Cannot inherit from final FinalClass
class AnotherClass extends FinalClass {

}
```

Q. Explain the use of super and this keyword

(Write about super and this keyword and give one code example of each**)**

AGGREGATION/COMPOSITION(Has-a relation)

If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.



Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

```
class Employee{
     int id;
     String name;
     Address address;//Address is a class
     ...
}

public class Address {
     String city,state,country;

}
```

Q. Define Inheritance and compostion. With the help of a suitable program illustrate the difference between them.

Inheritance is a mechanism where a new class (subclass or derived class) is created by inheriting properties and behaviors (fields and methods) from an existing class.

Composition is a mechanism where an object of one class is used as a component (member) within another class. The class that contains the object is called the container or composite class.

```
class Person{
     String name;
     public Person(String a) {
          name =a;
     }
     void displayPerson() {
          System.out.println("Name: "+name);;
     }
}
class Address{
     String country, zone, district;
     public Address(String a, String b, String c ) {
          country =a;
          zone = b;
          district = c;
     }
     void displayAddress() {
          System.out.println("Country: " + country);
        System.out.println("Zone: " + zone);
        System.out.println("District: " + district);
     }
}
```

```
class Student extends Person{
     int roll;
     Address add;
     public Student(String a, int b, Address ad) {
          super(a);
          roll =b;
          add = ad;
     }
     void displayStudent() {
               System.out.println("Roll: " + roll);
              add.displayAddress();
     }
}
class inheritanceComposition{
      public static void main(String[] args) {
Address a = new Address("Nepal", "Mechi", "Dhulabari");
           Student s = new Student("Ankit", 95, a);
           s.displayPerson();
           s.displayStudent();
      }
}
```

```
Name: Ankit
Roll: 95
Country: Nepal
Zone: Mechi
District: Dhulabari
```

ABSTRACTION

Q. What are the uses of abstract keyword? Explain with a suitable example.

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

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

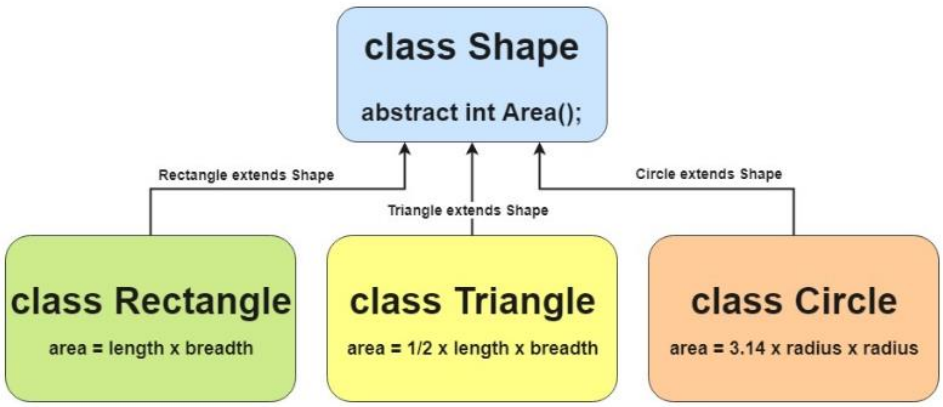
1. ABSTRACT CLASS

An abstract class in Java is a class that cannot be instantiated on its own and is typically used as a blueprint or template for creating concrete (non-abstract) subclasses. It can have abstract and non-abstract methods. It is declared using "abstract" keyword.

Abstract Methods: A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

abstract void Draw();//no method body and abstract



```
abstract class Shape{
     public abstract void area();
     public void display() {
          System.out.println("I am abstract class");
     }
}
class Triangle extends Shape{
     float height, base;
     Triangle(float a, float b){
          height=a;
          base = b;
     }
     public void area() {
System.out.println("Area of Triangle: "+ 0.5*base*height);
     }
}
class Test{
     public static void main(String[] args) {
          Triangle t = new Triangle(5, 4);
          t.display();
          t.area();
     }

}
```

```
I am abstract class
Area of Triangle: 10.0
```

INTERFACE

Q. What are the significant uses of interface? Explain how it is implemented in java? Explain with suitable example.

An interface is a fully abstract class. It has static constants and abstract methods. It is also used to support the functionality of multiple inheritance. An interface is not extended by a class; it is implemented by a class.

Significance of interfaces are:

- It is used to achieve total abstraction.
- Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
- Any class can extend only 1 class but can any class implement infinite number of interface.
- It is also used to achieve loose coupling.
- Interfaces are used to implement abstraction.

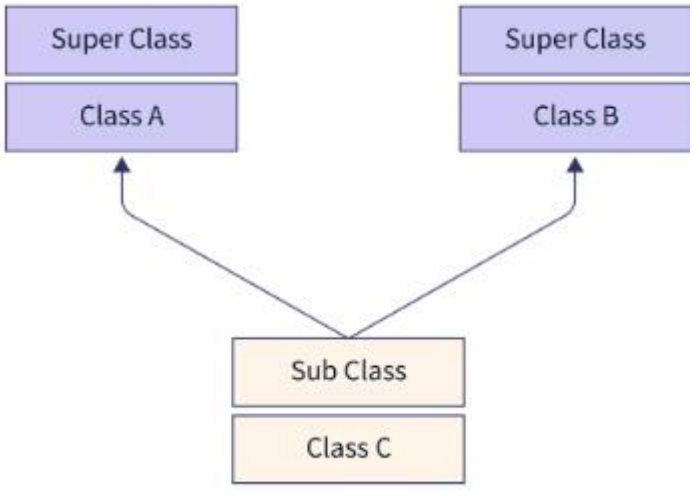
```
Syntax:
     interface interface_name
     {
        //abstract methods and constants
}
```

```
interface Polygon {
  void getArea(int length, int breadth);
}
class Rectangle implements Polygon {
  // implementation of abstract method
  public void getArea(int length, int breadth) {
    System.out.println("The area of the rectangle is " +
     (length * breadth));
  }
}
class Triangle implements Polygon{
      // implementation of abstract method
       public void getArea(int base, int height) {
         System.out.println("The area of the rectangle is "
          + (0.5*base * height));
       }
}
class Test {
  public static void main(String[] args) {
    Rectangle r = new Rectangle();
    r.getArea(5, 6);
    Triangle t = new Triangle();
    t.getArea(4, 2);
  }

}
```

```
The area of the rectangle is 30
The area of the rectangle is 4.0
```

Q. How is multiple inheritance implemented in Java. Exaplain with code example.

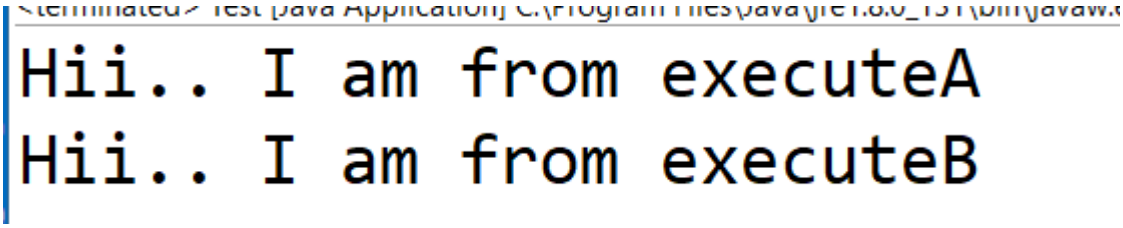
Multiple Inheritance is the process in which a subclass inherits more than one superclass.



Java does not support Multiple Inheritance, but we can use **Interfaces** to achieve the same purpose.

```
interface A
{
    public abstract void executeA();
}
interface B
{
    public abstract void executeB();
}
class C implements A,B
{
    public void executeA()
    {
        System.out.println("Hii.. I am from executeA");
    }
```

```
    public void executeB()
    {
        System.out.println("Hii.. I am from executeB");
    }
}
class Test
{
     public static void main(String[] args)
     {
          C obj = new C(); // creating object of class C
          obj.executeA(); //calling method execute1
          obj.executeB(); // calling method execute2
     }
}
```



From the above output, we could observe that though interface A and interface B contain the same method with the same signature, **its implementation is provided in the implementation class C**. Interface methods are abstract by default, so they don't contain a definition, due to which there is only one definition of execute(), and the object of the child class had only to call that.

Q. How does interface differ from abstract classes? Elaboarate using code snippets to justify.

Interfaces and abstract classes in Java both serve as mechanisms for achieving abstraction and defining contracts for classes. However, they have some key differences in terms of their purpose and usage.

ABSTRACT CLASS	INTERFACE
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) It can have main method and constructor.	It cannot have main method and constructor.
7) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

Give one example of Abstract class and Interface.

CHAPTER: 3

EXCEPTION, STREAM AND I/O

EXCEPTION HANDLING

An exception is an unwanted or unexpected event that occurs during the execution of the program, that disrupts the flow of the program. s. 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

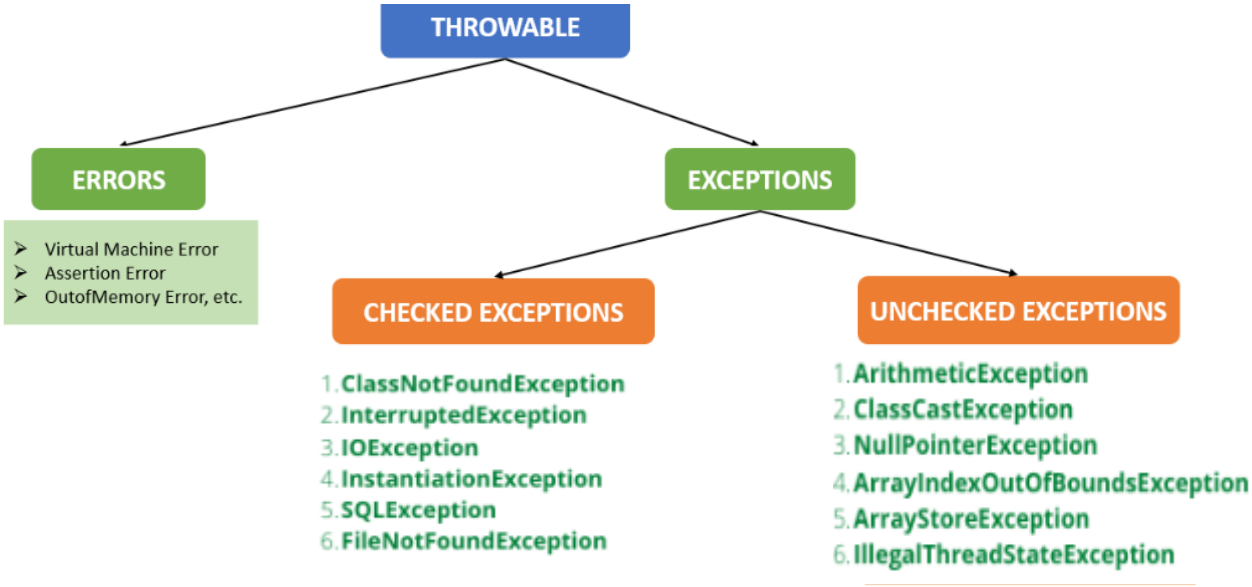
• JVM has run out of memory.

It is necessary to handle Exception because of following reasons (**Significane/Uses of Exception Handling**):

- Handling exceptions prevents program crashes and ensures that the application remains stable during runtime.
- Even when errors occur, we can exit the program gracefully and provide informative error messages instead of abrupt termination.
- It improves the user experience by preventing the application from displaying cryptic error messages or crashing, making it more user-friendly.

EXCEPTION HIERARCHY

All exception and error types are subclasses of the class **Throwable**, which is the base class of the hierarchy.



There are mainly two types of exceptions in Java as follows:

1. CHECKED EXCEPTION

Checked exceptions are also known as compile-time exceptions as these exceptions are checked by the compiler during the compilation process to confirm whether the exception is handled by the programmer or not. If not, then the system displays a compilation error.

For example, ***SQLException***, ***IOException***, ***ClassNotFoundException***.

SQLException: This exception provides detail about SQL database error if it occurs. The error can be SQL syntax or SQL driver.

IOException: It is an input/output exception and it occurs whenever an input or output operation fails or is interpreted in java. For instance, when you try to read a file that does not exist, it will throw an IOException.

ClassNotFoundException: It is a kind of checked exception that occurs when a **Java virtual machine** (JVM) tries to load a class but fails because it is unable to find the classpath.

2. UNCECKED EXCEPTION

These are also called *runtime exceptions.* These exceptions occur during program execution. These exceptions are not checked at compile time, so the programmer is responsible for handling these exceptions. Unchecked exceptions do not give compilation errors.

For Examples **, *ArithmeticException*** and ***ArrayIndexOutOfBoundException***.

ArithmeticException: It is a type of unchecked error in code that is thrown whenever there is a **wrong mathematical** or **arithmetic calculation** in the code, especially during run time.

ArrayIndexOutOfBoundException: It occurs when we access an array with an invalid index. This means that either the index value is less than zero or greater than that of the array's length.

InputMismatchException: It occurs when an input provided by the user is incorrect. The type of incorrect input can be out of range or incorrect data type.

Q. Difference between Error and Exception

ERRORS	EXCEPTIONS
Typically, errors are caused by the Java runtime environment or external factors beyond the programmer's control.	Exceptions are typically caused by issues within the program's code.
Errors are generally considered to be unrecoverable and can lead to the termination of the program.	Exceptions are recoverable and can be caught and handled in code to prevent program termination.
Errors are not usually handled programmatically because they often signify critical issues that can't be resolved at the application level.	Exceptions are meant to be caught and handled using try-catch blocks or propagated up the call stack.
Errors are unchecked.	Exceptions can be either checked or unchecked
OutOfMemoryError, **StackOverflowError**, **NoClassDefFoundError**	**NullPointerException**, **FileNotFoundException**, **NumberFormatException**

try, catch, throw , throws, finally

Q. Define Exception handling with suitable example.

Exception handling in Java is a mechanism that allows us to gracefully handle unexpected or exceptional conditions that may arise during the execution of a program. Java provides a structured way to handle exceptions using try, catch, throw, and throws keywords

a. try Block: the **try** block includes the statements that might throw an exception during execution.

Syntax:

```
try
{
    // Statements with possible Exceptions

}
```

b. catch Block: the **catch** statement allows us to define a block of code that will be executed to handle the exception.

Syntax:

```
catch(Exception e)
{
    // Code to handle the possible Exceptions

}
```

c. throw Keyword: The **throw** keyword is used to explicitly throw a single exception. When we throw an exception, the flow of the program moves from the try block to the catch block.

Syntax:

```
throw throwableObject;
```

d. throws Keyword: The **throws** keyword is used in method signatures to declare that a method may throw a particular type of exception.

Syntax:

```
public void someMethod() throws SomeException {
    // Code that may throw SomeException
}
```

e. finally Block: The **finally** block is optional and follows the **try-catch** blocks. It is used to define code that will always be executed, whether an exception occurred or not.

Syntax:

```
finally
{
    // Code that always runs, even if an exception occurred

}
```

Example:

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
            int numerator = 10;
            int denominator = 0;
            int result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (Exception e) {
            System.out.println("An arithmetic exception
              occurred: " + e.getMessage());
        }
```

```
        finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

```
An arithmetic exception occurred: / by zero
Finally block executed.
```

Q. Write a program to input from keyboard and print it on the console. Fire an exception if the input is other than integer using try and catch block.

```
import java.util.*;
class ExceptionExample{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        try {
            System.out.println("Enter a number:");
            int num = sc.nextInt();
            System.out.println("The number is: "+num);
        }catch(InputMismatchException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
Enter a number:
A
null
```

CUSTOM EXCEPTION

Q. How can you create your own exception in java? Explain with program.

Custom exceptions are useful when you want to handle specific error conditions in your code that are not adequately represented by built-in Java exceptions. Steps to Create a Custom Exception in Java are as following:

1. Define a new class that extends the **Exception** class.
2. Create a Constructor with String statement as an argument in the new class. String as an argument allows us to custom message each time we throw an exception.
3. Call **super(statement)** i.e. Constructor method of a superclass with a statement as input. This will print our custom message with the exception.
4. Within the code, identify the conditions where we want to throw our custom exception. Enclose the code in **try** block and use the **throw** keyword to create and throw an instance of our custom exception.
5. In the **catch** block, handle the custom exception as needed.

```
import java.util.*;

class AnkitException extends Exception{
     public AnkitException(String str) {
          super(str);
     }
}

class Test{
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          System.out.println("Enter Age: ");
          int age = sc.nextInt();
```

```
    try {
        if(age<18) {
   throw new AnkitException("Age must be greater than 18!");
        }
        else {
        System.out.println("You can Vote\nHappy Voting");
        }
    }catch(AnkitException e){
            System.out.println(e.getMessage());
    }
}
}
```

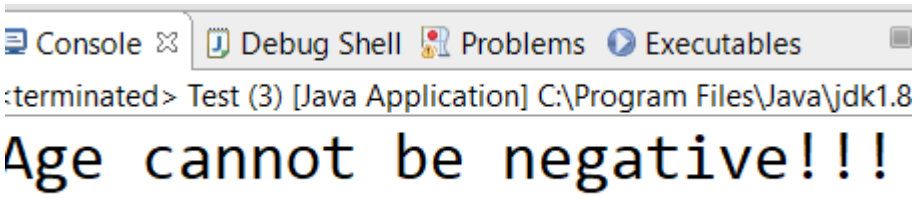
```
Enter Age:
12
Age should be greater than 18!
```

```
Enter Age:
22
You can Vote
Happy Voting
```

Q. Create a class called Student with name and age. Fire a custom exception if age is negative.

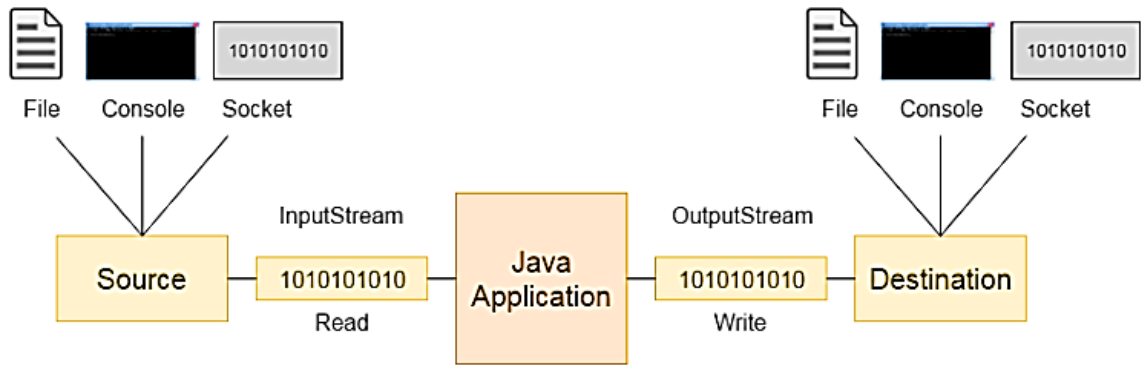
```
class AgeException extends Exception{
     AgeException(String str){
          super(str);
     }
}

class Student{
     String name;
     int age;
     Student(String name, int age) throws AgeException{
     if(age<0) {
      throw new AgeException("Age cannot be negative!!!");
     }
          this.name=name;
          this.age=age;
     }
     void display() {
          System.out.println("Name: "+name);
          System.out.println("Age: "+age);
     }
}

public class Test {
     public static void main(String[] args) {
          try {
               Student s = new Student("Ankit Kharel", -2);
```

```
        }catch(AgeException e) {
            System.err.println(e.getMessage());
        }
    }
}
```



JAVA I/O STREAM

Streams are the sequence of data that are read from the source and written to the destination.

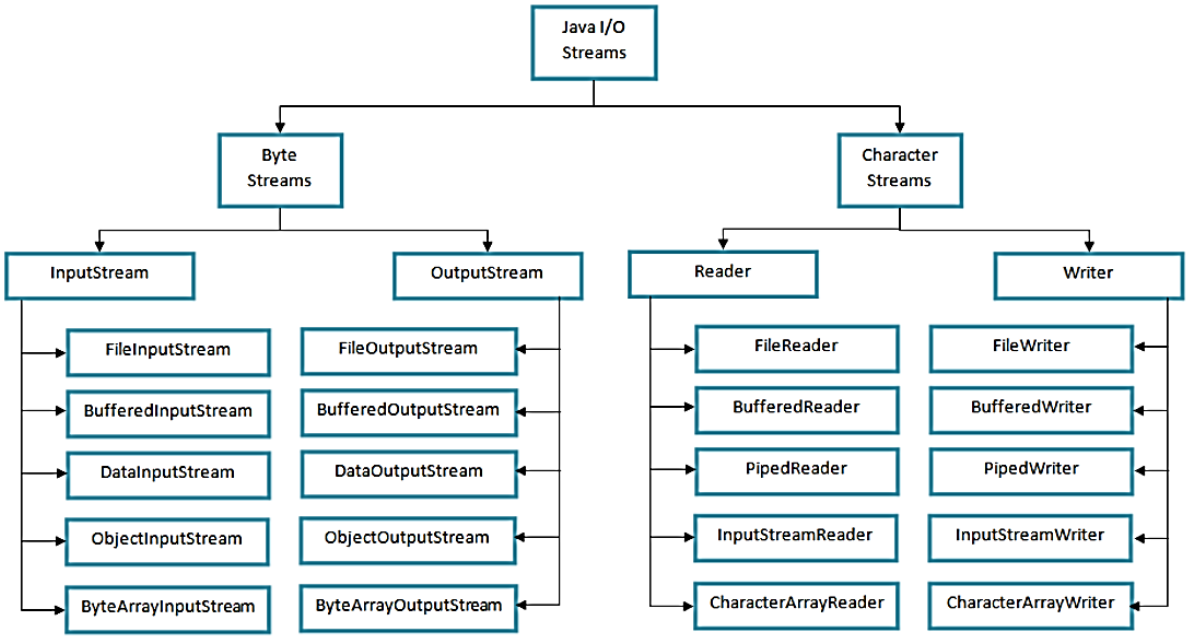


In Java, Input/Output (I/O) streams are used to handle the input and output of data to and from various sources, such as files, network connections, or memory. Java provides a set of classes and methods for working with I/O streams, which are part of the java.io package.

STREAM CLASSES

Q. Explain various stream class available in java. WAP using any stream class for writing text to a file.

The Java Input/Output (I/O) is a part of **java.io** package. This package contains a relatively large number of classes that support input and output operations. These classes may be categorized into groups based on the data type on which they operate.



1. **Byte Stream Classes:** It provides support for handling I/O operations on **bytes**. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**.

   (a) FileOutputStream: This writes data into a specific file. It is suitable for writing non-textual data like images, audio, video, or any data that doesn't have a specific character encoding. To write the contents of a file using this class we should perform following steps:

   - Instantiate this class by passing a String variable, representing the path of the file to be read.

     FileOutputStream fos = new FileOutputStream("file_path");

   - Then write the data to a specified file using either of the variants of **write()** method:

     void write(int): writes the specified byte to the output stream.

     ***Example:* fos.write(65);** *(Writes byte with value 65 (ASCII code for 'A')*

     void write(byte[]): writes the specified array of bytes to the output stream.

     Example: fos.write("Hellow".getBytes()); *getBytes() converts the string into array of bytes. write("Hellow".getBytes()) is equivalent to write (72, 101, 108, 108, 111);*

   *These methods throw **IOException***

   (b) FileInputStream: This reads data from a specific file. It is suitable for reading non-textual data like images, audio, video, or any data that doesn't have a specific character encoding. To read the contents of a file using this class we should perform following steps:

   - Instantiate this class by passing a String variable, representing the path of the file to be read.

     FileInputStream fis = new FileInputStream("file_path");

   - Then read the data to a specified file using either of the variants of **read()** method:

     int read( ): This simply reads data from the current InputStream and returns the read data byte by byte (in integer format). This method returns -1 if the end of the file is reached.

Example:
```
int data;
while ((data = fis.read()) != -1) {
    //(cast it to char for text-based processing)
     System.out.print((char) data);
 }
```

int read(byte[ ] array_name): reads a chunk of bytes to the specified byte array(array_name), up to the size of the array. The method returns an integer value representing the number of bytes actually read. It returns -1 if there's no more data or the end of the file is reached.

Example:
```
byte[] buffer = new byte[1024]; // Define a
    buffer to read data into
int bytesRead;
while ((bytesRead = fis.read(buffer)) != -1) {
  // Process the bytes in the buffer
   for (int i = 0; i < bytesRead; i++) {
     System.out.print((char) buffer[i]); // Cast
      to char for text-based processing
   }
}
```

*These methods throw **FileNotFoundException***

2. **Character Stream Classes:** It provides support for managing I/O operations on **characters**. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**.

   (a) FileWriter: This class is used for writing character data to a specific file. It is suitable for writing textual data with proper character encoding. To write the contents of a file using this class we should perform following steps:

   - Instantiate this class by passing a String variable, representing the path of the file to be read.

     FileWriter fw = new FileWriter("file_path");

   - Then write the data to a specified file using either of the variants of **write()** method:

     void write(int c): writes a single character.

Example: **fw.write('A');** *(Writes character 'A' to the file)*

void write(String str): writes a string.

Example: **fw.write("Hellow");** *(Writes the string "Hellow" to the file)*

These methods throw ***IOException***

(b) FileReader: This reads data from a specific file. It is suitable for reading textual data with proper character encoding. To read the contents of a file using this class we should perform following steps:

- Instantiate this class by passing a String variable, representing the path of the file to be read.

  FileReader fr = new FileReader("file_path");
- Then read the data to a specified file using either of the variants of **read()** method:

  int read( ): reads a single character, and returns the character as an integer value. Returns -1 if the end of the file is reached.

Example:
```
int data;
while ((data = fis.read()) != -1) {
    //(cast it to char for text-based processing)
     System.out.print((char) data);
 }
```

  int read(char[ ] array_name): reads characters to an array(array_name), and returns the number of characters read. Returns -1 if the end of the file is reached.

Example:
```
char[] buffer = new char[1024]; // Define a
   buffer to read data into
int charsRead;
while ((charssRead = fr.read(buffer)) != -1) {
  for (int i = 0; i < charsRead; i++) {
    System.out.print((char) buffer[i]); // Cast
     to char for text-based processing
 }
}
```

These methods throw ***FileNotFoundException***

WRITING DATA TO A FILE

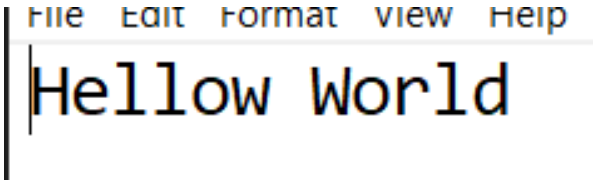
The steps for writing to the file are:

- Import java.io.*
- Handle the exception
- Create FileOutputStream or FileWriter object by passing file name as argument
- Use **write()** method for writng the data
- Close the file using **close()**

Q. Create a program to write "Hellow World" in a file "abc.txt" (using FileWriter).

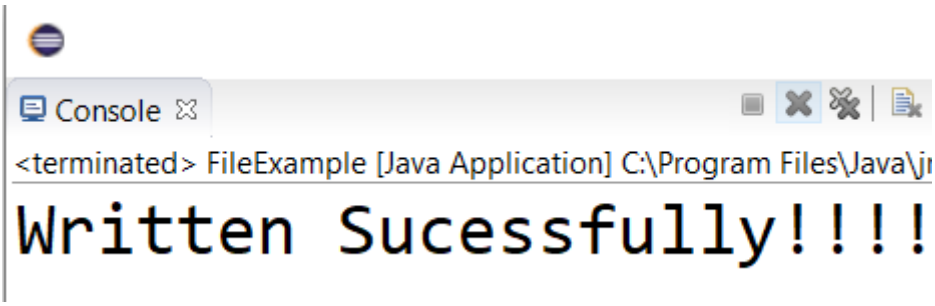
```
import java.io.*;
class FileExample{
     public static void main(String[] args) throws
     IOException{
          FileWriter fw = new FileWriter("abc.txt");
          fw.write("Hellow World");
          fw.close();
          System.out.println("Written Sucessfully!!!!");
     }

}
```

```
<terminated> FileExample [Java Application] C:\Program Files\Java\jre1.8.0_
Written Sucessfully!!!!
```



Q. Create a program to write "Hellow World" in a file "abc.txt" (using FileOutputStream).

```
import java.io.*;
class FileExample{
     public static void main(String[] args) throws
     IOException{
  FileOutputStream fos = new FileOutputStream ("abc.txt");
          fos.write("Hellow World".getBytes());
          fos.close();
          System.out.println("Written Sucessfully!!!!");
     }

}
```



READING DATA FROM A FILE

The steps for reading from the file are:

- Import java.io.*
- Create FileInputStream or FileReader object by passing file name as argument
- Handle the exception
- Use **read()** method and assign it to int type variable
- Close the file using **close()**

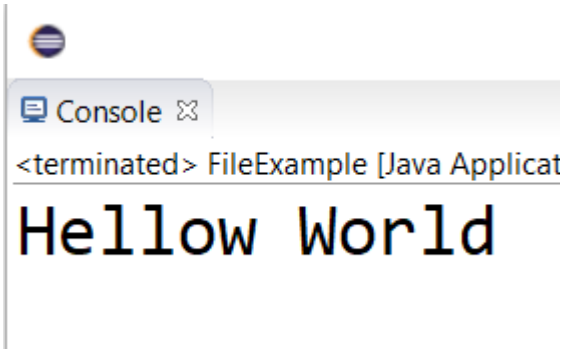
Q. Program to read "Hellow World" from a file "abc.txt". (using FileReader).

```
import java.io.*;
class FileExample {
    public static void main(String[] args) throws
     FileNotFoundException {
        FileReader fr = new FileReader("abc.txt");
        int data;
        while ((data = fr.read()) != -1) {
    // Convert the integer data to a character and print it
            System.out.print((char) data);
        }
        fr.close();
    }
}
```

```
<terminated> FileExample [Java Applicati
Hellow World
```

Q. Program to read "Hellow World" from a file "abc.txt". (using FileInputStream).

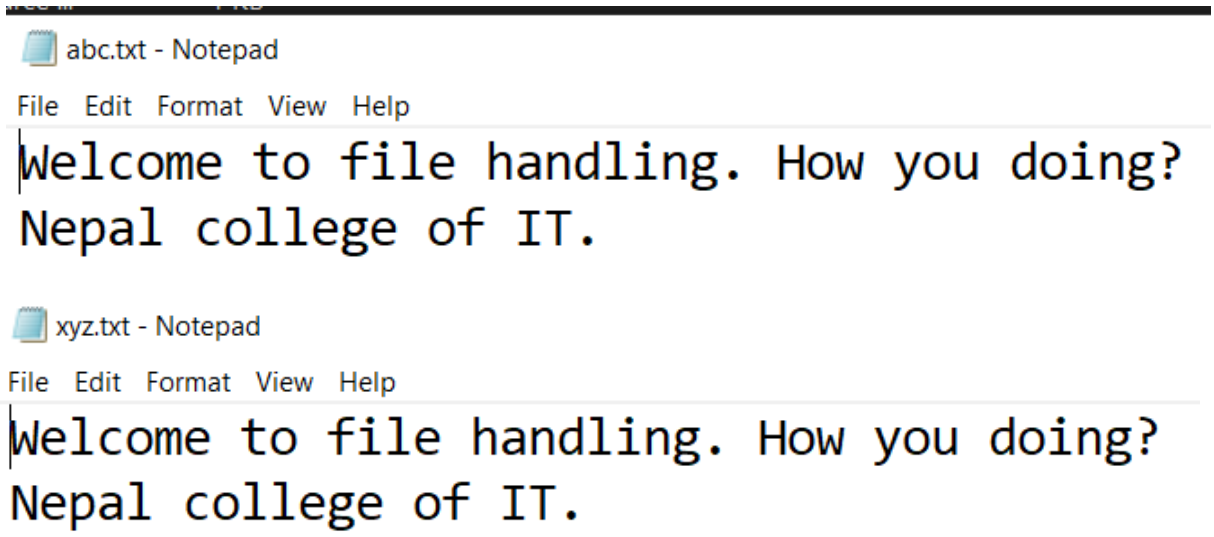
```
import java.io.*;
class FileExample {
    public static void main(String[] args) throws
     FileNotFoundException {
     FileInputStream fis = new FileInputStream ("abc.txt");
        int data;
        while ((data = fis.read()) != -1) {
    // Convert the integer data to a character and print it
            System.out.print((char) data);
        }
        fis.close();
    }
}
```



Q. WAp to copy content from"abc.txt" and store it in "xyz.txt".

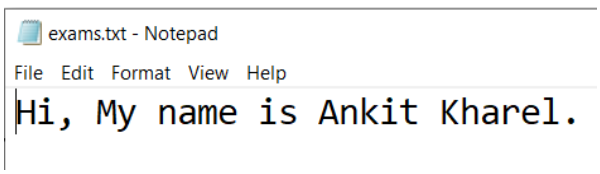
```
import java.io.*;
class FileCopyExample{
     public static void main(String[] args) throws
     IOException, FileNotFoundException{
          // Create FileReader for the source file
         FileReader fr_source = new FileReader("abc.txt");
```

```
            // Create FileWriter for the destination file
    FileWriter fw_destination = new FileWriter("xyz.txt");
          int data;
          while((data=fr_source.read())!=-1) {
               // Write the character to the destination file
               fw_destination.write(data);
          }
          System.out.println("Successfully Copied!!!!");
          fr_source.close();
          fw_destination.close();
     }

}
```



Q. WAP to write bytes of data into a file called "exams.txt"(exams.jsp) and reading from it unitl end of file is obtained using proper exception handling.

```
import java.io.*;
import java.util.*;
class FileExample{
     public static void main(String[] args) {
          Scanner sc = new Scanner(System.in);
          try {
          //Writing to the file
FileOutputStream fos = new FileOutputStream("exams.txt");
          System.out.println("Enter Some Strings: ");
          String str = sc.nextLine();
          fos.write(str.getBytes());
          System.out.println("Written Successfully!!!");
          fos.close();

          //Reading from file
FileInputStream fis = new FileInputStream("exams.txt");
          int bytesRead;
          while((bytesRead = fis.read())!=-1) {
               System.out.print ((char)bytesRead);
          }
          fos.close();
        }catch(IOException e) {
      System.out.println("Error in performing Operation.");
               e.printStackTrace();
          }
     }
}
```



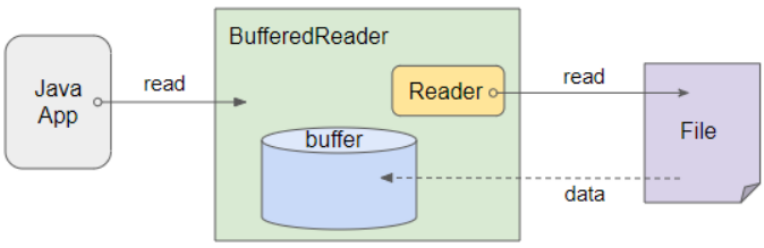
Q. WAP to copy an mp3 file.

```
import java.io.*;
class Test{
     public static void main(String[] args) throws
Exception {
     FileInputStream fis = new FileInputStream("abc.mp3");
   FileOutputStream fos = new FileOutputStream("xyz.mp3");
          int data;
          System.out.println("Copying started.........");
          while((data=fis.read())!=-1) {
               fos.write(data);
          }
          System.out.println("Copied successfully.....");
     }
}
```

```
//this program takes nearly 3 minutes for copying mp3. We
can decrease this time by using buffered stream.
```

BufferedReader and BufferedWriter Class

1. BufferedReader: It inherits Reader class which is used for reading text from a character-input stream, such as a file, in larger chunks (a buffer), making the reading process more efficient. BufferedReader wraps inside it a Reader object, which automatically reads data from the origin (such as file) and stores it in BufferedReader's buffer.



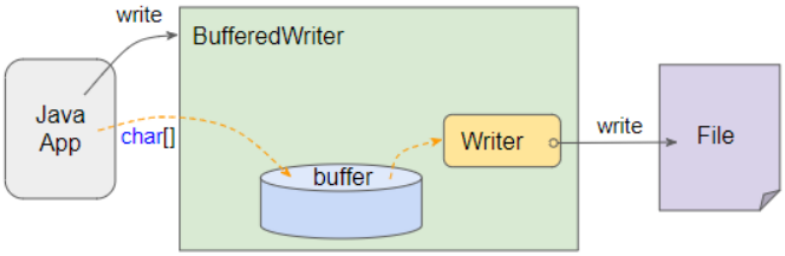
- Reading data directly from a source (e.g., a file) one character at a time can be slow. **BufferedReader** temporarily stores a chunk of data in an internal buffer with default size of 8192 during reading.
- This buffer allows reading larger portions of data at once, reducing the number of reads from the underlying source(File).
- Reading from the buffer is faster because it involves in-memory operations, which are quicker than reading from external sources.
- The combination of buffering, reduced read operations, and in-memory processing significantly improves reading speed.

```
Syntax:
// Creates a FileReader
      FileReader fr = new FileReader(file_name);
// Creates a BufferedReader
      BufferedReader buffer = new BufferedReader(fr);
```

Methods of BufferedReader:

- **read()** - reads a single character from the internal buffer of the reader
- **read(char[] array)** - reads the characters from the reader and stores in the specified array.
- **readline()** - reads a line of text

2. BufferedWriter: inherits from the Writer class and is used for writing text to a character-output stream, such as a file. It writes data to the output stream in larger chunks (a buffer), making the writing process more efficient. BufferedWriter wraps inside it a Writer object, which is responsible for writing data to the target (such as file).



- Writing data directly to the output stream can be slow due to frequent write operations.**BufferedWriter** temporarily stores data in an internal buffer before writing it to the output stream.
- This buffer allows writing larger portions of data at once, reducing the number of writes to the destination.
- Writing to the buffer is faster because it involves in-memory operations, which are quicker than writing directly to external destinations.
- The combination of buffering, reduced write operations, and in-memory processing significantly improves writing speed.

```
Syntax:
// Creates a FileWriter
     FileWriter fw = new FileWriter(File_name);
// Creates a BufferedWriter
BufferedWriter bufferedWriter = new BufferedWriter(fw);
```

Methods of BufferedWriter:

- **write(int c) -** Writes a single character to the internal buffer of the writer.
- **write(String str)** - Writes a string to the internal buffer of the writer.
- **newline() -** Writes a newline character(s) to the internal buffer of the writer.

Q.Explain about FileReader and BufferedWriter Class.

Q. A data dile "emp.txt" contains name, address and salary of 30 employees. WAP to display only those records who are from "Kathmandu".

```
import java.io.*;
class EmployeeExample{
     public static void main(String[] args) throws
     IOException, FileNotFoundException{
          FileReader fr = new FileReader("Emp.txt");
          BufferedReader br = new BufferedReader(fr);
          String str = br.readLine();
          while(str!=null) {
               if(str.contains("kathmandu")) {
                    System.out.println(str);
               }
               str = br.readLine();
          }
          br.close();
     }

}
```

Q. WAP to copy an mp3 file.(Using Buffer)

```
import java.io.*;
class Test{
     public static void main(String[] args) throws
Exception {
     FileInputStream fis = new FileInputStream("abc.mp3");
   BufferedInputStream bis = new BufferedInputStream(fis);
```

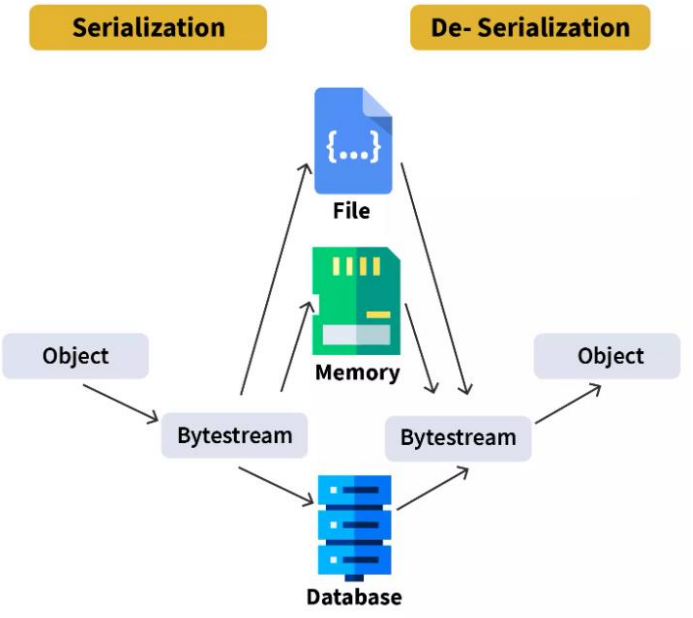
```
FileOutputStream fos = new FileOutputStream("xyz.mp3");
BufferedOutputStream bos = new BufferedOutputStream(fos);

        int data;
        System.out.println("Copying started.........");
        while((data=bis.read())!=-1) {
             bos.write(data);
        }
        System.out.println("Copied successfully.....");
    }
```

OBJECT SERIALIZATION & DESERIALIZATION

Object serialization is a process where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.

Need For Serialization: Serialization mechanism is usually used when there is a need to send the data (objects) over the network or to store in files. The hardware components like network infrastructure, hard disk etc understands only bytes and bits, not the java objects. So Serialization is used in this case which translates Java object's state to byte-stream to send it over the network or save it in file. Serialization can be used for securing sensitive data. We can encrypt the serialized byte stream before transmission or storage, providing a level of security to the data.

The reverse operation of serialization is called ***deserialization*** where byte-stream is converted into an object.



The byte stream which is created during Serialization is not dependent on platform so object Serialized on one platform can be Deserialized on other platform also.

Process of Serialization:

- The class of the object which we want to serialize must implement the **java.io.Serializable** interface.
- Create an **ObjectOutputStream** by providing an output stream where we want to write the serialized data.

  Example: for output stream file

```
FileOutputStream fos = new FileOutputStream("file.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

- The **writeObject()** method of the **ObjectOutputStream** is used to serialize the object. Pass the object which we want to serialize as an argument to this method.

```
oos.writeObject(Object_to_serialize);
```

- After writing the object, close the **ObjectOutputStream.**

```
oos.close();
```

Process of Deserialization:

- Create an **ObjectInputStream** by providing an input stream from where we want to read the serialized data.

  Example: for output stream file

  ```
  FileInputStream fis = new FileInputStream("file.txt");
  ObjectInputStream ois = new ObjectInputStream(fis);
  ```

- The **readObject()** method of the **ObjectInputStream** is used to deserialize the object. Cast the returned object to the appropriate class type.

  ```
  Class_name Object_to_serialize = (Class_name) ois.readObject();
  ```

- After reading the object, close the **ObjectInputStream.**

  ```
  ois.close();
  ```

During serialization and deserialization, handle exceptions such as ***IOException*** *and* ***ClassNotFoundException*** *that may occur.*

Example of Serialization and Deserialization:

Q. WAP to store object of a class Student into a file "student.dat" also read the objects from the same file and display the state of objects on the screen. Handle possible exception.

```
import java.io.*;
class Student implements Serializable {
    private String name;
    private int rollNumber;
    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }
```

```
    public void display() {
        System.out.println("Student Name: " + name + "\n
        Roll Number: " + rollNumber);
    }
}

public class FileExample {
    public static void main(String[] args)throws
     IOException, ClassNotFoundException {
        //Create FileOutputStream object
FileOutputStream fos = new FileOutputStream("student.dat");
     //Create ObjectOutputStream object
     ObjectOutputStream oos = new ObjectOutputStream(fos);
     //Create class object
     Student s = new Student("Ankit Kharel", 95);
     //serialize the object
     oos.writeObject(s);
     //close ObjectOutputStream
     oos.close();
     System.out.println("Serialization Completed!!!!");

     //create FileInputStream object
 FileInputStream fis = new FileInputStream("student.dat");
     //Create ObjectInputStream object
     ObjectInputStream ois = new  ObjectInputStream(fis);
    // Deserialize and read the Student object from the file
     Student std = (Student) ois.readObject();
     //close ObjectInputStream
```

```
        ois.close();
        System.out.println("After Deserialization, the content
        of object are:");
        std.display();
       }
}
```

```
Serialization Completed!!!!
After Deserialization, the content
Student Name: Ankit Kharel
 Roll Number: 95
```

ZIP FILE STREAM

ZIP is a common file format that's used to compress one or more files together into a single location. This reduces file size and makes it easier to transport or store. A recipient can unzip (or extract) a ZIP file after transport and use the file in the original format.

The necessary classes and interfaces that are required to create a zip file are defined in **java.util.zip** package.

java.util.zip Package Important Classes	
ZipFile	Reads a ZIP file.
ZipEntry	Represents a ZIP file entry.
ZipInputStream	Reads the files in ZIP file format.
ZipOutputStream	Writes the files in ZIP file format.
GZIPInputStream	Reads the files in GZIP file format.
GZIPOutputStream	Writes the files in GZIP file format.

Steps to zip a file:

Step 1: Open the File to be ZIP.

```
FileInputStream fis = new FileInputStream("sample.txt");
```

Step 2: Create a ZIP Output File

```
FileOutputStream fos = new FileOutputStream("zipsample.zip");
```

Step 3: Create a ZipOutputStream

```
ZipOutputStream zos = new ZipOutputStream(fos);
```

Step 4: Create a ZipEntry

```
ZipEntry ze = new ZipEntry("sample.txt");
```

Step 5: Put the ZipEntry into the ZipOutputStream

```
zos.putNextEntry(ze)
```

Step 6: Write Data from FileInputStream to ZipOutputStream

```
zos.write()
```

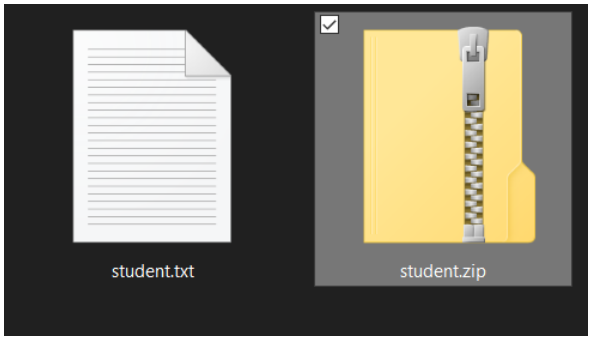
Step 7: Close the ZipEntry , FileInputStream and ZipOutputStream

```
zos.closeEntry(), fis.close(), zos.close()
```

Q. WAP to copy file into zip.

```
import java.io.*;
import java.util.zip.*;
public class ZipFileExample {
    public static void main(String[] args) {
        try {
            // Step 1: Open the "student.txt" File
  FileInputStream fis = new FileInputStream("student.txt");

         // Step 2: Create a ZIP Output File
FileOutputStream fos = new FileOutputStream("student.zip");

         // Step 3: Create a ZipOutputStream
        ZipOutputStream zos = new ZipOutputStream(fos);

         // Step 4: Create a ZipEntry
        ZipEntry ze = new ZipEntry("student.txt");

    // Step 5: Put the ZipEntry into the ZipOutputStream
            zos.putNextEntry(ze);

// Step 6: Write Data from FileInputStream to ZipOutputStream
            int bytesRead;
            while ((bytesRead = fis.read()) != -1) {
                zos.write(bytesRead);
            }
```

```
    // Step 7: Close the ZipEntry, FileInputStream, ZipOutputStream
            zos.closeEntry();
            fis.close();
            zos.close();
          System.out.println("File zipped successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



REFLECTION

Reflection is an API that is used to examine or modify the behavior of methods, classes, and interfaces at runtime. The required classes for reflection are provided under **java.lang.reflect** package which is essential in order to understand reflection. Here are some of the key classes and interfaces from the **java.lang.reflect** package:

Class class: It provides methods to examine class-level information, such as the class name, package, superclass, implemented interfaces, and more.

Constructor class: It allows us to create new instances of a class by invoking constructors with specified arguments.

Method class: It provides methods to examine and invoke methods at runtime.

Field class:: It allows us to get and set the values of fields at runtime.

Modifier class: Provides constants and static methods for examining and manipulating the access modifiers and other modifiers of classes, methods, and fields.

Reflection regarded as a powerful aspect in java because of following (*Need for reflction/Advantages of Reflection*):

- Reflection gives us information about the class to which an object belongs and also the methods of that class that can be executed by using the object.
- Reflection can access and modify private fields and methods of classes, even though they are not normally accessible from outside the class.
- Reflection allows us to load and create instances of classes at runtime without knowing their names in advance.

```
import java.lang.reflect.*;

class Student {
    private String name;
    private int rollNumber;

    public Student() {
     System.out.println("I am Student");
    }

    public Student(String name, int rollNumber) {
        this.name = name;
        this.rollNumber = rollNumber;
    }
    public void display() {
        System.out.println("Student Name: " + name + "\n
        Roll Number: " + rollNumber);
    }
}
public class Test {
         public static void main(String[] args) throws
Exception {
              Class c = Class.forName("Student");
              System.out.println("Constructors are:");
         Constructor cons[] = c.getDeclaredConstructors();
              for (Constructor i : cons) {
                   System.out.println(i);
              }
```

```
            System.out.println("Fields are:");
            Field fld[] = c.getDeclaredFields();
            for(Field i:fld) {
                System.out.println(i);
            }
            System.out.println("Methods are:");
            Method mth[]= c.getDeclaredMethods();
            for(Method i:mth) {
                System.out.println(i);
            }
        }
}
```

```
Constructors are:
public Student()
public Student(java.lang.String,int)
Fields are:
private java.lang.String Student.name
private int Student.rollNumber
Methods are:
public void Student.display()
```

Accessing private method of class using reflection

Step 1 − Find the class that contains the private method using its name. Use **Class.forName()** to obtain a **Class** object that represents the class containing the private method.

```
Class  c = Class.forName("FullyQualifiedClassName");
```

Step 2 − Find the specific private method we want to use inside that class. Use the **getDeclaredMethod()** method on the **Class** object to obtain a **Method** object representing the private method.

```
Method m = c.getDeclaredMethod("methodName", parameters);
```

Step 3 − Give permission to access the private method. Set **setAccessible(true)** on the **Method** object to make the private method accessible

```
m.setAccessible(true);
```

Step 4 – Create an instance of the class represented by Class object c.

```
Class_Name obj = (Class_Name)c.newInstance();
```

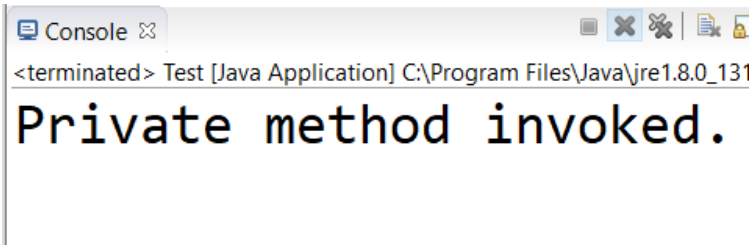
Step 5 − Perform the method's actions. Use the **invoke()** method on the **Method** object to invoke the private method

```
m.invoke(obj, arg1, arg2, ...);
```

Q. WAP that does reflective operation of accesing a private method of a class. (Why Refeltion is a powerful tool used in softeare testing and debugging?Elaborate with a small example)

Reflection allows testers and debuggers to access and modify private fields, methods, and constructors of classes. This can be valuable for testing private methods and changing internal states for debugging purposes.

```
import java.lang.reflect.*;
class MyClass {
    private void privateMethod() {
        System.out.println("Private method invoked.");
    }
}
 class test {
    public static void main(String[] args) throws Exception
{
        // Step 1: Obtain the Class Object
        Class c = Class.forName("MyClass");

        // Step 2: Obtain the Method Object
        Method m = c.getDeclaredMethod("privateMethod");

        // Step 3: Set Method Accessibility
        m.setAccessible(true);

         // Step 4: Create an instance
        Object obj = c.newInstance();

        // Step 5: Invoke the Method
         m.invoke(obj);
    }
}
```



Q. Write a program in Java to access a constructor, a private field and a private method using Reflection API. (Why is reflection regarded as a powerful aspect in java? WAP to demonstrate your answer)

```
import java.lang.reflect.*;
class MyClass {
     private String str = "I am private String Filed.";

     private MyClass() {
          System.out.println("I am private Constructor.");
     }

    private void privateMethod() {
        System.out.println("I am private method.");
    }
}

 class test {
    public static void main(String[] args) throws Exception
{
        // Step 1.1: Obtain the Class Object
        Class c = Class.forName("MyClass");
        // Step 1.2: Obtain the Construtor Object
        Constructor cons = c.getDeclaredConstructor();
        // Step 1.3: Obtained the Method object
        Method m = c.getDeclaredMethod("privateMethod");
        // Step 1.4: Obtained the Field object
        Field f = c.getDeclaredField("str");
```

```
        // Step 2.1: Set Constructor Accessiblity
        cons.setAccessible(true);
        // Step 2.2: Set Method Accessibility
        m.setAccessible(true);
        // Step 2.3: Set Filed Accessibility
        f.setAccessible(true);

    // Step 3: Create an Instance Using private construtor
        MyClass obj = (MyClass)cons.newInstance();

        // Step 4.1: Access Private Field
        String s = (String)f.get(obj);
        //The return type of this method is Object, so we
          may need to cast it to the appropriate type.
        System.out.println(s);
        // Step 4.2 Access Private Method
        m.invoke(obj);
    }
}
```

Disadvantages of Using Reflection

- Reflection can bypass access modifiers, allowing access to private members of classes breaking Encapsulation.
- The absence of compile-time type checking can lead to runtime errors that are difficult to diagnose and fix.
- Reflection operations can be significantly slower than their non-reflective counterparts.

CHAPTER: 4

EVENTS, EVENTS HANDLING, AWT/SWINGS

BASICS OF EVENTS HANDLING

EVENT

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.

TYPES OF EVENTS

Foreground Events: They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse

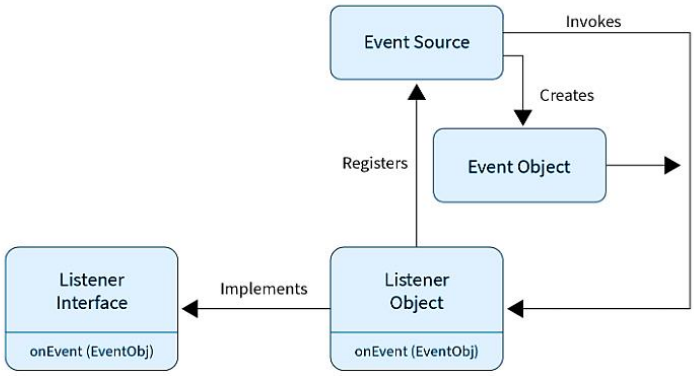
Background Event: Those events that require the interaction of end user are known as background events. Operating system interrupts, hardware or software failure, timer expires.

EVENT HANDLING MECHANISM

Q. What is event? Briefly explain the model available for event handling.

Q. How is event handled in java?

Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have 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.



- **Event Source** - Events are generated from the source. There are various sources like buttons, checkboxes, list, menu-item, choice, scrollbar, text components, windows, etc., to generate events.
- **Event Object** - When an event occurs, an event object is created. The event object contains information about the event, such as the type of event and the source of the event.
- **Event Listener interface** - A listener interface is defined for each type of event. It declares the methods that must be implemented by any class that wants to handle the corresponding event.
- **Event Listener object** - It registers with event source to receive events and implements methods in event listener interface.

JAVA EVENT CLASSES & LISTENER INTERFACES

Q. Explain event handling with various type of event available in java.

[1] Name_of_Event : *ActionEvent*

Source: *Buttons, list, meanuIteam, Text filed*

Description: *When an action, like a button click, occurs.*

Listener Interface: *ActionListener*

Methods: *actionPerformed(ActionEvent e)*

[2] Name_of_Event : *MouseEvent*

Source: *Mouse-related events*

Description: *Generated on mouse actions like clicks and movements.*

Listener Interface: *MouseListener*

Methods: *mouseClicked(MouseEvent e): Called when a mouse button is clicked.*
mousePressed(MouseEvent e): Invoked when a mouse button is pressed.
mouseReleased(MouseEvent e): Triggered when a mouse button is released.
mouseEntered(MouseEvent e): Generated when the mouse enters a component.
mouseExited(MouseEvent e): Generated when the mouse exits a component.

[3] Name_of_Event : *KeyEvent*

Source: *Keyboard*

Description: *Generated when a key is pressed or released*

Listener Interface: *KeyListener*

Methods: *keyPressed(KeyEvent e): Invoked when a key is pressed.*
keyReleased(KeyEvent e): Triggered when a key is released.
keyTyped(KeyEvent e): Generated when a key press results in a character.

[4] Name_of_Event : *ItemEvent*

Source: *Checkbox, choice, list*

Description: *Generated when an item's state changes.*

Listener Interface: *ItemListener*

Methods: *temStateChanged(ItemEvent e): Called when an item's state changes.*

[5] Name_of_Event : *WindowEvent*

Source: *Windows or Frame*

Description: *Generated when a window's state changes, such as when it is opened, closed*

Listener Interface: *WindowListener*

Methods: *windowClosed(WindowEvent e): Called after the window has been closed.*

windowActivated(WindowEvent e): called when the Window is set to be an active Window.

windowDeactivated(WindowEvent e): called when a Window is not an active Window anymore.

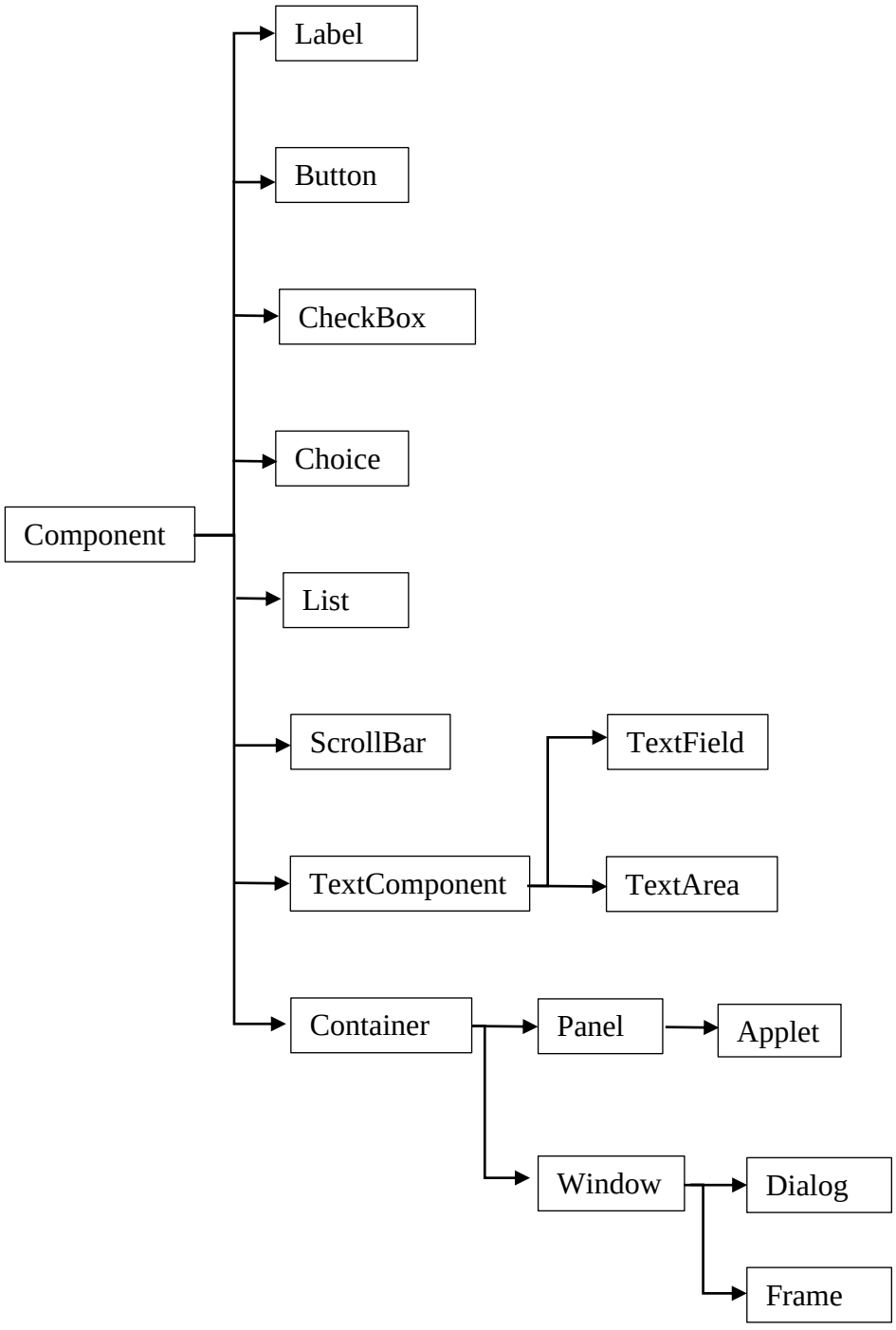
windowIconified(WindowEvent e): Called when the window is minimized.

windowDeiconified(WindowEvent e): Generated when the window is restored

JAVA AWT HIERARCHY

Abstract Window Toolkit (AWT) in Java provides a set of classes and components for creating graphical user interfaces (GUIs).(Package= java.awt)

The hierarchy of Java AWT classes are given below.



Component

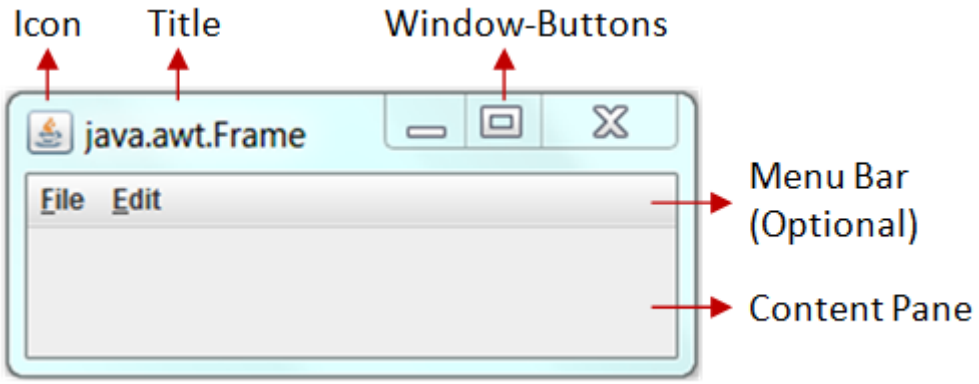
All the elements like the button, text fields, scroll bars, etc. are called components. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

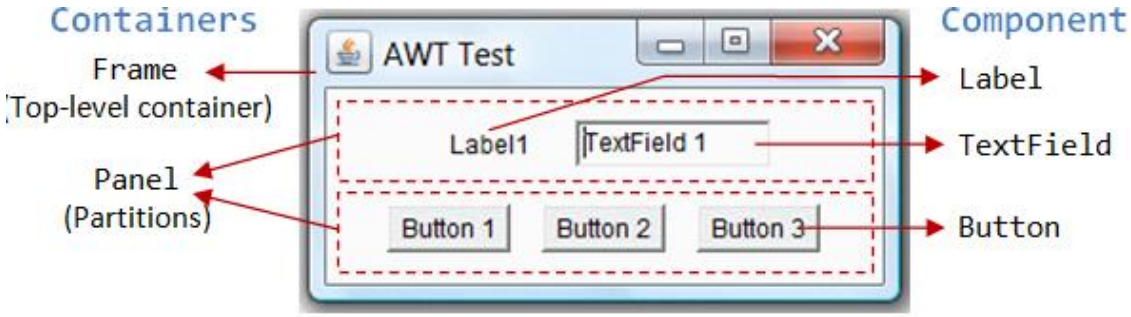
The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components. The commonly-used top-level containers in AWT are Frame, Dialog , Panel and Applet

(a) Frame: A ***Frame*** provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area. Frame is most widely used container while developing an AWT application.

The content pane is a special container within the **Frame** that holds the components we add to the frame.



(b) Panel: A **Panel** is a rectangular area used to group related GUI components in a certain layout.The ***Panel*** is the container that doesn't contain title bar, border or menu bar. It can have other components like button, text field etc.



Q. Difference between frame and panel

FRAME	**PANEL**
Top-level container	Container typically used within other containers
Can have a title bar with a title	Doesn't have a title bar
Can set its size and position on the screen	Takes the size provided by its parent container
Supports window closing operations (e.g., close button)	Doesn't have built-in window close operations
Extends **java.awt.Frame**	Extends **java.awt.Panel**
Frame uses BorderLayout as default layout manager	Panel uses FlowLayout as default layout manager
Frame frame = new Frame();	Panel panel = new Panel();

EVENT HANDLING CODE

Some Tips For Event Handling:

1. Extend Frame and implement the required ListenerClass.

2. Create the required Components.

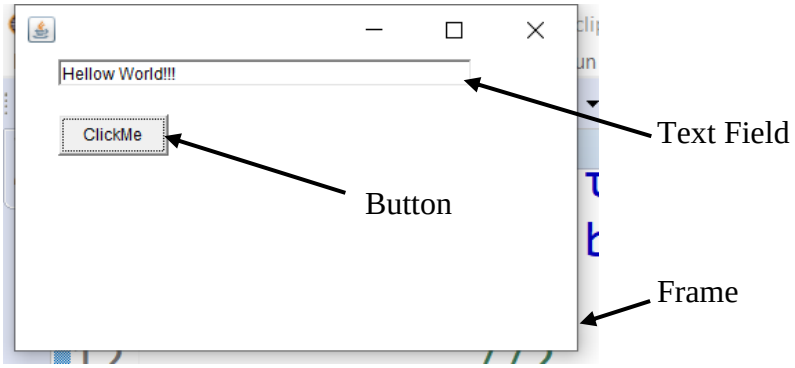
3. Set the layout of the Frame.

4. Add components to the Frame.

5. Add the listener to the components.

6. Handle the event in the implemented function of listener class.

Example: Handling *ActionEvent*. When the user clicks the botton name "ClickMe" then, "Hellow World " should be displayed in the Text Field.




```
import java.awt.*;
import java.awt.event.*;
public class EventHandlingWithinClass extends Frame
implements ActionListener {
        TextField tf;
        Button bt;
```

```
public EventHandlingWithinClass() {

        //1. Create Components
            tf = new TextField();
            bt = new Button("ClickMe");
        //2. Set Layout of Frame
            setLayout(null);
/* it means that we are taking manual control over the positioning and sizing of the components within that container.*/
            setSize(100, 100);
/*setSize(width, height) is used to set the size of the Frame (window)*/
            setVisible(true);
/*It means the window becomes visible and is displayed on the screen.)*/

        //3. Add components to the Frame
            tf.setBounds(40, 40, 300, 20);
            bt.setBounds(40, 80, 80, 30);
/*setBounds(int x, int y, int width, int height) method is used in Java's Abstract Window Toolkit (AWT) and Swing libraries to specify the position and size of a GUI component, such as a Button, TextField, or Label*/
            add(tf);
            add(bt);
/*used to add graphical components (such as buttons, text fields, labels, etc.) to a container*/

        //4. Register Listener
            bt.addActionListener(this);

    }
```

```
        //4. Handle the event
        public void actionPerformed(ActionEvent e) {
//e.getSource() returns the object that triggered the event
            if(e.getSource() == bt) {
                tf.setText("Hellow World!!!");
            }
        }
        public static void main(String[] args) {
            new EventHandlingWithinClass();
        }
}
```

Q. How to close AWT. Explain all methods with example.

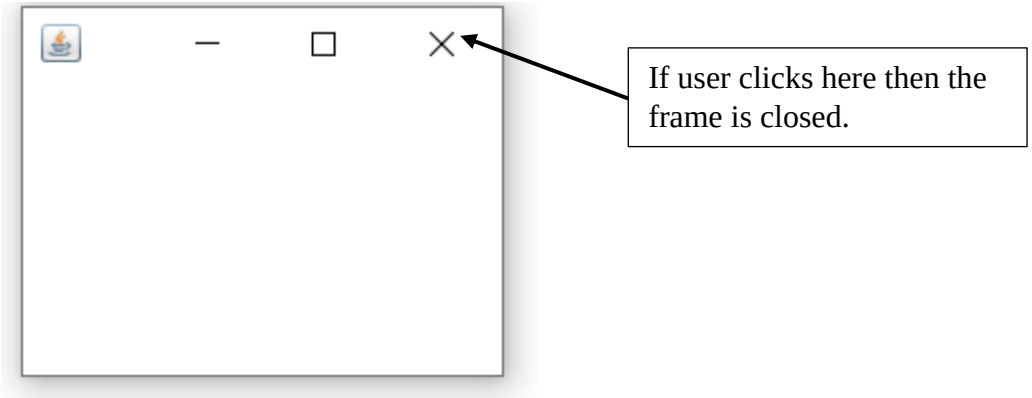
We can close the AWT window or Frame by overriding ***windowClosing()*** method found in WindowAdapter class and calling ***dispose()*** or ***System.exit()*** inside windowClosing() method.

- ***dispose():*** It is used to close individual windows or frames. It does not terminate the entire application; it only closes the specific window or frame on which it is called.

- ***System.exit():*** I it terminates the entire Java application and exits the JVM (Java Virtual Machine). It is used when you want to forcefully exit the entire application, closing all windows and ending the program's execution.

```
import java.awt.*;
import java.awt.event.*;
public class AWT_CloseExample extends WindowAdapter{

    //Create Frame oject
    Frame f;
```

```
    AWT_CloseExample() {
        //Create frame and set layout
        f=new Frame();
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        f.addWindowListener(this);

        //add the Window listener to the frame
        f.addWindowListener(this);
    }
    public void windowClosing(WindowEvent e) {
        f.dispose();
    }
    public static void main(String[] args) {
        new AWT_CloseExample();
    }

}
```



JAVA SWING

AWT (Abstract Window Toolkit) is a Java library used to create graphical user interfaces (GUIs). However, it has some issues. AWT components rely on native methods, which can cause problems with portability, making the GUIs look different on different operating systems (e.g., Windows and macOS). Additionally, AWT components are resource-intensive, consuming more memory and processor time.

Due to this, Java developers decided to redevelop AWT package without internally taking the help of native methods. Hence all the classes of AWT are extended to form new classes and a new class library is created. This library is called JFC (Java Foundation Classes).

JFC represents class library developed in pure Java which is an extension to AWT and swing is one package in JFC, which helps to develop GUIs and the name of the package is:

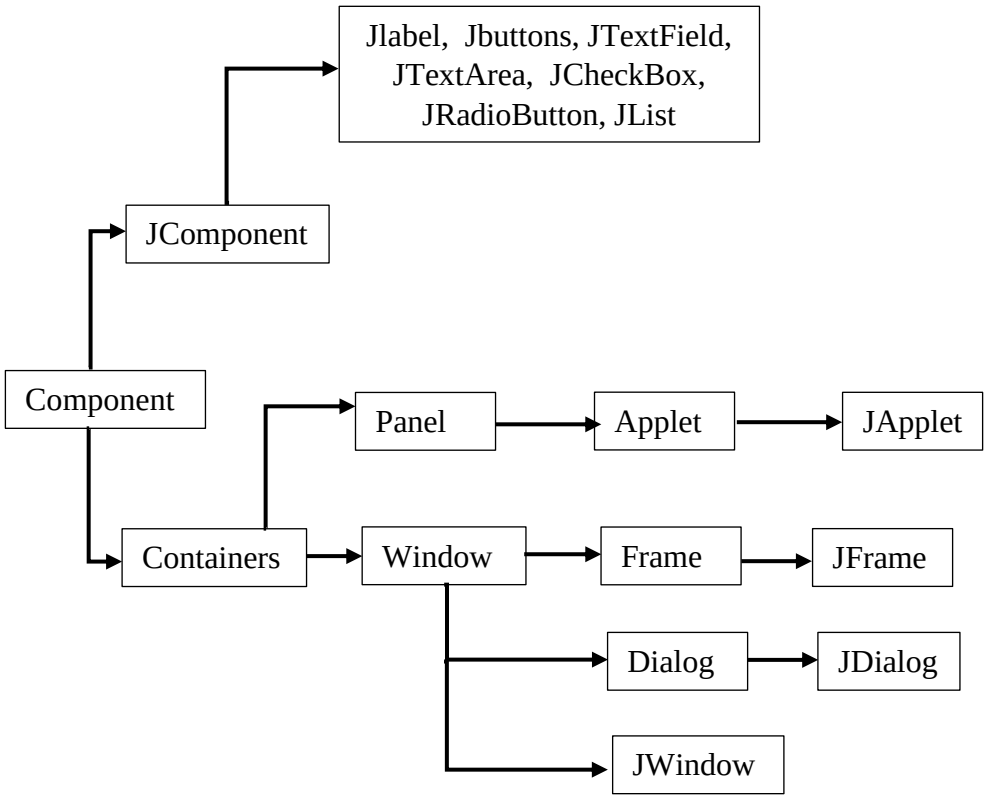
```
import javax.swing.*;
```

Different between AWT and SWING

AWT	SWING
AWT components are platform-dependent.	Java swing components are platform independent.
AWT components are heavyweight	Swing components are lightweight
AWT doesn't support pluggable look and feel	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc
AWT doesn't follows MVC where model represents data, view represents presentation and controller acts as an interface between model and view	Swing follows MVC

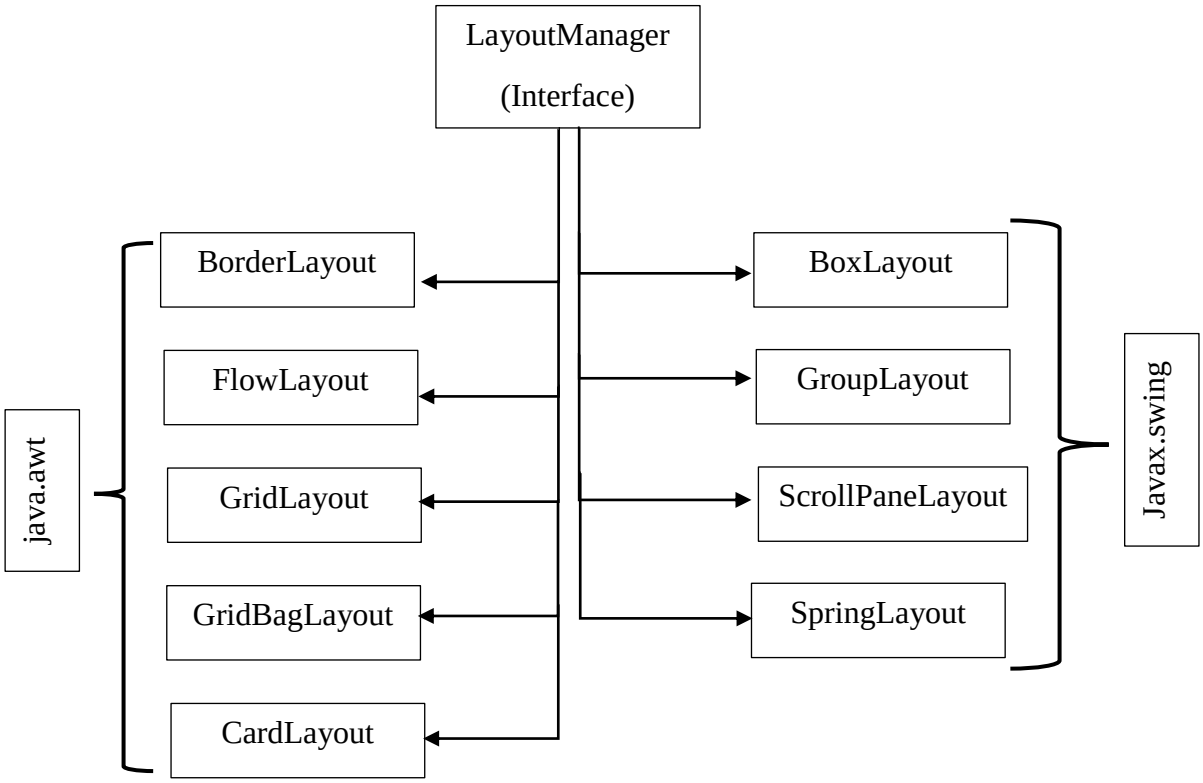
HIERARCHY OF JAVA SWING

The hierarchy of java swing API is given below.



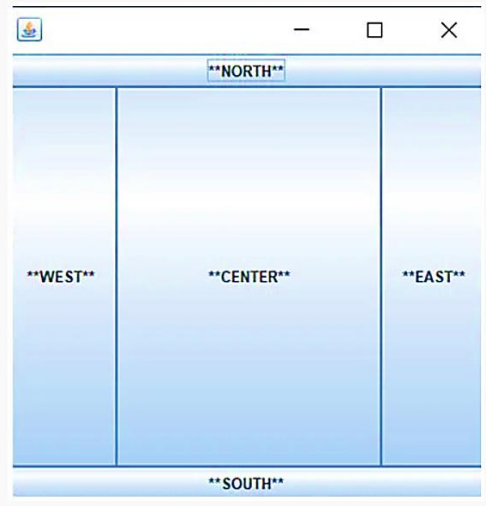
JAVA LAYOUT MANAGER

The Layout managers are used to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers. It is an interface that is implemented by all classes of layout managers.



Each container (Frame, JPanel , JFrame) in Java has a layout manager. A layout manager is an object created from a class that implements the LayoutManager interface.

1. Border Layout



BorderLayout is used, when we want to arrange the components in five regions. The five regions can be north, south, east, west and the centre. We can add components to these regions using the **add()** method. When components are added to these regions, the layout manager automatically positions and resizes them to fit within their respective regions. It is the default layout of a frame or window.

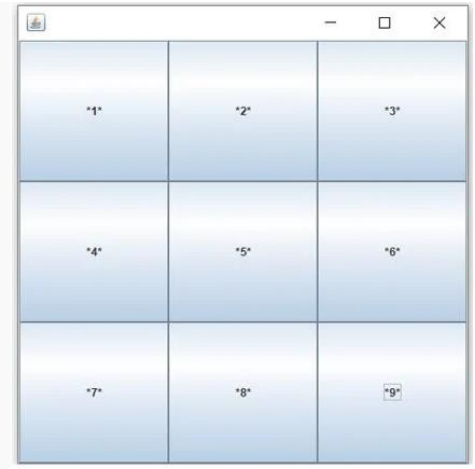
The BorderLayout provides five constants for each region:

- **public static final int NORTH**
- **public static final int SOUTH**
- **public static final int EAST**
- **public static final int WEST**
- **public static final int CENTER**

```
import java.awt.*;
import javax.swing.*;

public class BorderDemo1 extends JFrame {

    JButton box1, box2, box3, box4, box5;
```

```
    BorderDemo1()
    {
              //create buttons
               box1=new JButton("**NORTH**");
               box2=new JButton("**SOUTH**");
               box3=new JButton("**EAST**");
               box4=new JButton("**WEST**");
               box5=new JButton("**CENTER**");

               //set layout
               setLayout(new BorderLayout());
               setSize(400,400);
               setVisible(true);

               //add buttons
              add(box1, BorderLayout.NORTH);
               add(box2, BorderLayout.SOUTH);
               add(box3, BorderLayout.EAST);
               add(box4,BorderLayout.WEST);
               add(box5,BorderLayout.CENTER);



    }
    public static void main(String[] args)
    {
        new BorderDemo1();
    }
}
```

2. Grid Layout

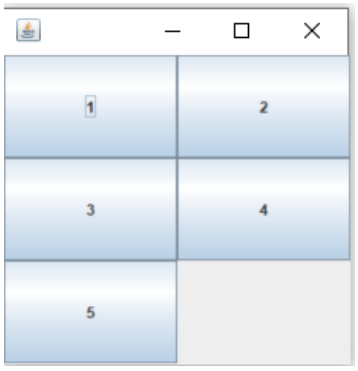


GridLayout is a layout manager divides a container into a grid of rows and columns, where each cell in the grid can contain a component. It ensures that all cells have the same size, making it suitable for creating uniform grids of components. Constructors of GridLayout class are:

- **GridLayout():** creates a grid layout with one column per component in a row.
- **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
- **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutExample extends JFrame
{
    JButton btn1, btn2, btn3, btn4, btn5;

    GridLayoutExample()
    {
```

```
        //create buttons
        btn1 = new JButton("1");
        btn2 = new JButton("2");
        btn3 = new JButton("3");
        btn4 = new JButton("4");
        btn5 = new JButton("5");

        //add buttons
        add(btn1);
        add(btn2);
        add(btn3);
        add(btn4);
        add(btn5);

    //set Layout by calling any one of the 3 Constructor
        setLayout(new GridLayout(3,2));
        setSize(300, 300);
        setVisible(true);
    }
    public static void main(String args[])   {
        new GridLayoutExample();
    }
}
```



3. Flow Layout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). When the row is filled, components are automatically flowed to the next row. It is the default layout of the applet or panel.

Fields of FlowLayout class are:

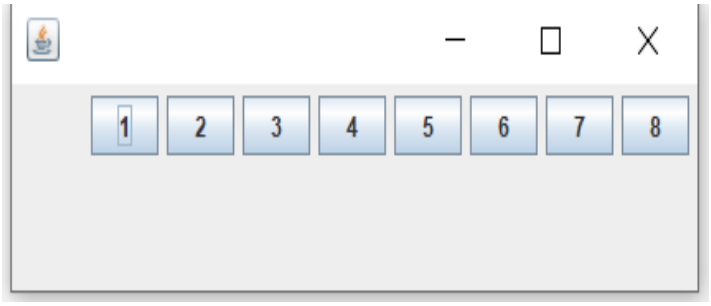
- public static final int LEFT (used to left-align components within each row)
- public static final int RIGHT (is used to right-align components within each row)
- public static final int CENTER (used to center-align components horizontally within each row)
- public static final int LEADING
- public static final int TRAILING

Constructors of FlowLayout class

- **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
- **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
- **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

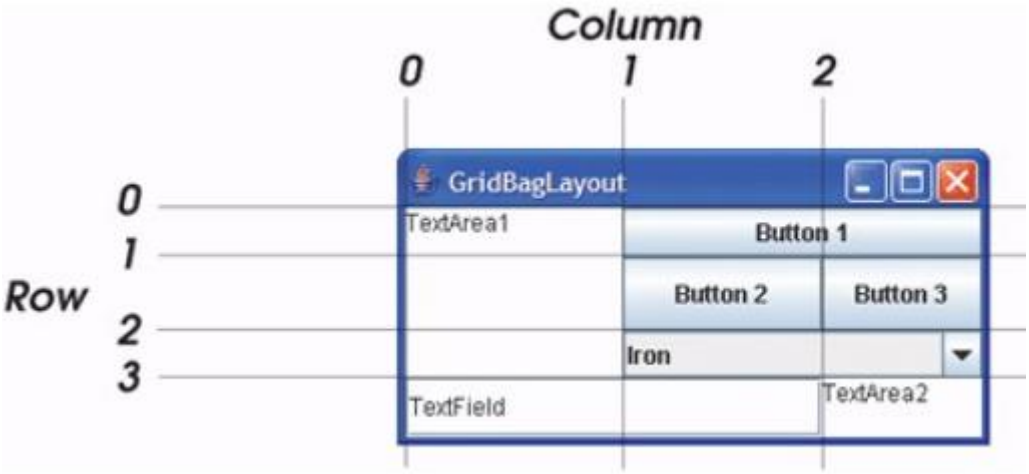
```
import java.awt.*;
import javax.swing.*;
public class FlowLayoutExample extends JFrame
{
     JButton btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8;
     FlowLayoutExample()
     {
          //Create Buttons
          btn1 = new JButton("1");
          btn2 = new JButton("2");
          btn3 = new JButton("3");
          btn4 = new JButton("4");
          btn5 = new JButton("5");
          btn6 = new JButton("6");
          btn7 = new JButton("7");
          btn8 = new JButton("8");

          //Add Buttons
          add(btn1); add(btn2);
          add(btn3); add(btn4);
          add(btn5);  add(btn6);
          add(btn7);  add(btn8);

     //set Layout by calling any one of the 3 Constructor
          setLayout(new FlowLayout(FlowLayout.RIGHT));
          setSize(300, 300);
          setVisible(true);
     }
```

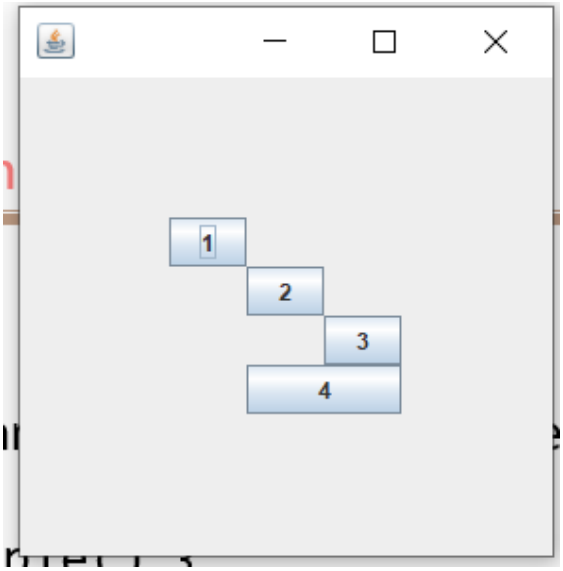
```
    public static void main(String args[]){
            new FlowLayoutExample();
    }
}
```



4. GridBag Layout

GridBagLayout organizes components in a grid of cells, where each cell can have different sizes and properties. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of the constraints object, we arrange the component's display area on the grid.GridBagLayout is particularly useful when you need to create forms, dialogs, or other complex layouts where components may have varying sizes and need to be aligned in different ways.






```
import java.awt.*;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame{
      JButton btn1, btn2, btn3, btn4;
      GridBagConstraints gbc;

     public GridBagLayoutExample() {

          // Create Buttons
          btn1 = new JButton("1");
          btn2 = new JButton("2");
          btn3 = new JButton("3");
          btn4 = new JButton("4");


          // First Set Layout and arrange components
          setLayout(new GridBagLayout());
          setSize(300, 300);
          setVisible(true);
          // Create object to arrange component in grid
          gbc = new GridBagConstraints();
```

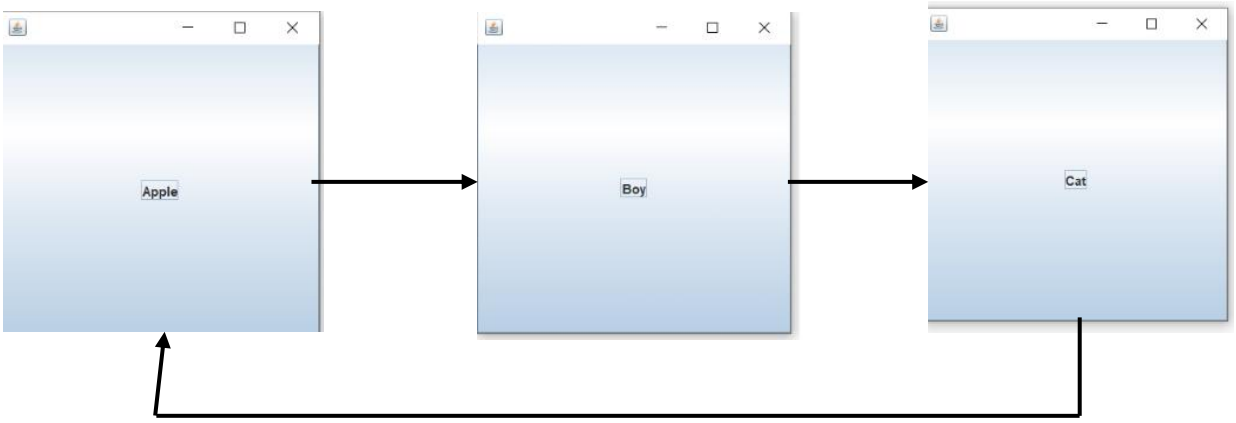
```
        // Add Button 1 (first clomun first row)
        gbc.gridx = 0;
        gbc.gridy = 0;
        add(btn1, gbc);
        // Add Button 2 (Second column Second row)
        gbc.gridx = 1;
        gbc.gridy = 1;
        add(btn2, gbc);

        // Add Button 3 (third column third row)
        gbc.gridx = 2;
        gbc.gridy = 2;
        add(btn3, gbc);

        // Add Button 4 (second column fourth row)
        gbc.gridx = 1;
        gbc.gridy = 3;
    // Set the button to occupy 2 grid(2nd and 3rd column)
        gbc.gridwidth = 2;
    // Stretch the width of button to occupy 2 grid
        gbc.fill = GridBagConstraints.HORIZONTAL;
        add(btn4, gbc);

    }
       public static void main(String[] args) {
           new GridBagLayoutExample();
        }

}
```

5. Card Layout

The Java CardLayout class manages the components in such a manner that only one component is visible at a time as if they were cards in a deck of cards.

Constructors of CardLayout Class

- **CardLayout():** creates a card layout with zero horizontal and vertical gap.



- **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

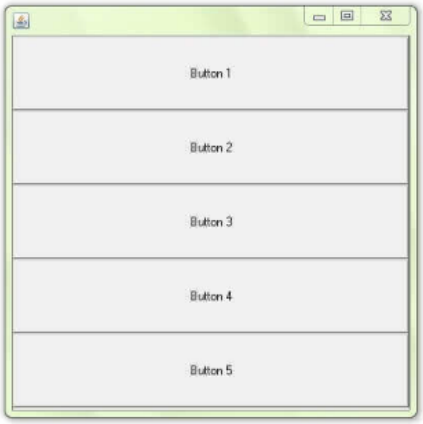
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements
ActionListener{
     CardLayout c_Card; //We need to create object because we
                          have to call the method next();
     JButton b1,b2,b3;
     public CardLayoutExample() {
          // Create Buttons
           b1 = new JButton("Apple");
           b2 = new JButton("Boy");
           b3 = new JButton("Cat");

          // Add event listener
          b1.addActionListener(this);
          b2.addActionListener(this);
          b3.addActionListener(this);

          // Add Buttons
          add(b1);
          add(b2);
          add(b3);

          // Set Layout
          c_Card=new CardLayout();
          setLayout(c_Card);
          setSize(300,300);
          setVisible(true);
     }
```

```
    @Override
    public void actionPerformed(ActionEvent e) {
    /* switch to the next card within the content pane of
    the JFrame */
              c_Card.next(getContentPane());
    }
    public static void main(String[] args) {
         new CardLayoutExample();
    }
}
```

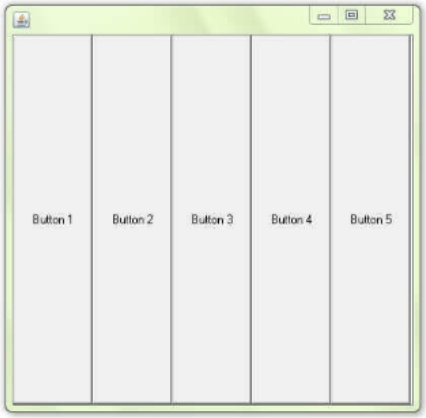
6. Box Layout

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:

- **public static final int X_AXIS:** Alignment of the components are horizontal from left to right.
- **public static final int Y_AXIS:** Alignment of the components are vertical from top to bottom.
- **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
- **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.



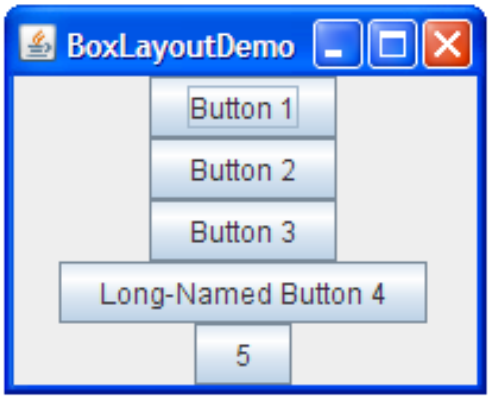
Example of BoxLayout class with X-AXIS



Example of BoxLayout Class with LINE_AXIS

Constructor of BoxLayout class

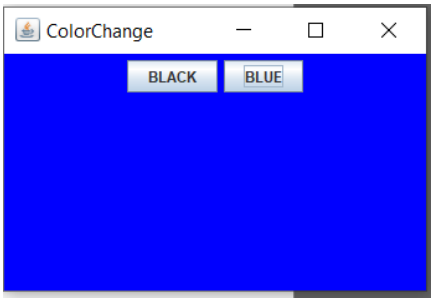
- **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.




```
import java.awt.*;
import javax.swing.*;
public class BoxLayoutExample extends JFrame{
      JButton b1, b2, b3, b4, b5;
         public BoxLayoutExample() {

            // Create Buttons
             b1 = new JButton("Button_1");
             b2 = new JButton("Button_2");
             b3 = new JButton("Button_3");
```

```
            b4 = new JButton("Long Named Button_4");
            b5 = new JButton("5");


            // Add Buttons
            add(b1);
            add(b2);
            add(b3);
            add(b4);
            add(b5);


           //set Alignment
            b1.setAlignmentX(CENTER_ALIGNMENT);
            b2.setAlignmentX(CENTER_ALIGNMENT);
            b3.setAlignmentX(CENTER_ALIGNMENT);
            b4.setAlignmentX(CENTER_ALIGNMENT);
            b5.setAlignmentX(CENTER_ALIGNMENT);


              //set Layout
setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));
            setSize(300, 300);
            setVisible(true);


        }
    public static void main(String[] args) {
         new BoxLayoutExample();
    }
}
```

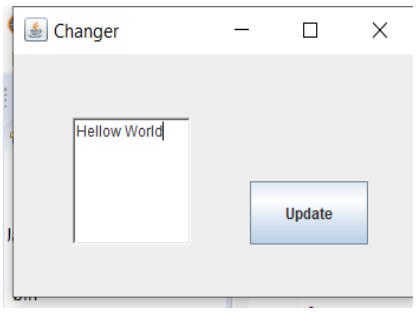
Q. Write a program to generate a Frame with two buttons "BLACK" and "BLUE". When a button is clicked it should change the background color of the frame to respective color.

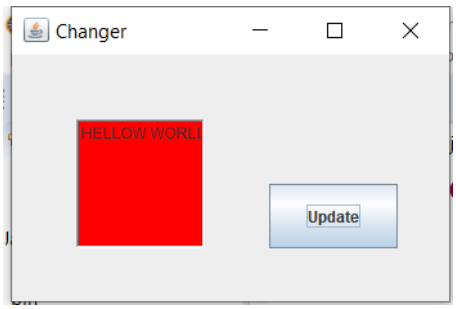



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class BackgroundColorChangeApp extends JFrame
implements ActionListener {
    //Create Component Class object
     JButton bk;
     JButton bl;
     public BackgroundColorChangeApp() {
     //Create Components
          bk = new JButton("BLACK");
          bl = new JButton("BLUE");
     //Set Frame layout
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setLayout(new FlowLayout());
          setVisible(true);
          setSize(400, 400);
          setTitle("ColorChange");
```

```
        //add listener
        bl.addActionListener(this);
        bk.addActionListener(this);

    //add components to the frame
        add(bk);
        add(bl);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        //find source
        if(e.getSource()==bk) {
            //set background to black
            getContentPane().setBackground(Color.BLACK);
        }if(e.getSource()==bl) {
            getContentPane().setBackground(Color.BLUE);
        }

    }
    public static void main(String[] args) {
        new BackgroundColorChangeApp();
    }
}
```

Q. Create a Frame with one button and one TextField. When the user clicks on the button the text entered should be changed to uppercase and color of text field must be changed.

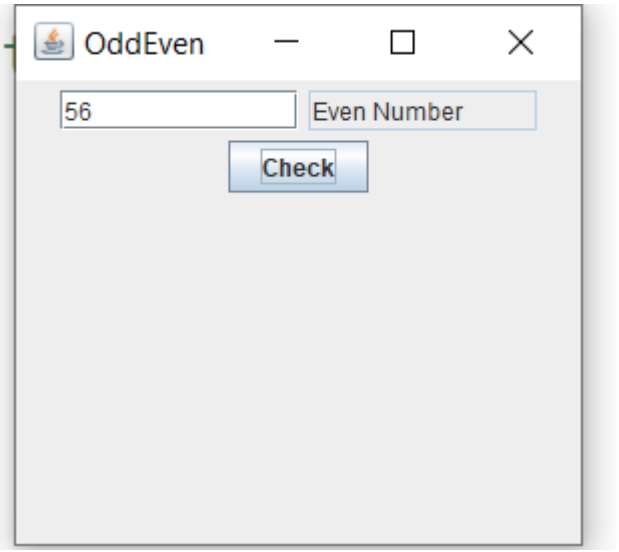





```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class UpperCaseExample extends JFrame implements
ActionListener  {
     //Create component class objects
          JTextField tf;
          JButton bt;
     public UpperCaseExample() {
     //Create Components
          tf = new JTextField("Enter Some Text");
          bt = new JButton("Update");
     //Set Frame Layout and components layout
          setLayout(new FlowLayout());
          setTitle("Changer");
          setSize(500, 500);
          setVisible(true);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
        //Register Event Listener
            bt.addActionListener(this);
        //add components to the frame
            add(tf);
            add(bt);
        }
        public void actionPerformed(ActionEvent e) {
            //find the source
            if(e.getSource()==bt){
                 // Get the text from the text field
                String str = tf.getText();
                 // Convert the text to UpperCase
                String up_str = str.toUpperCase();
            // Set the UpperCase text back to the text field
                tf.setText(up_str);
                // Change the text field's background color
                tf.setBackground(Color.red);
            }

        }
        public static void main(String[] args) {
            new UpperCaseExample();
        }

}
```

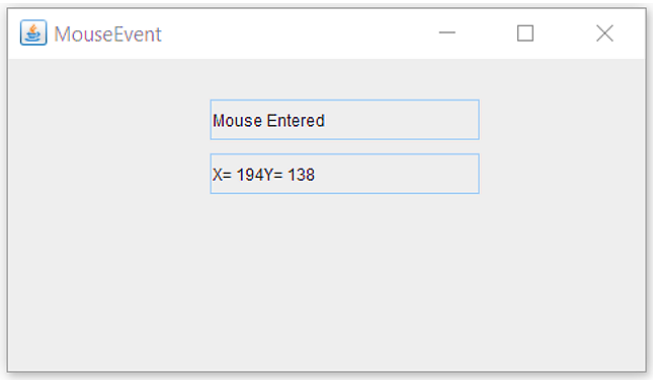
Q. Create a swing GUI applications that contains a button and two text fileds. When the button is clicked the value of first text filed should be checked and display "Odd Number' or "Even Number" in second text field. Also delegate the GUI and event handling section into separate classes.




```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class OddEven extends JFrame implements
ActionListener {
     //create component class objects
          JTextField tf1, tf2;
          JButton bt;
     public OddEven(){
     //Create Components
          tf1 = new JTextField("Enter a Number here");
          tf2 = new JTextField();
          tf2.setColumns(10);
          bt = new JButton("Check");
          tf2.setEditable(false);
```

```
        //Set Frame layout
            setLayout(new FlowLayout());
            setVisible(true);
            setTitle("OddEven");
        //Register Listener
            bt.addActionListener(this);
        //Add Components to the frame
            add(tf1); add(tf2); add(bt);
        }
        @Override
        public void actionPerformed(ActionEvent e) {
            // find the source
            if(e.getSource() == bt) {
                try {
                    //get value from the tf1
                    int n = Integer.parseInt(tf1.getText());
                    if(n%2==0) {
                        tf2.setText("Even Number");
                    }else {
                        tf2.setText("Odd Number");
                    }
                }catch(NumberFormatException n) {
                tf2.setText("Please enter a integer number");
                }
            }
        }
        public static void main(String[] args) {
            new OddEven();
        }
}
```

Q. Create a Frame with two textFields, one of which show the mouse pointer inside or outside Frame and another should xhow X and Y coordinate of pointer when the user moves the mouse inside the frame.




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MousePointerStatusApp extends JFrame
implements MouseListener, MouseMotionListener {
     //Create Component class Objects
     JTextField stf;
     JTextField ctf;

     public MousePointerStatusApp() {
          //Create components
          stf = new JTextField();
          stf.setColumns(10);
          ctf = new JTextField();
          ctf.setColumns(10);
          //Set Frame layout
          setLayout(new FlowLayout());
```

```
        setVisible(true);
        setSize(400, 200);
        setTitle("MouseEvent");
        stf.setEditable(false);
        ctf.setEditable(false);
        //Register Event Listener
        addMouseListener(this);
        addMouseMotionListener(this);
        //add components to the frame
        add(stf); add(ctf);
    }
    @Override
    public void mouseClicked(MouseEvent e) { }
    @Override
    public void mousePressed(MouseEvent e) { }
    @Override
    public void mouseReleased(MouseEvent e) { }
    @Override
    public void mouseEntered(MouseEvent e) { }
    @Override
    public void mouseDragged(MouseEvent e) { }
    @Override
    public void mouseMoved(MouseEvent e) {
        stf.setText("Mouse Entered");
        int x = e.getX();
        int y = e.getY();
        ctf.setText("X= "+ x + " Y= "+ y);
    }
```

```
    @Override
    public void mouseExited(MouseEvent e) {
        stf.setText("Mouse Outside");
        ctf.setText(" ");
    }

    public static void main(String[] args) {
        new MousePointerStatusApp();
    }

}
```

DialogBox (JOptionPane)

Q. What is Dialog box? Explain its types?

A dialog box is used to interact with the users such as collecting user data, confirming actions, displaying alerts or warnings etc. The ***JOptionPane*** class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.

JOptionPane class is used to display three types of dialog boxes:

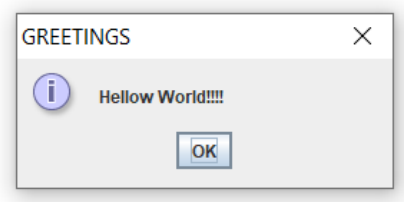
(a) showMessageDialog(): We can use **JOptionPane.showMessageDialog()** to display a message to the user with an OK button.

Syntax:

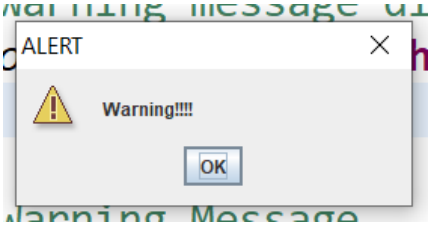
```
JOptionPane.showMessageDialog(Component parentComponent,
Object message, String title, int messageType);
```

- ***parentComponent****:* The parent component or window relative to which the message dialog is displayed.
- ***message****:* The message to be displayed in the dialog. This can be a string or any object that will be converted to a string.
- ***title****:* The title of the dialog, typically a string.
- ***messageType****:* An integer constant specifying the message type. Common values include:
    - ***JOptionPane.PLAIN_MESSAGE****: No icon (plain message).*
    - ***JOptionPane.INFORMATION_MESSAGE****: Information icon.*
    - ***JOptionPane.WARNING_MESSAGE****: Warning icon.*
    - ***JOptionPane.ERROR_MESSAGE****: Error icon.*
    - ***JOptionPane.QUESTION_MESSAGE****: Question icon.*

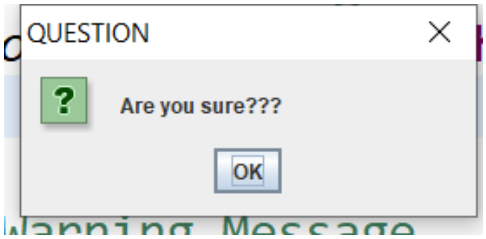
```
import javax.swing.*;
public class OptionPaneExample extends JFrame{

	public OptionPaneExample() {
		// Display a simple message dialog
	 JOptionPane.showMessageDialog(this, "Hellow World!!!!",
	"GREETINGS", JOptionPane.INFORMATION_MESSAGE);
```




```
		// Display a Warning message dialog
	 JOptionPane.showMessageDialog(this, "Warning!!!!",
	"ALERT", JOptionPane.WARNING_MESSAGE);
```




```
		// Display a Question message dialog
	 JOptionPane.showMessageDialog(this, "Are you sure???",
	"QUESTION", JOptionPane.QUESTION_MESSAGE);
```




```
	}

	public static void main(String[] args) {
		new OptionPaneExample();
	}

}
```

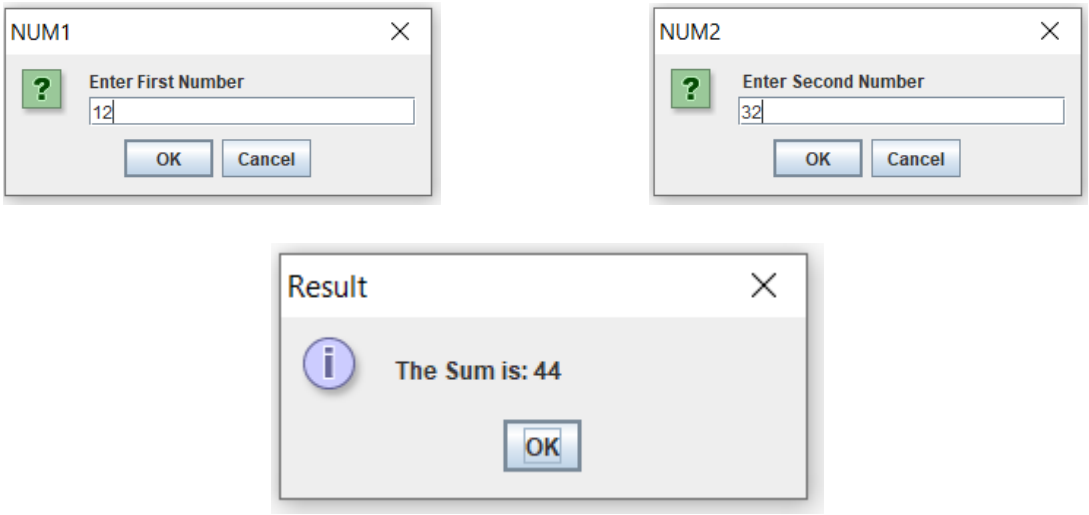
(b) showInputDialog(): We can use **JOptionPane.showInputDialog()** to prompt the user for input.

Syntax:

String input = JOptionPane.showInputDialog(Component parentComponent, Object message, String title, int messageType);

Q. Write a simple GUI addition application that used two dialogs to obtain integer from the user and a message dialog to display the sum.

```
import javax.swing.*;


public class InputDialog {
     public static void main(String[] args) {
          try {
          //Get two numbers from user
   int num1 = Integer.parseInt(JOptionPane.showInputDialog( null,
   "Enter First Number", "NUM1 ", JOptionPane.QUESTION_MESSAGE));
   int num2 = Integer.parseInt(JOptionPane.showInputDialog( null,
   " Enter Second Number", "NUM2", JOptionPane.QUESTION_MESSAGE));
                //Find Sum
                int sum = num1+num2;
                //Show the value of sum
JOptionPane.showMessageDialog(null, "The Sum is: "+
String.valueOf(sum), "Result", JOptionPane.INFORMATION_MESSAGE);
          }catch(NumberFormatException e) {
JOptionPane.showMessageDialog(null, "Please Enter Integer Only!",
"Invalid Input", JOptionPane.ERROR_MESSAGE);
          }
     }
}
```



(b) showConfirmDialog(): We can use **JOptionPane.showConfirmDialog()** to display a confirmation dialog that typically asks the user to confirm an action or make a choice with options like "Yes," "No," "OK," or "Cancel."

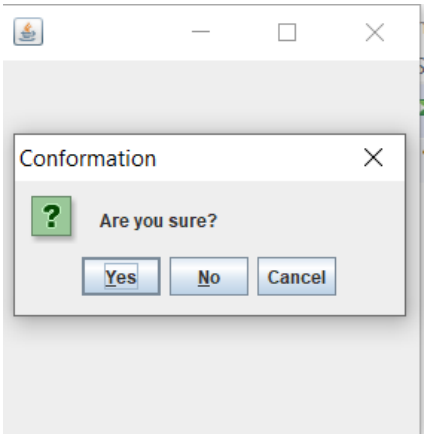
Syntax:

```
int Choice = JOptionPane.showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType);
```

optionType: An integer constant specifying the set of buttons or options to display in the dialog. Common values include:

- ***JOptionPane.YES_NO_OPTION:*** Display "Yes" and "No" buttons.
- ***JOptionPane.YES_NO_CANCEL_OPTION:*** Display "Yes," "No," and "Cancel" buttons.
- ***JOptionPane.OK_CANCEL_OPTION:*** Display "OK" and "Cancel" buttons.

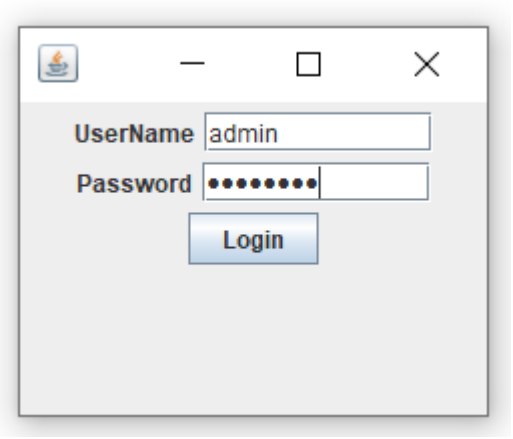
Example: Let's create a JFrame such that if user clicks the close botton and a confirmation dialogue appears with options: Yes, No and Cancel.

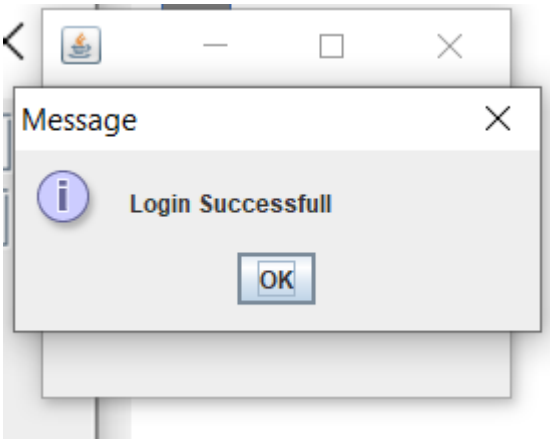



```
import javax.swing.*;
import java.awt.event.*;
public class OptionPaneExample extends WindowAdapter{
JFrame f;
OptionPaneExample(){
     //create a JFrame
    f=new JFrame();
/*Set the default close operation to DO_NOTHING_ON_CLOSE to
 allow custom handling of the window closing event*/
   f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);

    //Register a window listener to handle window events
    f.addWindowListener(this);

    //set Layout
    f.setSize(300, 300);
    f.setLayout(null);
    f.setVisible(true);
}
```

```
public void windowClosing(WindowEvent e) {
    int a=JOptionPane.showConfirmDialog(f,"Are you sure?",
         "Conformation",JOptionPane.YES_NO_CANCEL_OPTION);
     if(a==JOptionPane.YES_OPTION){
         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     }
}
public static void main(String[] args) {
    new  OptionPaneExample();
}
}
```

Q. Create a login Application in JAVA. If username and password is incorrect show invalid logigin message in dialog box else show login successfull.



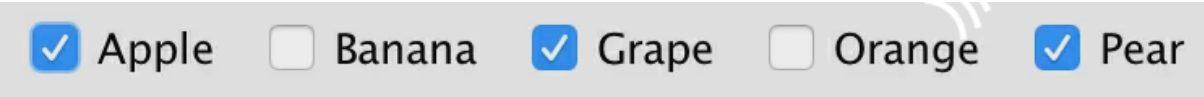



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Test extends JFrame implements ActionListener{
     JTextField userName;
     JPasswordField password;
```

```
JButton btn;
JLabel pLabel, uLabel;

Test(){
     userName = new JTextField();
     userName.setColumns(10);
     password = new JPasswordField();
     password.setColumns(10);
     btn = new JButton("Login");
     uLabel = new JLabel("UserName");
     pLabel = new JLabel("Password");
     setLayout(new FlowLayout());
     setVisible(true);
     setSize(300, 300);
     add(uLabel);
     add(userName);
     add(pLabel);
     add(password);
     add(btn);
     btn.addActionListener(this);
}

@Override
public void actionPerformed(ActionEvent e) {
     if(e.getSource()==btn) {
          String uname = userName.getText();
          char[] pass = password.getPassword();
          String pword = new String(pass);
```

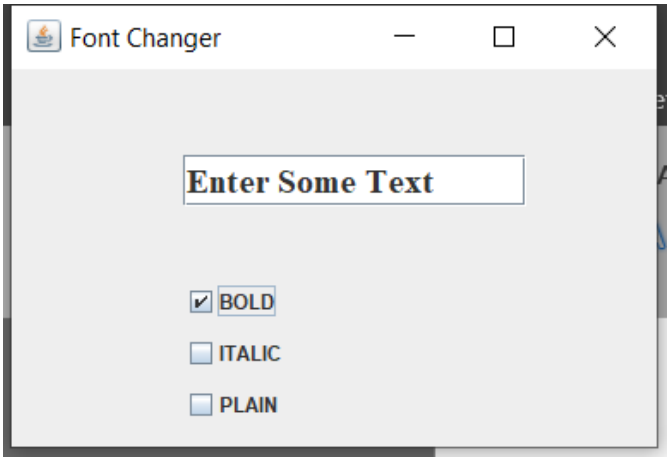
```
     if(uname.equals("admin")  &&  pword.equals("password"))
{
JOptionPane.showMessageDialog(this, "Login Successfull");
}else {
JOptionPane.showMessageDialog(this, "Login UnSuccessfull");
}
          }
     }
     public static void main(String[] args) {
          new Test();
     }

}
```

JCheckBox



The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on.

Q. WAP to change font of the text int the TextField of the frame. The frame should contains 3 checkBoxes named bold, italic and plain.




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FontChangeApp extends JFrame implements ItemListener {
    //Create Component class objects
    JTextField jtf;
    JCheckBox bold, italic, plain;
    //Create Font objects
Font plain_font =new Font("Times New Roman",Font.PLAIN, 20);
```

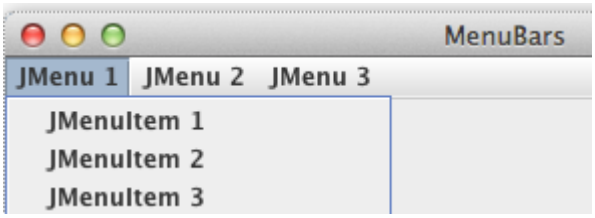
```
Font bold_font = new Font("Times New Roman", Font.BOLD, 20);
Font italic_font= new Font("Times New Roman",Font.ITALIC,
20);
Font default_font = new Font("Times New Roman",Font.PLAIN,
20);
     public FontChangeApp() {
          //Create Componens
          jtf = new JTextField("Enter Some Text");
          bold = new JCheckBox("BOLD");
          italic = new JCheckBox("ITALIC");
          plain = new JCheckBox("PLAIN");

          //Set frame layout and component layouts
          setLayout(new FlowLayout() );
          setTitle("Font Changer");
          setSize(300, 300);
          setVisible(true);

          //Register Listener
          bold.addItemListener(this);
          italic.addItemListener(this);
          plain.addItemListener(this);

          //Add components to the frame
          add(jtf);add(bold);add(plain);add(italic);
     }
```

```
    public void itemStateChanged(ItemEvent e) {
        //find the source that generated event
        if(e.getSource() == bold) {
    //if bold CheckBox is checked change JTF font to bold
                if(e.getStateChange() == 1) {
                    jtf.setFont(bold_font);
                }else {
                    jtf.setFont(default_font);
                }
        }
        if(e.getSource() == italic) {
//if italic CheckBox is checked change JTF font to italic
                if(e.getStateChange() == 1) {
                    jtf.setFont(italic_font);
                }else {
                    jtf.setFont(default_font);
                }
        }
        if(e.getSource() == plain) {
    //if plain CheckBox is checked change JTF font to plain
                if(e.getStateChange() == 1) {
                    jtf.setFont(plain_font);
                }else {
                    jtf.setFont(default_font);
                }
        }
    }
```

```
    public static void main(String[] args) {
        new FontChangeApp();
    }
}
```

JMenuBar, JMenu & JMenuItem

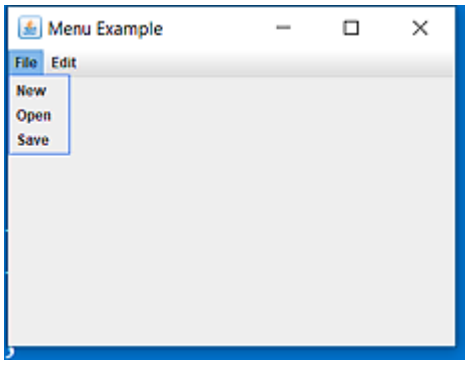
Q. How do you create meanu in swing. Explain with an example.

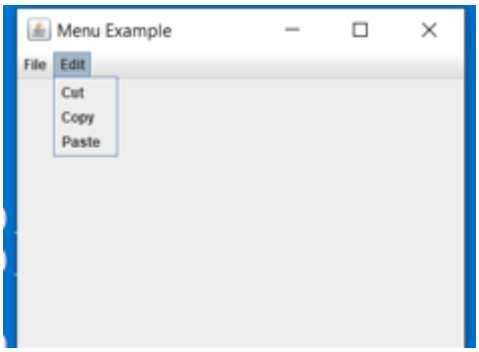


JMeauBar: ***JMenuBar*** is a Swing component used to create a menu bar at the top of a frame or window. It can contain multiple menu items.

JMenu: ***JMenu*** is a Swing component that represents a menu within a menu bar. When we click on a menu, it displays a list of items or submenus that can be selected.

JMenuItem: ***JMenuItem*** is a Swing component that represents an individual item within a menu.

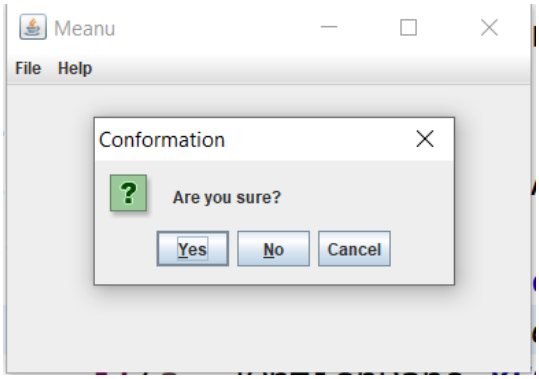




```
import javax.swing.*;
public class MenuExample extends JFrame{
     JMenuBar mb;
     JMenu file_menu, edit_menu;
JMenuItem new_menu_item, open_menu_item, save_menu_item;
JMenuItem cut_menu_item, copy_menu_item, paste_menu_item;


     public MenuExample() {
               //Set Frame
          setLayout(null);
          setVisible(true);
          setTitle("Meanu");
          setSize(400, 300);
            //Create MenuBar
          mb = new JMenuBar();

          //Create Menus
          file_menu = new JMenu("File");
          edit_menu = new JMenu("Edit");

          //Create MenuItems
          new_menu_item = new JMenuItem("New");
          open_menu_item = new  JMenuItem("Open");
          save_menu_item = new JMenuItem("Save");
          cut_menu_item = new JMenuItem("Cut");
          copy_menu_item = new JMenuItem("Copy");
          paste_menu_item = new JMenuItem("Paste");
```

```
        //Add MenuItems to the menu
        file_menu.add(new_menu_item);
        file_menu.add(open_menu_item);
        file_menu.add(save_menu_item);
        edit_menu.add(cut_menu_item);
        edit_menu.add(copy_menu_item);
        edit_menu.add(paste_menu_item);

        //add menu to the MenuBar
        mb.add(file_menu);
        mb.add(edit_menu);

        //add menu bar to frame
        setJMenuBar(mb);
    }
    public static void main(String[] args) {
        new MenuExample();
    }
}
```

Q. Write a program to create menubar. The menubar should contains two menus(File and Help). File menu should contain menuitems "open" and "close". Also handle the event to display a dialog box with message "Exiting from program" when user click menuitem close.

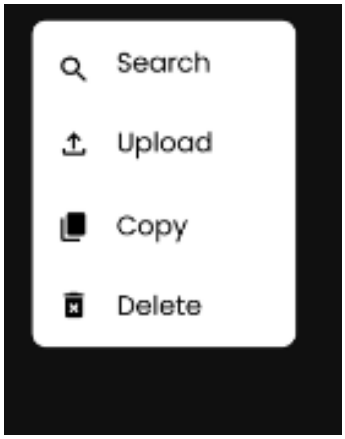


```
import java.awt.event.*;
import javax.swing.*;
public class MenuExample extends JFrame implements
ActionListener{
     JMenuBar mb;
     JMenu file_menu, help_menu;
     JMenuItem open_menu_item, close_menu_item;
     public MenuExample() {
               //Set Frame
          setLayout(null);
          setVisible(true);
          setTitle("Meanu");
          setSize(400, 300);
            //Create MenuBar
           mb = new JMenuBar();

          //Create Menus
          file_menu = new JMenu("File");
          help_menu = new JMenu("Help");
          //Create MenuItems
          open_menu_item = new  JMenuItem("Open");
          close_menu_item = new JMenuItem("Close");

          //Add MenuItems to the menu
          file_menu.add(open_menu_item);
          file_menu.add(close_menu_item);
```

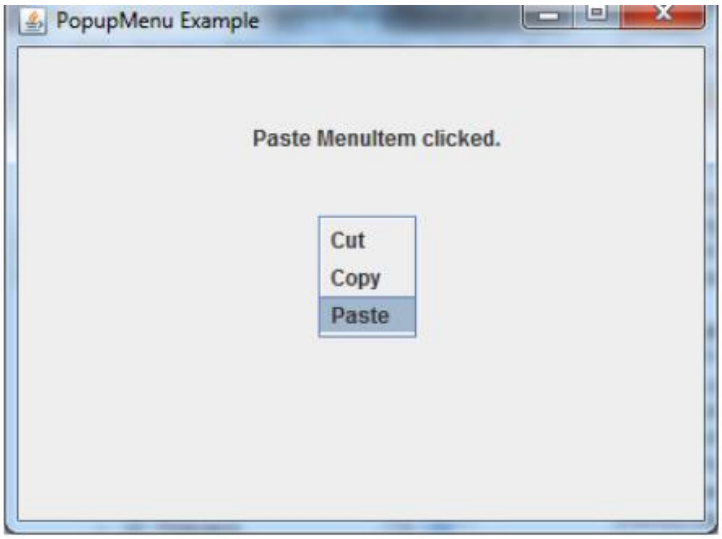
```
            //add menu to the MenuBar
            mb.add(file_menu);
            mb.add(help_menu);

            //add menu bar to frame
            setJMenuBar(mb);

            //add event listener
            close_menu_item.addActionListener(this);
      }
      public void actionPerformed(ActionEvent e) {
            //find source
            if(e.getSource() == close_menu_item) {
int   a=JOptionPane.showConfirmDialog(mb,"Are  you  sure?",
"Conformation",JOptionPane.YES_NO_CANCEL_OPTION);
            if(a==JOptionPane.YES_OPTION) {
                        System.exit(0);
                  }
            }
      }
      public static void main(String[] args) {
            new MenuExample();
      }
}
```

JPopupMenu

PopupMenu can be dynamically popped up at specific position within a component. It inherits the JComponent class.



Q. Create a popup menu attached to frame, if you click within the area of frame popup meanu will be shown. Similarly clicked on the menu item show message "...item is selected" on label.




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PopupMenuExample extends JFrame implements
ActionListener, MouseListener {
```

```
JLabel label;
JPopupMenu popMenu;
JMenuItem item1, item2, item3;

public PopupMenuExample() {

    //Set Frame
    setLayout(null);
    setVisible(true);
    setTitle("PopupMenu Example");
    setSize(400, 300);

    //create label
    label = new JLabel();

    //create poppup meanu
    popMenu = new JPopupMenu("Edit");

    //create meanu items
    item1 = new JMenuItem("cut");
    item2 = new JMenuItem("copy");
    item3 = new JMenuItem("paste");
    //Add event listener
    item1.addActionListener(this);
    item2.addActionListener(this);
    item3.addActionListener(this);
    addMouseListener(this);
```

```
        //add menuitems to popupmenu
        popMenu.add(item1);
        popMenu.add(item2);
        popMenu.add(item3);
        //add label to the frame
        add(label);
        label.setBounds(100, 10, 200, 100);

    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource()==item1) {
            label.setText("cut MenuItem clicked");
        }
        if(e.getSource()==item2) {
                label.setText("copy MenuItem clicked.");
        }
        if(e.getSource()==item3) {
                label.setText("paste MenuItem clicked.");
        }
    }
    @Override
    public void mouseClicked(MouseEvent e) {
        // Check if the right mouse button is clicked
        if (e.getButton() == MouseEvent.BUTTON3) {
            popMenu.show(this, e.getX(), e.getY());
        }
    }
```

```
    @Override
    public void mousePressed(MouseEvent e) {}
    @Override
    public void mouseReleased(MouseEvent e) {}
    @Override
    public void mouseEntered(MouseEvent e) {}
    @Override
    public void mouseExited(MouseEvent e) {}
    public static void main(String[] args) {
        new PopupMenuExample();
    }
}
```

CHAPTER: 5

APPLETS & APPLICATION

APPLETS

A Java applet is a short programme created specifically to be embedded and run inside a web browser. Typically, applets are used to add interactive elements and functionality to web pages. They may carry out a variety of functions, including processing user input and displaying visuals, animations, and user interfaces. In order to run Java applets in a web browser, a Java Virtual Machine (JVM) is required. The JVM is responsible for executing Java bytecode, which is the compiled form of Java applet code. It allows the applet to run within the web browser environment.

They were more commonly used in the past but have become less common in modern web development due to security concerns and the availability of other web technologies like JavaScript.

To create applete in we need to import two packages(**java.applet & java.awt**) and the class must extends **java.applet.Applet** class:

- **java.applet:** This package contains the Applet class, which is the core class for creating applets.
- **java.awt:** This package provides classes and interfaces for creating graphical user interfaces, including the Graphics class for rendering graphics and text in applets.

Executing Program in Applet

Q. Create a sample Applet and demonstrate how we can run a Java Applet.

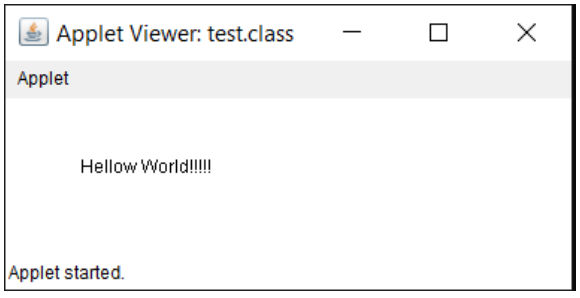
Step 1: Create a Java class that extends the **java.applet.Applet** class, which is the base class for applets. In this class, override the **paint** method to specify what should be drawn on the applet's canvas.

```
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet{
     public void paint(Graphics g) {
          g.drawString("Hellow World!!!!!", 50, 50);
     }

}
```

Step 2: Execute applet. There are two ways to execute applete:

A. Using an applet viewer. An applet viewer executes applet in a window. This is generally the fastest and easiest way to test the applet. Create an applet that contains ***applet tag*** in comment .

```
/*<applet code="test.class" width="300" height="200">

</applet>
*/
import java.applet.*;
import java.awt.*;
public class HelloWorldApplet extends Applet{
     public void paint(Graphics g) {
          g.drawString("Hellow World!!!!!", 50, 50);
     }

}
```



B. Executing the applet within a Java-compatible web browser.

Create an HTML file (e.g., **index.html**) that will embed our applet.

<applet code="AppletClassName.class" width="Width" height="Height">

<!-- Alternative content or message for non-Java-enabled browsers -->

</applet>

Here,

<applet> is the HTML tag used to embed a Java applet.

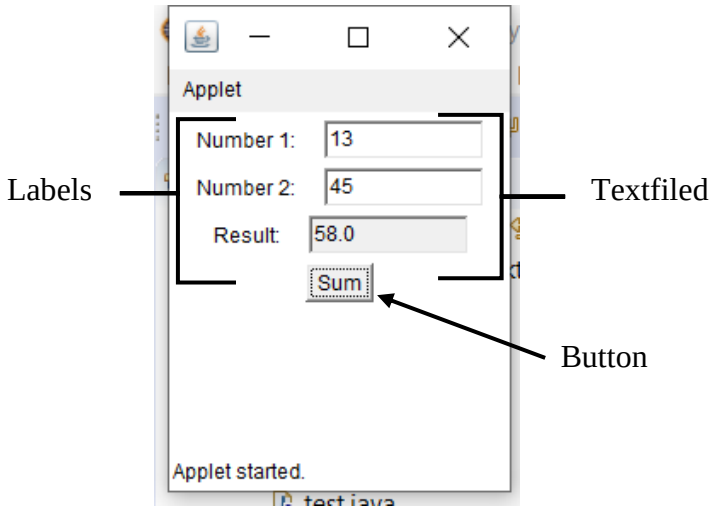
code is an attribute that specifies the applet's class file.

width and **height** are attributes that define the dimensions of the applet's display area

Open the HTML file (**index.html**) in a web browser that still supports Java applets (e.g., older versions of Firefox or Internet Explorer with Java support). We should see the "Hello, World!" message displayed on the webpage.

```
<!DOCTYPE html>
<html>
<head>
    <title> Applet</title>
</head>
<body>
    <applet code="HelloWorldApplet.class" width="300" height="200">
        Shit!!!!!Your browser does not support Java applets.
    </applet>
</body>
</html>
```

Q. Write an applet program with three text fields with following names: "number1", "number2", "result". When the user clicks a "sum" button then the sum of two number in "number1" & "number2" should be displayed in the text field named "result".




```
/*<applet code="test.class" width="300" height="200">

</applet>
*/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class test extends Applet implements
ActionListener {

    TextField number1, number2, result;
    Button sumButton;
    Label FirstNumber, SecondNumber, Sum;
```

```
public void init() {
    //Create TextFields for input and result display
    number1 = new TextField(10);
    number2 = new TextField(10);
    result = new TextField(10);
    result.setEditable(false);//read only

    //Create labels for TextFields
    FirstNumber = new Label("Number 1: ");
    SecondNumber = new Label("Number 2: ");
    Sum = new Label("Result: ");

    // Create Button
    sumButton = new Button("SUM");

    //register Listener
    sumButton.addActionListener(this);

    //Add Components to the Applet
    add(FirstNumber);
    add(number1);
    add(SecondNumber);
    add(number2);
    add(Sum);
    add(result);
    add(sumButton);
}
```

```
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == sumButton) {
         //Get the values from text filed number1 & number2
      double num1 = Double.parseDouble(number1.getText());
      double num2 = Double.parseDouble(number2.getText());
      //getText() returns String so we should parse it
      double add = num1 + num2;

      //Display result in TextField result
      result.setText(String.valueOf(add));
       }
    }
}
```

The HTML attributes of applet Tag

Q. List out the attributes of applet tag and explain them.

The <applet> tag takes following attributes:

Attribute	Explanation	Example
code	Specifies the URL of the Java applet class file.	**<applet code="MyApplet.class">**
width	Sets the display width of the applet panel in pixels.	**<applet width="300">**
height	Sets the display height of the applet panel in pixels.	**<applet height="200">**
align	Specifies the position of the applet relative to surrounding content.	**<applet align="middle">**
alt	Provides alternative text if the browser doesn't support Java.	**<applet alt="Please enable Java to view this content.">**
archive	Specifies the archived or compressed version of an applet.	**<applet archive="applet_resources.jar">**
object	An alternative way to specify the URL or reference to a serialized representation of an applet.	**<applet object="MyApplet.class">**
codebase	Specifies the exact or relative URL of the applet's **.class** file specified in the **code** attribute.	**<applet codebase="http://example.com/applets/">**
hspace	Specifies the horizontal space around the applet.	**<applet hspace="10">**
vspace	Specifies the vertical space around the applet.	**<applet vspace="10">**

name	Specifies the name for the applet, which can be used for targeting with JavaScript or as a reference in the HTML document.	**<applet name="myApplet">**

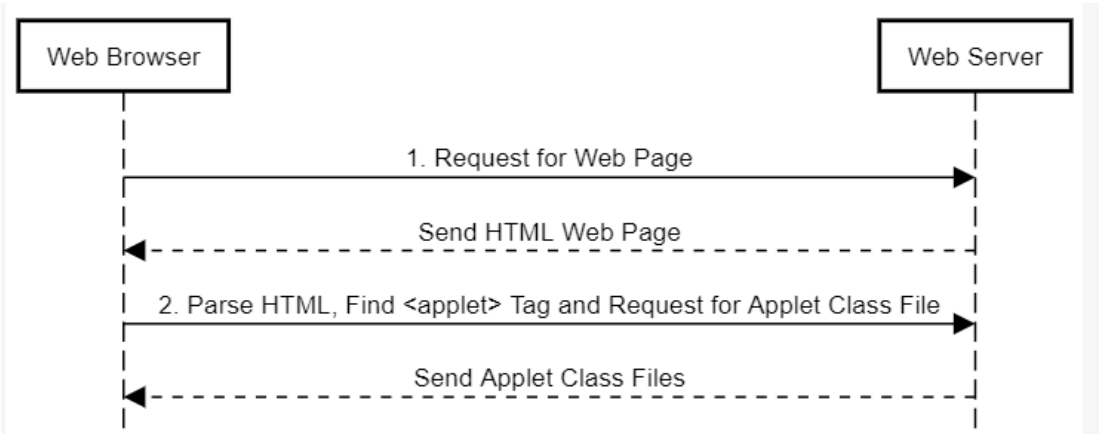
APPLET ARCHITECTURE

An applet is a window based programming environment. So applet architecture is different than console base program.

GUI-Based Program: applets are typically used to create graphical user interface (GUI) applications. Unlike console-based programs that interact with users through text in a terminal or command prompt, applets provide a graphical interface where users can interact with buttons, menus, forms, and other GUI components.

Event-Driven: In event-driven programming, the program waits for events (e.g., user actions like clicking a button or typing in a text field) and responds to those events by executing specific event handlers. Applets rely on event handlers to respond to user interactions or system events.

Additional Threads for Long-Running Tasks: In cases where an applet needs to perform tasks that could take an extended period or run continuously (e.g., animation, background processing), it's advisable to start an additional thread of execution. This separate thread can handle these tasks without blocking the main event-handling thread, keeping the applet responsive.

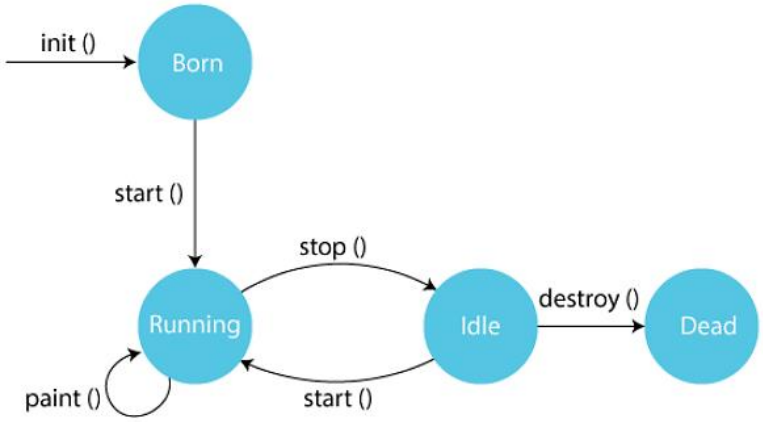


APPLET vs JAVA APPLICATION

SN	Characteristic	Java Application	Java Applet
1.	**Definition**	An application is a standalone Java program that can be run independently on a client/server without the need for a web browser.	An applet is a form of Java program which is embedded with an HTML page and loaded by a web server to be run on a web browser.
2.	**main() method**	The execution of the program starts from the main() method.	There is no requirement of main() method for the execution of the program.
3.	**Access Restrictions**	The application can access local disk files/ folders and the network system.	The applet doesn't have access to the local network files and folders.
4.	**GUI**	It doesn't require any Graphical User Interface (GUI).	It must run within a GUI.
5.	**Security**	It is a trusted application and doesn't require much security.	It requires high-security constraints as applets are untrusted.
6.	**Environment for Execution**	It requires Java Runtime Environment (JRE) for its successful execution.	It requires a web browser like Chrome, Firefox, etc for its successful execution.

LIFE CYCLE OF APPLET

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application.



There are five applet methods that are called by the applet container from the time the applet is loaded into the browser to the time it's terminated by the browser.

1. init(): The ***init()*** method is the first method to run that initializes the applet. Typical actions performed here are initializing fields, creating GUI components, loading sounds to play, loading images to display and creating threads. This method is called **only once** during the run time of applet.

2. start(): The ***start()*** method contains the actual code of the applet and starts the applet. It is invoked immediately after the ***init()*** method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tab to another in the browser.

3. paint(): The ***paint()*** method belongs to the Graphics class in Java. It is used to draw shapes like circle, square, trapezium, etc., in the applet. It is executed after the ***start()*** method and when the browser or applet windows are resized.

```
public void paint(Graphics g) {

// Typical actions performed here involve drawing with the
Graphics object g

    }
```

4. stop(): When the user navigates away from the web page or minimizes the browser, the applet becomes invisible, and the ***stop()*** method is called. ***stop()*** performs tasks that might be required to suspend the applet's execution, so that the applet does not use computer processing time when it's not displayed on the screen. If the user returns to the applet later, the ***start()*** method will be called again to resume its operation.

4. destroy(): The **d*estroy()*** method is called when the applet is no longer needed or when the web page is closed. It is used for cleanup tasks, such as releasing resources, closing files, and ending any threads or background processes. Once the **destroy()** method is called, the applet cannot be restarted. It marks the end of the applet's lifecycle.

PASSING PARAMETERS FROM HTML TO APPLETS

Q. How can a HTML file pass data to an Applet? Explain with relevant code tags.

Q. Write an applet program to illustrate the message sharing between HTML and Applets.

Q. Create an applet that takes value from the HTML in which it is embedded and displays it.

The APPLET tag in HTML allows us to pass parameters to our applet. Java allows users to pass user-defined parameters to an applet with the help of <***PARAM***> tags.

Syntax of the <PARAM> tag is:

```
<PARAM NAME= "parameter_name" VALUE = "parameter_value">
```

Example:

```
<PARAM NAME= "AGE" VALUE = "25">

<PARAM NAME= "NAME" VALUE = "Ankit Kharel">
```

The ***getParameter()*** method of the Applet class can be used to retrieve the parameters passed from the HTML page. It returns the value of the specified parameter in the form of a String object. Thus, for **numeric** and **boolean** values, we will need to convert their string representations into their internal formats.

Syntax of the getParameter() is:

```
String getParameter("parameter_name");
```

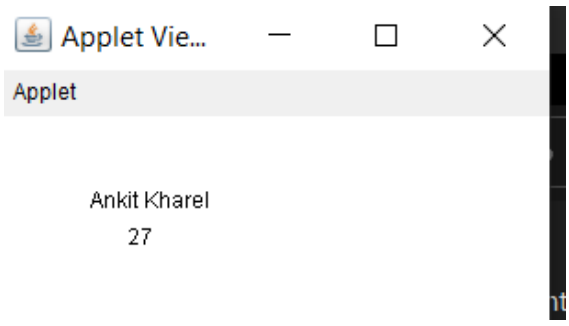
Example:

```
String str = getParameter("NAME");
```

`String age = getParameter("AGE");` *retrieves the "AGE" parameter as a string from the applet's parameters.*

`int ageValue = Integer.parseInt(age);` *parses the retrieved string into an integer (**int**) using the **Integer.parseInt** method. The **Integer.parseInt** method in Java throws a **NumberFormatException** if the string it is trying to parse is not a valid integer representation.*

```
<html>
<applet code="test.class" width="300" height="200">
          <param name="NAME" value="Ankit Kharel">
          <param name="AGE" value="27">
</applet>
</html>
import java.applet.*;
import java.awt.*;
public class test extends Applet{
     String name, age;

     public void init() {
          //get name and age
          name = getParameter("NAME");
          age = getParameter("AGE");

     }
     //paint name and age
     public void paint(Graphics g) {
          g.drawString(name, 50, 50);
          g.drawString(age, 70, 70);
     }
}
```



Display Images in Applet

Q. Explain how to display image in applet.

An applet can display images of the format GIF, JPEG, BMP, and others. To display an image within the applet, we use the drawImage/getImage method found in the java.awt.Graphics class.

Step 1: Create object of class Image

```
Image image_object;
```

Step 2: Initialize the Image object using getImage() method.

```
image_object = getImage(getDocumentBase(), "Image_Name");
```

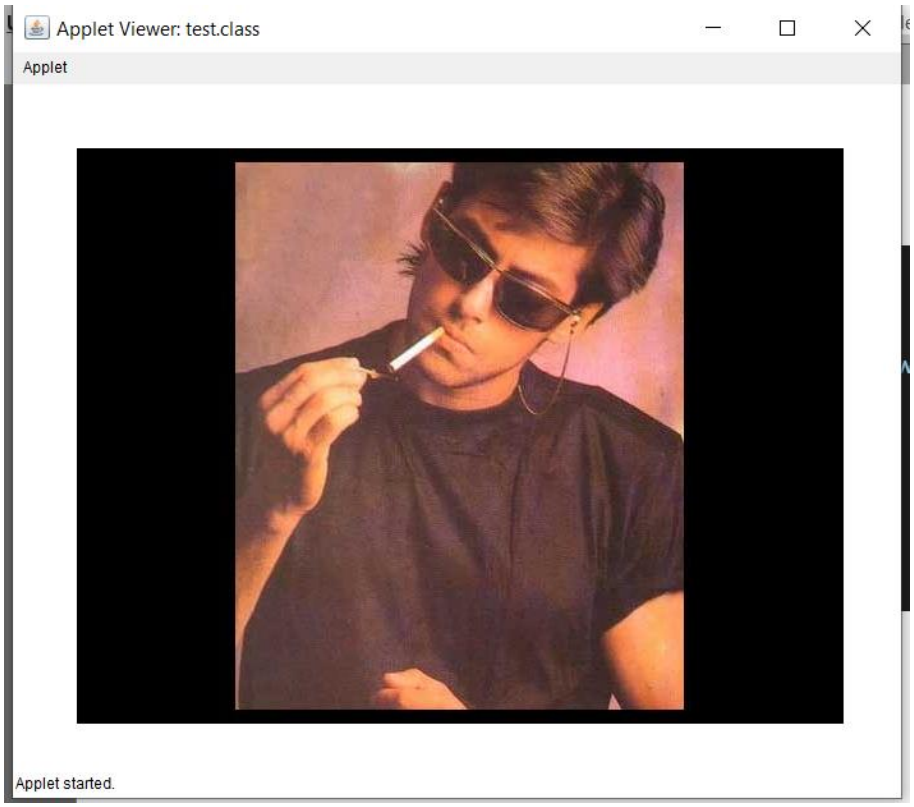
getDocumetBase() : This is another method provided by the java.applet.Applet class. It returns the URL of the document (usually an HTML web page) in which the applet is embedded.

Step 3: Paint the Image on the Applet Canvas using drawImage() method.

```
drawImage(image_object, x_coordinate, y_coordinate, this );
```

```
/*<applet code="test.class" width="300" height="200">
</applet>
*/
import java.applet.*;
import java.awt.*;
public class test extends Applet{
    Image img;
    public void init() {
       img=getImage(getDocumentBase(), "sampleImage.jpg");
    }
```

```
    public void paint(Graphics g) {
        if(img!=null) {
            g.drawImage(img, 50, 50, this);
        }else {
            g.drawString("Image not Loaded", 50,50);
        }
    }

}
```



Audio in Applet

An applet can play an audio file represented by the AudioClip interface in the java.applet package.

Step 1: Create object of class AudioClip.

```
AudioClip audio_clip_object;
```

Step 2: Initialize the audio_clip object using getAudioClip () method.

```
audio_clip_object = getAudioClip(getDocumentBase(), "Audio_Name");
```

`getDocumetBase()` : *This is another method provided by the java.applet.Applet class. It returns the base URL of the document (usually an HTML web page) in which the applet is embedded.*

Step 3: Start playing the audio.

```
Audio_clip_object.play();   or   Audio_clip_object.loop();
```

`public void play():` *Plays the audio clip one time, from the beginning.*

`public void loop():` *Causes the audio clip to replay continually.*

Step 4: Stop playing three audio clip.

```
Audio_clip_object.stop();
```

Q. WAP using applet to upload and play an audio clip.

Q. Create an applet with the functionalities to play, stop and repeat the audio.

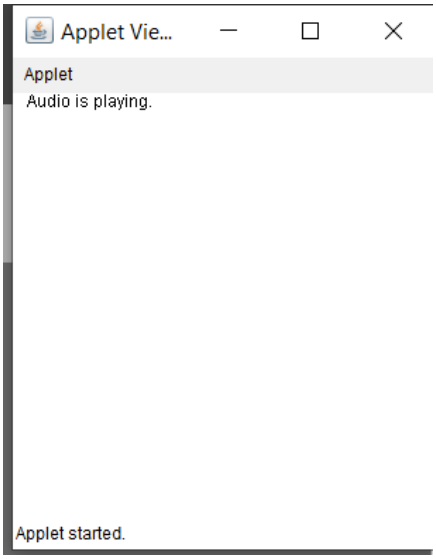
Q. Create an applet that plays a sample auiod file. Also create a GUI that allows users to play, stop or replay the audio. Add a functionality to automatically stop the audio if the applet is inactive.

```
/*<applet code="test.class" width="300" height="200">
</applet>
*/
import java.applet.*;
import java.awt.*;
public class test extends Applet {
    private AudioClip audioClip;

    public void init() {
        // Load an audio clip here
 audioClip = getAudioClip(getDocumentBase(), "Sample.wav");
    }

    public void start() {
      // Start playing the audio when the applet starts
        if (audioClip != null) {
            audioClip.loop(); // Play audio in a loop
        }
    }

    public void stop() {
        // Stop playing the audio when the applet stops
        if (audioClip != null) {
```

```
            audioClip.stop();
        }
    }
    public void paint(Graphics g) {
        // Check if the audio clip has been loaded
        if (audioClip != null) {
            g.drawString("Audio is playing.", 10, 10);
        } else {
            g.drawString("Audio not loaded!", 10, 10);
        }
    }
}
```

Let's call the applet as:

```
<html>
    <body>
        <applet code="test.class" width = "300" height = "300">

        </applet>
    </body>

</html>
```



Q. Write an applet program to play an audio file. The name of audio file is supplied from HTML tag.

HTML Code:

```
<html>
    <body>
        <applet code="test.class" width = "300" height = "300">
           <param name="AudioFile" value="Sample.wav">
        </applet>
    </body>

</html>
```

```
import java.applet.*;
import java.awt.*;
/*<applet code="test.class" width="300" height="200">
</applet>
*/
public class test extends Applet {
    private AudioClip audioClip;
    public void init() {
        // Get the audio file name from the HTML parameter
     String audioFileName = getParameter("audioFileName");
        // Load the audio clip using the supplied file name
audioClip = getAudioClip(getDocumentBase(), audioFileName);
    }
    public void start() {
        if (audioClip != null) {
            audioClip.loop(); // Play audio in a loop
        }
    }
```

```
    public void stop() {
        // Stop playing the audio when the applet stops
        if (audioClip != null) {
            audioClip.stop();
        }
    }

    public void paint(Graphics g) {
        // Check if the audio clip has been loaded
        if (audioClip != null) {
            g.drawString("Audio is playing.", 10, 10);
        } else {
            g.drawString("Audio not loaded!", 10, 10);
        }
    }
}
```

Q. COVERT APPLET TO APPLICATION

SECURITY RESTRICTION IN Applet

Q. What does security restriction means in applet?

Q. Discuss about the Sandbox Security model with regard to java applet.

Because applets are designed to be loaded from a remote site and then executed locally, security becomes vital. Java applets operate within a security sandbox to protect the user's system from potentially harmful code. These security restrictions are in place to prevent applets from performing unauthorized or malicious actions on the user's computer. Here are some common security restrictions imposed on Java applets:

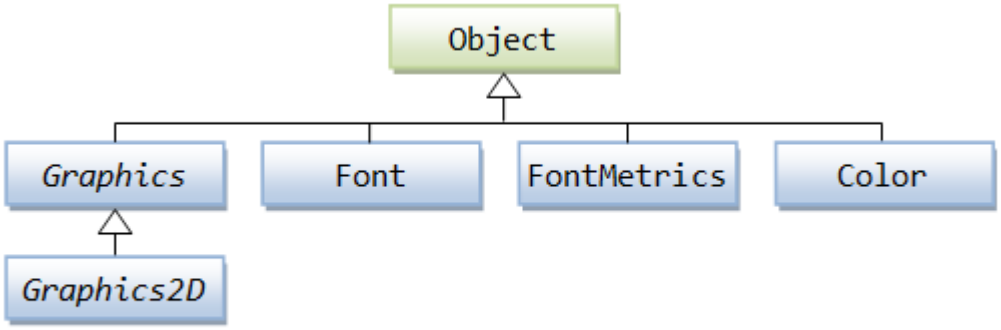
- Applets cannot access the local file system of the user's computer due to security concerns.
- They cannot connect to or retrieve resources from any third party server (any server other than the server it originated from).
- Applets have restricted access to certain system resources, such as hardware devices like printers or cameras. They usually require additional permissions to access these resources.
- Applet cannot read certain system properties.
- Applet cannot start any program on the host that's executing it.
- Windows that an applet brings up look different than windows that an application brings up.

CHAPTER: 6

GRAPHICS/IMAGES/ANIMATION/MULTIMEDIA

INTRODUCTION TO GRAPHICS PROGRAMMING

In Java, custom painting is done via the ***java.awt.Graphics*** class, which manages a graphics context, and provides a set of *device-independent* methods for drawing texts, figures and images on the screen on different platforms.



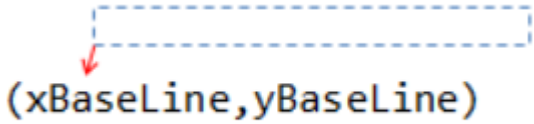
The Graphics class provides methods for drawing three types of graphical objects:

- Text strings: via the **drawString()** method.
- Vector-graphic primitives and shapes: via methods **drawXxx()** and **fillXxx(),** where Xxx could be Line, Rect, Oval, Arc, PolyLine, RoundRect, or 3DRect.
- Bitmap images: via the **drawImage()** method.

In Java, We can draw lines and rectangles on graphical components like **JPanel**, **Canvas**, or **Component** using various methods provided by the Java AWT (Abstract Window Toolkit) or Swing framework. Here are some commonly used methods for drawing lines and rectangles, along with descriptions:

1. Drawing text

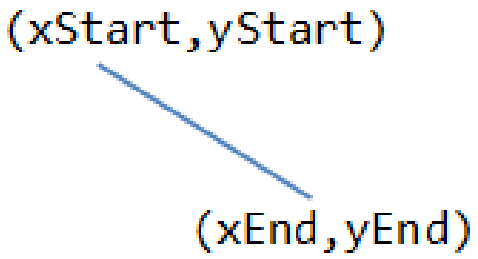
drawString(String *str*, int *xBaselineLeft*, int *yBaselineLeft*): used to draw text onto a graphical canvas.



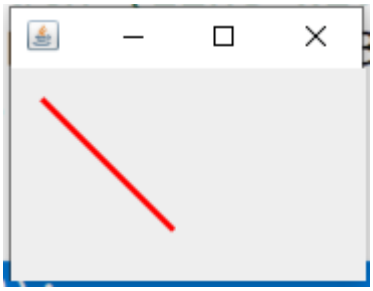

```
public void paintComponent(Graphics g) {
        Font font = new Font("Arial", Font.BOLD, 30);
        g.setFont(font);
        g.setColor(Color.RED);
        g.drawString("Hellow World", 100, 100);
}
```

2. Drawing Line

drawLine(int xStart, int yStart, int xEnd, int yEnd): used to draw a straight line between two points **(xStart, yStart)** and **(xEnd, yEnd)**.

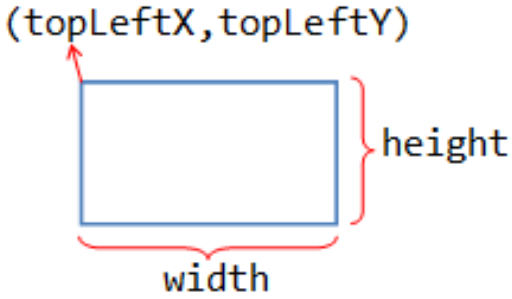


```
public void paintComponent(Graphics g) {
        g.setColor(Color.RED);
        g.drawLine (20, 20, 100, 100);
}
```

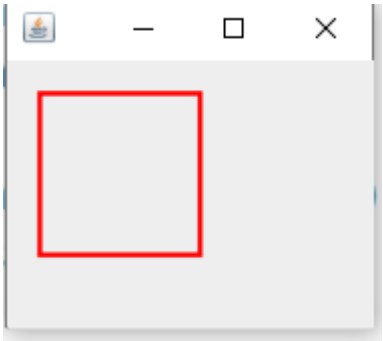


3. Drawing Rectangle

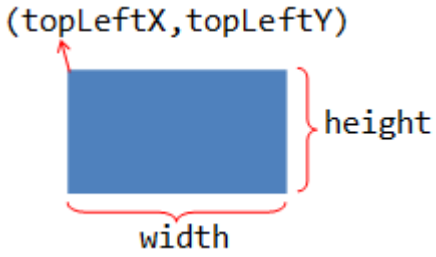
drawRect(int topLeftX, int topLeftY, int width, int height): This method is used to draw the outline of a rectangle with the specified coordinates **(topLeftX, topLeftY)** and dimensions **(width, height)**.



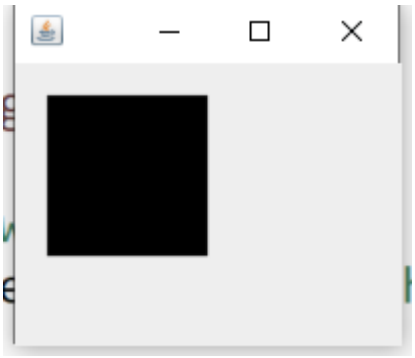

```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.drawRect (20, 20, 100, 100);
}
```



fillRect(int topLeftX, int topLeftX, int width, int height): Draws a filled rectangle in the current color.




```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillRect (20, 20, 100, 100);
}
```

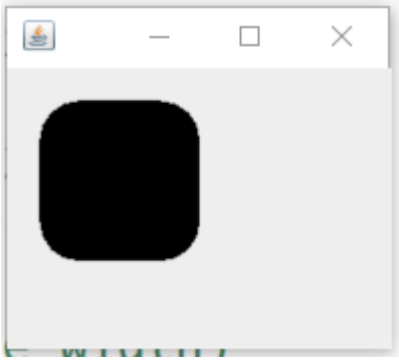


drawRoundRect(int topLeftX, int topLeftX, int width, int height, int arcWidth, int arcHeight): is used to draw the outline of a rounded rectangle.

```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.drawRoundRect (20, 20, 100, 100, 50, 50);
}
```

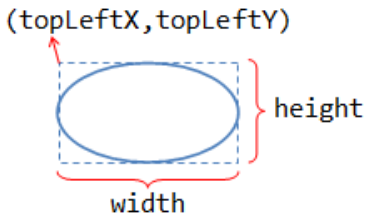
fillRoundRect(int topLeftX, int topLeftX, int width, int height, int arcWidth, int arcHeight): is used to draw the filled rounded rectangle

```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillRoundRect (20, 20, 100, 100, 50, 50);
}
```



4. Ovals

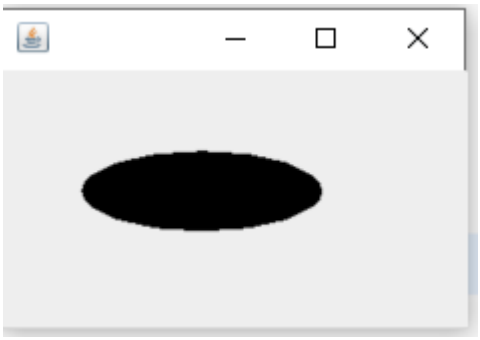
drawOval(int xTopLeft, int yTopLeft, int width, int height): Draws an oval in the current color with the specified width and height




```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.drawOval(50, 50, 150, 50);
}
```

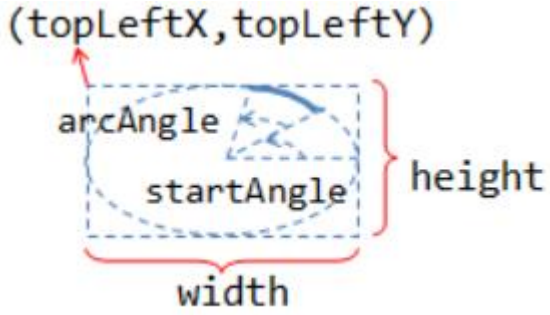
fillOval(int xTopLeft, int yTopLeft, int width, int height): Draws filled oval in the current color with the specified width and height.

```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillOval(50, 50, 150, 50);
}
```



5. Arcs

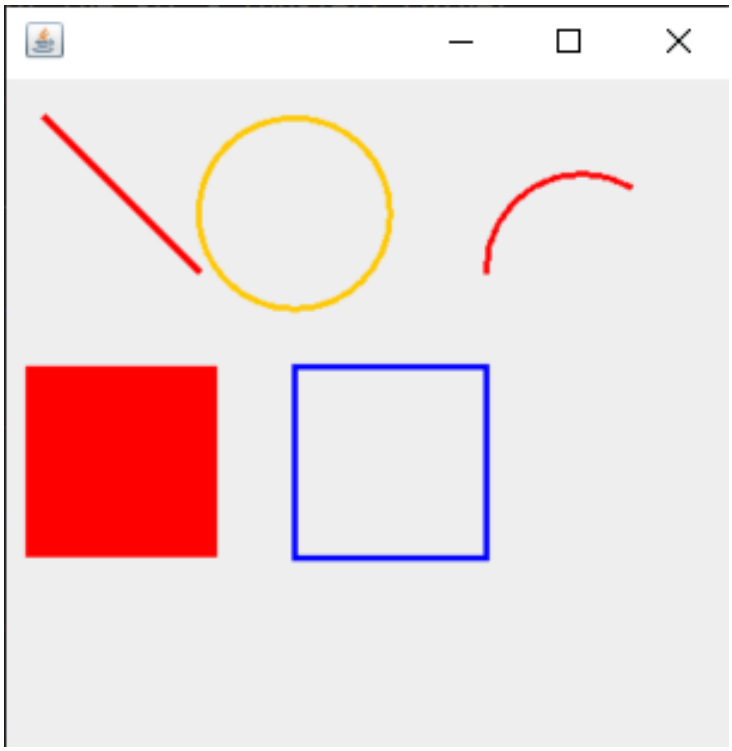
drawArc(int xTopLeft, int yTopLeft, int width, int height, int startAngle, int arcAngle): First four arguments same as drawOval() nest two arguments represents start angle and degrees around arc.




```
public void paintComponent(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillOval(50, 50, 150, 50);
}
```

```
import javax.swing.*;
import java.awt.*;
public class LineRectangle extends JPanel{
     public void paintComponent(Graphics g) {
          // Cast Graphics to Graphics2D
         Graphics2D g2d = (Graphics2D) g;
         // Set the stroke width (line width)
Stroke stroke = new BasicStroke(3); // line width 3 pixels
         g2d.setStroke(stroke);
          //draw red color line
          g.setColor(Color.RED);
          g.drawLine(20, 20, 100,100);

          //draw blue color rectangle
          g.setColor(Color.BLUE);
          g.drawRect(150, 150, 100, 100);

          //draw Orange color circle
          g.setColor(Color.ORANGE);
          g.drawOval(100, 20, 100, 100);

          //draw Filled color Rectangle
          g.setColor(Color.RED);
          g.fillRect(10, 150, 100, 100);

          //draw arc
          g.drawArc(250, 50, 100,100,60, 120);
     }
```

```
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setSize(400, 400);
        LineRectangle lr = new LineRectangle();
        f.add(lr);
        f.setVisible(true);
    }
}
```



6. Polygons

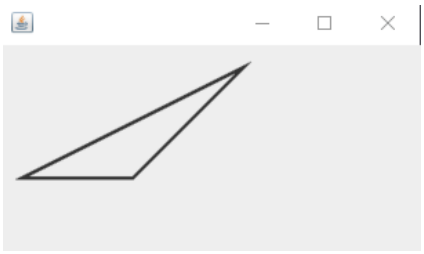
A polygon is considered as a set of lines connected together.

drawPolygon(int[] xPoints, int[] yPoints, int numPoints)

- int[] xPoints: An array of x coordinates of the polygon's vertices.
- int[] yPoints: An array of y coordinates of the polygon's vertices.
- int numPoints: The number of vertices in the polygon.

Polygon is closed by a line that connects the last point to the first.

```
public void paintComponent(Graphics g) {
          int x1[] = {120, 20, 220};
         int y1[] = {120, 120, 20};
         int n1 = 3;
         g.drawPolygon(x1, y1, n1);
}
```

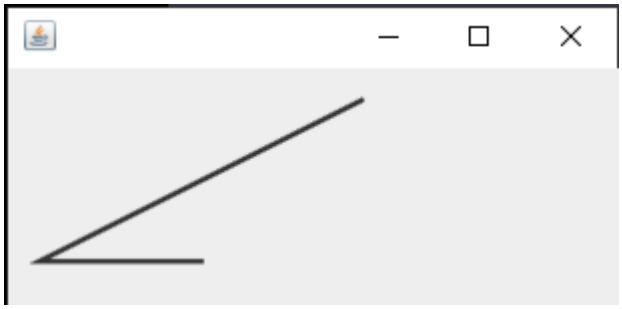


drawPolyline(int[] xPoints, int[] yPoints, int numPoints)

- int[] xPoints: An array of x coordinates of the polygon's vertices.
- int[] yPoints: An array of y coordinates of the polygon's vertices.
- int numPoints: The number of vertices in the polygon.

The polyline is not closed.

```
public void paintComponent(Graphics g) {
          int x1[] = {120, 20, 220};
         int y1[] = {120, 120, 20};
         int n1 = 3;
         g.drawPolyline(x1, y1, n1);
}
```



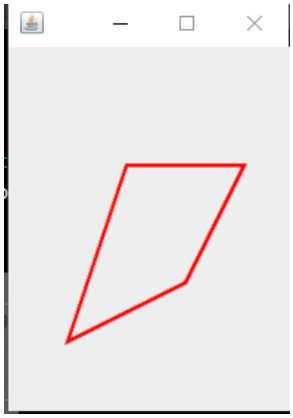
fillPolygon(int[] xPoints, int[] yPoints, int numPoints): Draws a filled polygon.

```
public void paintComponent(Graphics g) {
            int x1[] = {120, 20, 220};
           int y1[] = {120, 120, 20};
           int n1 = 3;
           g.fillPolygon(x1, y1, n1);
}
```

drawPolygon(Polygon p): Draws a Specified polygon.

```
public void paintComponent(Graphics g) {
     // Create a Polygon object
          Polygon polygon = new Polygon();
          polygon.addPoint(100, 100);
          polygon.addPoint(200, 100);
          polygon.addPoint(150, 200);
          polygon.addPoint(50, 250);
```

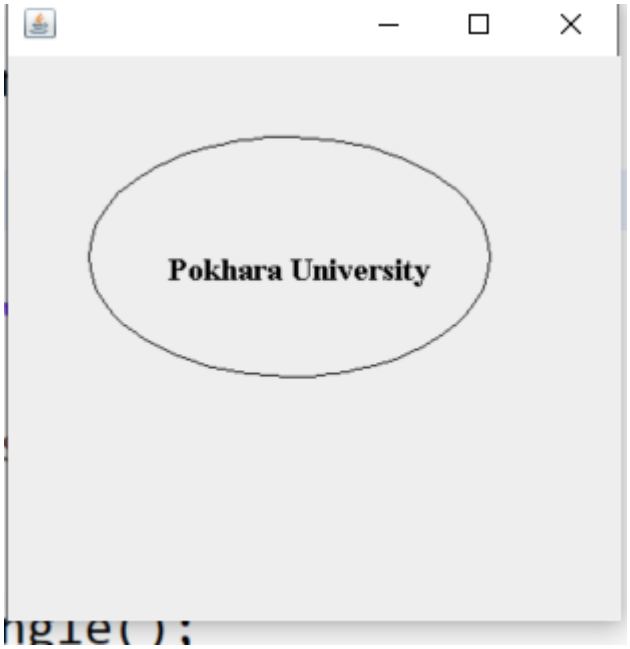
```
        // Draw the polygon
        g.setColor(Color.RED);
        g.drawPolygon(polygon);
}
```



Q. WAP to display "Pokhara Universit" inside an ellipse. Note that the string should be in serif font with size 20 and style bold.

```
import javax.swing.*;
import java.awt.*;
public class Example extends JPanel{

    public void paintComponent(Graphics g) {

        //draw ellipse
        g.setColor(Color.BLACK);
        g.drawOval(50, 50, 250, 150);

        //set font
        Font font = new Font("Serif", Font.BOLD, 20);
        g.setFont(font);
```

```
        //draw string
        g.drawString("Pokhara University", 100 , 140);
    }
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setSize(400, 400);
        Example ex = new Example();
        f.add(ex);
        f.setVisible(true);
    }
}
```

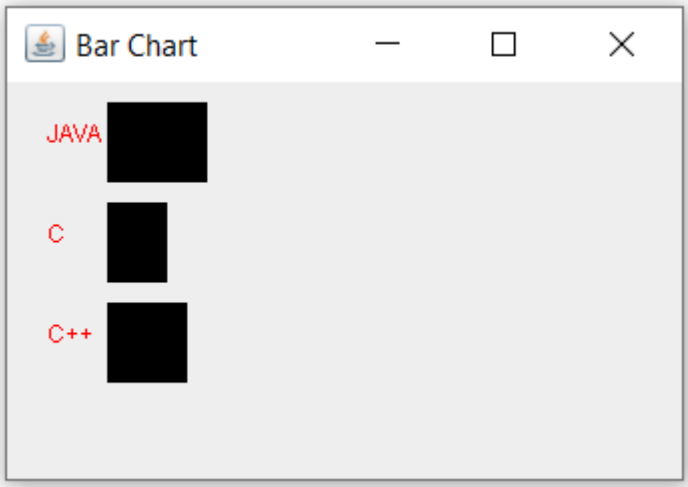


Q. WAP to draw a 2D rectangle in green color. Next draw the flag of Nepal inside the rectangle. Write a string "My Nepal" Below the rectangle.


```
import javax.swing.*;
import java.awt.*;
public class FlagOfNepal extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Draw a green rectangle
       g.setColor(Color.GREEN);
       g.fillRect(10, 10, 300, 300);

        // Draw the flag of Nepal
        int x[] = {20,200,100,240,20};
        int y[] = {20,150,150,300,300};
        g.setColor(Color.BLUE);
        g.fillPolygon(x, y, 5);
```

```
        //draw moon and sun
        g.setColor(Color.WHITE);
        g.fillArc(50, 80, 40, 40, 0, -180);
        g.fillOval(50, 200, 50, 50);

        //write my nepal
     Font font = new Font("Times New Roman", Font.BOLD, 30);
        g.setFont(font);
        g.setColor(Color.BLACK);
        g.drawString("My Nepal", 120, 330);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Flag of Nepal");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         FlagOfNepal nepalFlag = new FlagOfNepal();
         frame.add(nepalFlag);
          frame.setSize(600, 600);
          frame.setVisible(true);
    }
}
```

Q. WAP to draw a barchar of number of students giving exam for Java, C, C++.

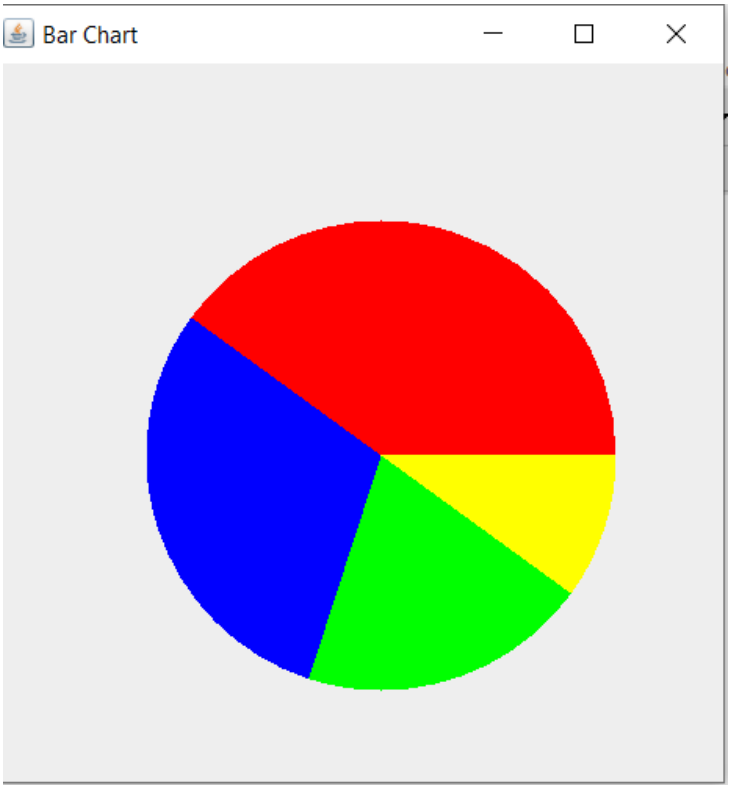



```
import javax.swing.*;
import java.awt.*;

public class BarChartExample  extends JPanel {
     int std[]  = {50, 30, 40};
     String sub[] = {"JAVA", "C", "C++"};

    public void paintComponent(Graphics g) {
        //draw bar chart
        for(int i=0;i<3;i++) {
          g.setColor(Color.red);
          g.drawString(sub[i], 20,30+i*50);
          g.setColor(Color.BLACK);
          g.fillRect(50, i*50+10, std[i], 40);
        }

    }
```

```
    public static void main(String[] args) {
        JFrame frame = new JFrame("Bar Chart");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BarChartExample  chart = new BarChartExample ();
        frame.add(chart);
        frame.setSize(800, 800);
        frame.setVisible(true);
    }
}
```

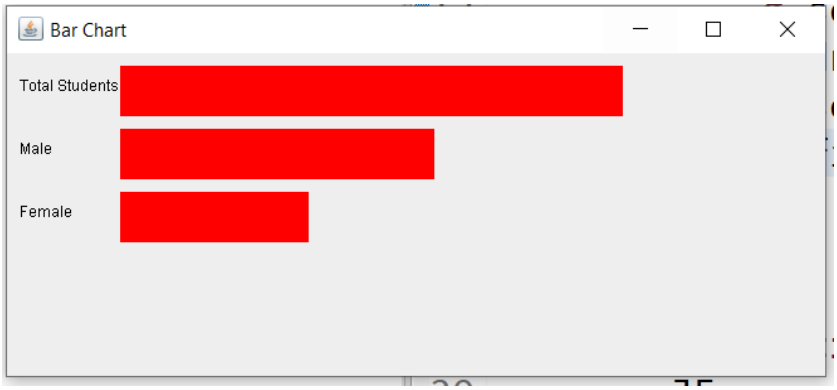
Q. As per a recent survey amopng all the javascript framrworkl 40% prefers react while angular comes second with 30% develop[ers prefering it. 20% prefer vue while 10% use other framework. Crate a pichart .

Q. WAP to draw piechar using AWT.



```
import java.awt.*;

public class PieChart extends Panel {
    public void paint(Graphics g) {

          int data[] = {40, 30, 20, 10};
Color  data_clr[]  =  {Color.RED,  Color.BLUE,  Color.GREEN,
Color.YELLOW};

            int total = 100;
            int start_angle = 0;

            for (int i = 0; i < data.length; i++) {
                int arc_angle = (int)(data[i]*360/total);
                g.setColor(data_clr[i]);
     g.fillArc(100, 100, 300, 300, start_angle, arc_angle);
                start_angle=start_angle+arc_angle;
          }

}
    public static void main(String[] args) {
        Frame frame = new Frame("Pie Chart Example");
        PieChart chart = new PieChart();
        frame.add(chart);
        frame.setSize(500, 500);
        frame.setVisible(true);
    }
}
```

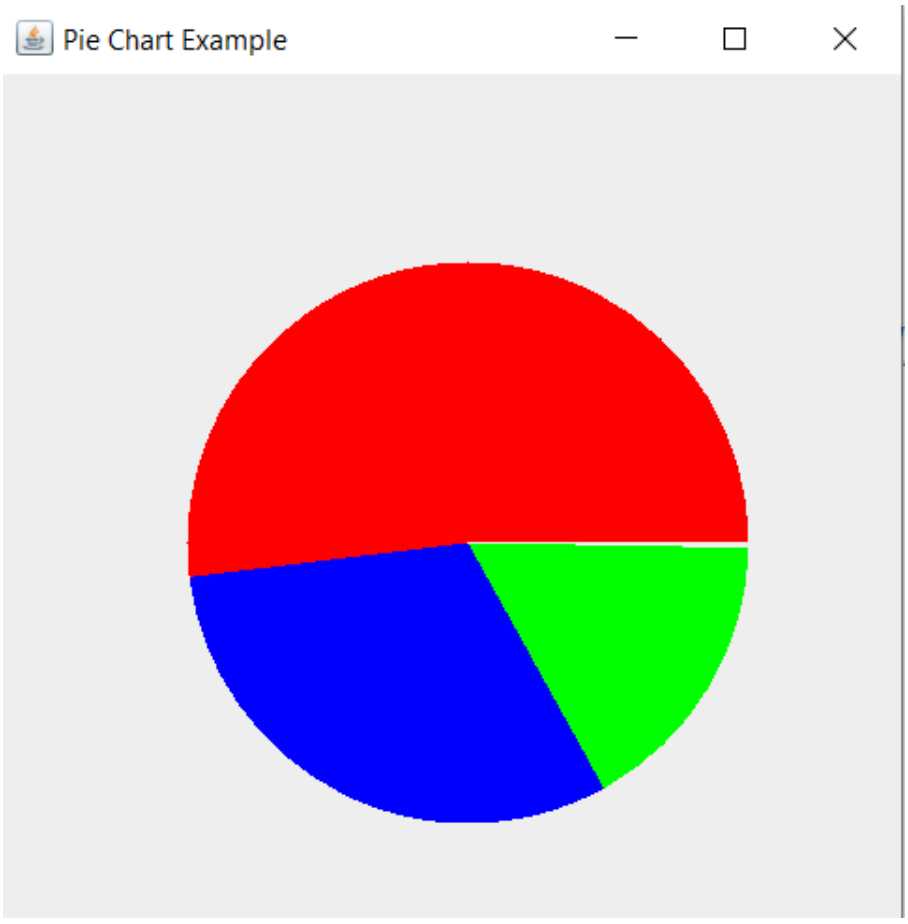
Q. WAP to draw a bar char of total number of students appearing exam in Java(400). Also provide bar chart for number of male(250) and felame(150) students.




```
import javax.swing.*;
import java.awt.*;

public class BarChartExample extends JPanel {
     int std[]  = {400,250, 150};
     String label[] = {"Total Students", "Male", "Female"};

    public void paintComponent(Graphics g) {
        //draw bar chart
        for(int i=0;i<3;i++) {
          g.setColor(Color.BLACK);
          g.drawString(label[i], 10,30+i*50);
          g.setColor(Color.RED);
          g.fillRect(90, i*50+10, std[i], 40);

        }

    }
```

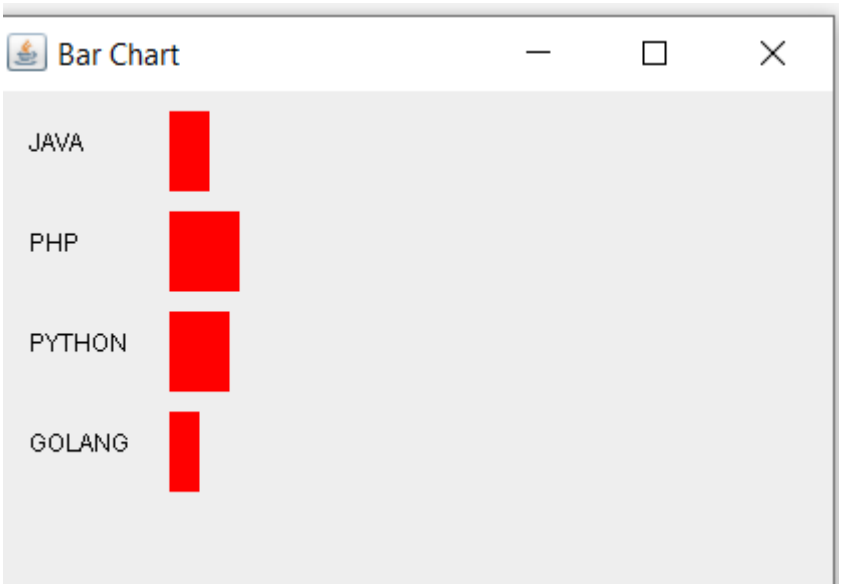
```
    public static void main(String[] args) {
        JFrame frame = new JFrame("Bar Chart");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Test  chart = new Test ();
        frame.add(chart);
        frame.setSize(800, 800);
        frame.setVisible(true);
    }
}
```

Q. As per a poll conducted in a class of 48 students, 25 students are supportive of virtual classroom while 15 are against it and remaing 8 students choose to stay netral. Create a pie chart to illustrate this poll result.



```
import java.awt.*;
import javax.swing.*;

public class PieChart extends JPanel {
public void paint(Graphics g) {
     int data[] = {25, 15, 8};
Color data_clr[] = {Color.RED, Color.BLUE, Color.GREEN};

                int total = 48;
                int start_angle = 0;

     for (int i = 0; i < data.length; i++) {
                int arc_angle = (int)(data[i]*360/total);
                g.setColor(data_clr[i]);
     g.fillArc(100, 100, 300, 300, start_angle, arc_angle);
                start_angle=start_angle+arc_angle;
                }
     }
public static void main(String[] args) {
             JFrame frame = new JFrame("Pie Chart Example");
             PieChart chart = new PieChart ();
             frame.add(chart);
             frame.setSize(500, 500);
             frame.setVisible(true);
         }
}
```

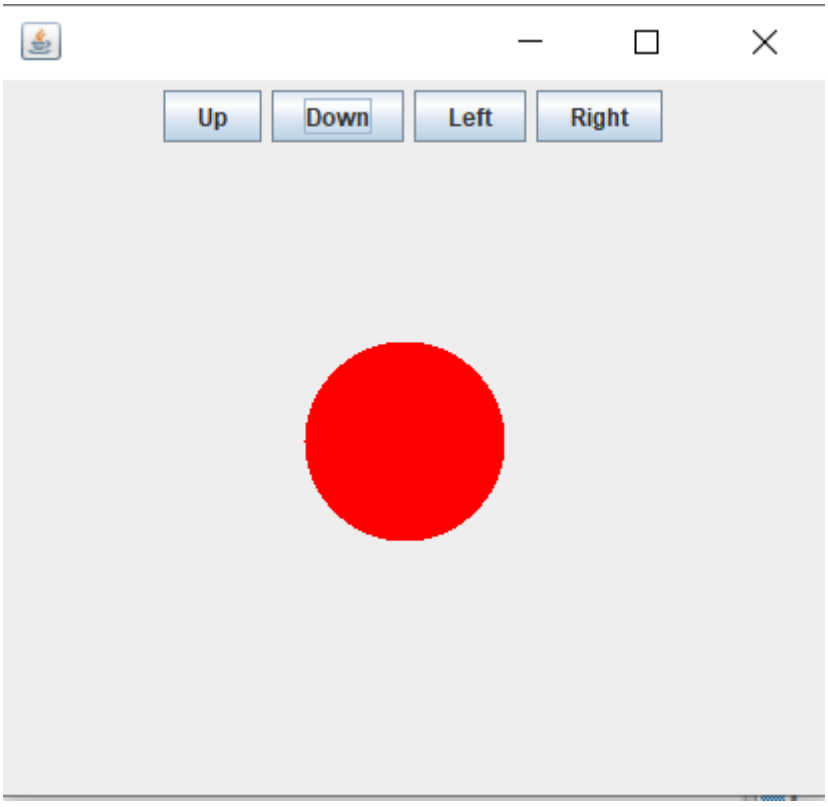
Q. In a class of 100 students 20 of them prefers java while 35 use php, 30 student use python and remaining uses Go lang as their preferred backend language. Illustarte this stat using a bar diagram




```
import javax.swing.*;
import java.awt.*;
public class BarChartExample extends JPanel {
     int data[] = {20, 35, 30, 15};
     String label[] = {"JAVA", "PHP", "PYTHON", "GOLANG"};
    public void paintComponent(Graphics g) {
        //draw bar chart
        for(int i=0;i<4;i++) {
          g.setColor(Color.BLACK);
          g.drawString(label[i], 20,30+i*50);
          g.setColor(Color.RED);
          g.fillRect(90, i*50+10, data[i], 40);
        }

    }
```

```
    public static void main(String[] args) {
        JFrame frame = new JFrame("Bar Chart");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BarChartExample chart = new BarChartExample ();
        frame.add(chart);
        frame.setSize(800, 800);
        frame.setVisible(true);
    }
}
```

Q. Create a frame with a red circular object in the centre. Also add four buttons named up, down, right and left. When a button is pressed and the ball should move in the direction corresponding to the name of button.



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class BallMovementApp extends JPanel implements
ActionListener {
     JButton up, left, down, right;
     int Ball_X=150, Ball_Y=150, Ball_diameter=100;

     public BallMovementApp() {

          //Create buttons
          up = new JButton("Up");
          down = new JButton("Down");
          right = new JButton("Right");
          left = new JButton("Left");

          //Add Event Listener
          up.addActionListener(this);
          down.addActionListener(this);
          right.addActionListener(this);
          left.addActionListener(this);

          //Add buttons to the frame
          add(up);
          add(down);
          add(left);
          add(right);
     }
```

```
        //draw ball
        public void paintComponent(Graphics g) {
         //for default rendering
         super.paintComponent(g);

         g.setColor(Color.RED);
g.fillOval(Ball_X, Ball_Y, Ball_diameter, Ball_diameter);
        }

   @Override
   public void actionPerformed(ActionEvent e) {

        if(e.getSource()==up) {
             Ball_Y = Ball_Y-10;
        }
        else if(e.getSource()==down) {
             Ball_Y  = Ball_Y+10;
        }
        else if(e.getSource()==left) {
             Ball_X = Ball_X -10;
        }
        else if(e.getSource()==right) {
             Ball_X = Ball_X +10;
        }
        //Redraw the shape
        repaint();
   }
```

```
    public static void main(String[] args) {
        JFrame f = new JFrame();
        f.setSize(400, 400);
        BallMovementApp lr = new BallMovementApp();
        f.add(lr);
        f.setVisible(true);
    }
}
```

CHAPTER: 7

NETWORK PROGRAMMING

NETWORK PRGRAMMING

Writing computer programmes that can interact and communicate with other computers through a network is referred to as network programming. Network programming is the practise of utilising programming languages and libraries to develop software that transfers data between various devices and processes it.

Java socket programming provides facility to share data between different computing devices.

JAVA NETWORKING TERMINOLOGY

The widely used Java networking terminologies are given below:

1. IP Address

An IP address is a unique numeric label assigned to each device (such as a computer, smartphone, or server) connected to a computer network. It serves two primary purposes: identifying the host or network interface and providing the location of the host in the network.

2. Port

A port is a 16-bit unsigned integer used to identify specific services or processes on a device within a network.

3. protocols

Protocols are rules and conventions that govern how data is transmitted and received over a network. They define the format, sequence, and meaning of data exchanged between devices or applications.

URL CLASS

URL or Uniform Resource Locator, is a reference or address used to access resources on the internet. It serves as a means to identify and locate a particular resource, such as a web page, a file, or a document.

A URL is divided into many sections:

https://www.example.com:8080/products/laptop?brand=apple&color=silver#reviews

In this example:

- **Scheme:** https
- **Host:** www.example.com
- **Port:** 8080 (the port in a URL is optional. If a port is not specified in a URL, the browser or client assumes the default port number associated with the chosen scheme (protocol))
- **Path:** /products/laptop
- **Query Parameters:** brand=apple&color=silver
- **Fragment Identifier:** reviews

This URL might represent a webpage about Apple laptops with silver color, and the fragment identifier directs the browser to a specific section about product reviews.

In Java, the **URL** class is part of the **java.net** package and is used to represent a Uniform Resource Locator (URL). The **URL** class in Java provides methods for creating, parsing, and manipulating URLs.

METHODS OF URL CLASS

- **getProtocol()**: Returns the protocol of the URL.
- **getHost()**: Returns the host (domain) of the URL.
- **getPort()**: Returns the port number of the URL.

- **getPath()**: Returns the path component of the URL.
- **getQuery()**: Returns the query string of the URL.
- **getRef()**: Returns the fragment identifier of the URL.
- **equals(Object obj):** Tests the equality of two URL objects.
- **hashCode():** Returns a hash code for the URL.
- **toURI():** Converts this URL to a URI object.

Q. What do you mean by URL? Provide a simple program using the URL class to show URL processing in java.

```
import java.net.*;
public class URLExample {
      public static void main(String[] args) {
            try {
                // Create a URL object
URL url = new   URL("https://www.youtube.com/results?search_query=nepal+news");
// Retrieve and print different components of the URL
System.out.println("Protocol: " + url.getProtocol());
System.out.println("Host: " + url.getHost());
System.out.println("Port: " + url.getPort());
System.out.println("Path: " + url.getPath());
System.out.println("Query: " + url.getQuery());
System.out.println("Fragment Identifier: " + url.getRef());
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

}
```

```
Protocol: https
Host: www.youtube.com
Port: -1
Path: /results
Query: search_query=nepal+news
Fragment Identifier: null
```

URLConnection CLASS

URLConnection class in Java is like a messenger that facilitates communication between a Java program and a website. It helps by opening a connection to a specific web address (URL), allowing the program to fetch information from the site.

1. **Create a URL Object:**

```
URL url = new URL("https://www.example.com");
```

2. **Open a Connection:**

```
URLConnection connection = url.openConnection();
```

3. **Get Information:**

```
InputStream inputStream = connection.getInputStream();
```

4. **Read the Information:**

5. **Close the connection**

Q. What do you mean by URL? WAP for reading contents in a given URL.

Q. Demonstrate URL Connection class method with suitable example program.

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

public class URLGetterExample {
      public static void main(String[] args) {
            try {
                // Create a URL object
             URL url = new URL("https://www.google.com/");
                // Crete URLConnection
                URLConnection uc = url.openConnection();

                InputStream is = uc.getInputStream();

                int i;
                while((i = is.read())!=-1) {
            // Displays Source code of the page
                  System.out.print((char)i);
                }

            } catch (Exception e) {
                e.printStackTrace();
         }
      }
}
```

Q. Differentiate between URL and URl class with suitable example?

URL: In Java, the URL class is used to represent and manipulate URLs. It supports the creation, parsing, and manipulation of URLs. This class does not link to the URL; instead, it is mostly used for URL manipulation.

URLConnection Class: The URLConnection class is used to connect to a URL and receive information about it, such as its content. It represents a channel of communication between the programme and a URL. It supports both reading and writing to the URL-specified resource

InetAddress class

Q. Explain InetAddress class?

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name *for example* www.javatpoint.com, www.google.com, www.facebook.com, etc.

METHODS OF *InetAddress* CLASS

getHostName(): *Returns the host name.*

getHostAddress(): *Returns the IP address string in textual presentation.*

getLocalHost(): (static method) *returns the instance of InetAdddress containing local host name and address.*

getByName(String host): *returns the instance of InetAddress representing the specified host name.*

getAllByName(String host): (static method) *returns an array of InetAddress objects that represent all the IP addresses associated with a given host name.*

Q. WAP to find local IP address and Hostname of system.

```
import java.net.InetAddress
public class Test {
    public static void main(String[] args) {
        try {
        InetAddress lH = InetAddress.getLocalHost();
System.out.println("Local Host Name: "+ lH.getHostName());
System.out.println("Local IP Add: "+lH.getHostAddress());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Local Host Name: DESKTOP-0T2N773
Local IP Add: 192.168.1.71
```

Q.WAP to print all ip address assoisated with "Netflix.com"

```
import java.net.*;
public class test {
    public static void main(String[] args) {
        try {
InetAddress[] add =
InetAddress.getAllByName("netflix.com");
System.out.println("IP addresses associated with
netflix.com:");
            for (InetAddress i : add) {
                System.out.println(i.getHostAddress());
            }
```

```
        } catch (UnknownHostException e) {
            e.printStackTrace();
        }
    }
}
```

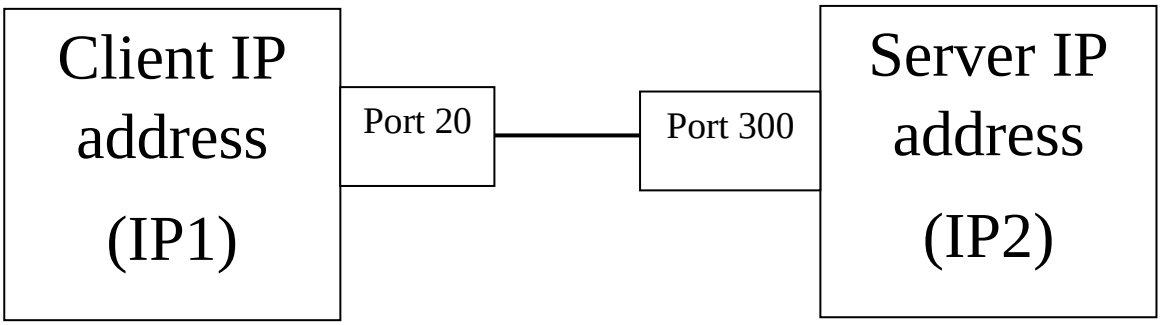
Q.WAP to print loopback ip address.

```
import java.net.*;
public class Test {
    public static void main(String[] args) {
        try {
            // Get the loopback address
  InetAddress lb= InetAddress.getLoopbackAddress();

            // Print the loopback address
System.out.println("Host Name: " + lb.getHostName());
System.out.println("Loopback IP:"+lb.getHostAddress());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
Loopback name: localhost
Loopback Address: 127.0.0.1
```

Socket Programming

Java programming, a socket is a mechanism that enables programs to establish a connection and communicate over a network. One end of the socket is bound to a specific port on one machine, while the other end is connected to a port on another machine. This connection allows data to be sent and received between the two programs.



Ip Address + port number = socket

Java Socket programming is used for communication between the applications running on different JRE. It can be connection-oriented(TCP) or connectionless(UDP).

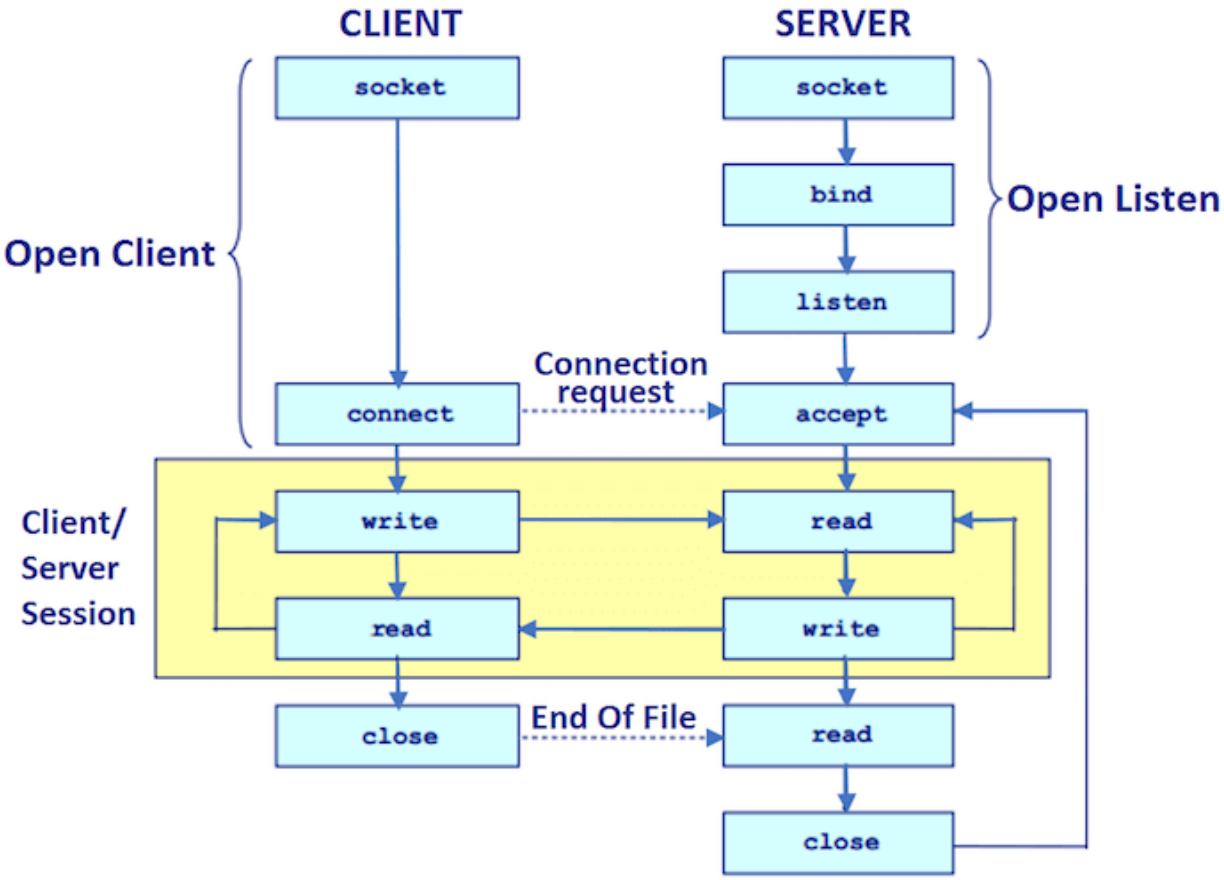
Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

TCP/IP server [Stream](Connection Oriented)

Transmission Control Protocol (TCP) is a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks.

Q. Draw a diagram with appropriate methods to describve client server interaction in java.

Q. Brifely explain the classes and methods that are used to create a TCP/IP Server application.



SERVER-SIDE (MyServer.java)

1. ServerSocket Class: Create a server socket listening on port.

(Think of **ServerSocket** as the server's ears waiting for someone to talk to. It creates a "hole" or a listening socket on a specific port, ready to accept incoming connections.)

```
ServerSocket serverSocket = new ServerSocket(port);
```

2. Socket Class: Listen for connection.

(When a client wants to communicate, it establishes a connection by "knocking on the door" of the server.The server, with its attentive ears, hears the knock and opens the door to establish a connection with the client.)

```
Socket clientSocket = serverSocket.accept();
```

3. Create input stream to read data from client: Get the InputStream from the Socket to read data from the client.

```
InputStream inputStream = clientSocket.getInputStream();
```

(This is like setting up a pipe or a channel to receive a continuous flow of water (data) from the client. It's a raw stream of bytes.)

3. Read and print from client: Read data from the input stream, e.g using a DataInputStream.

```
DataInputStream dis =new DataInputStream(inputStream);
```

(Now, imagine putting a specialized translator (DataInputStream) at the end of the pipe. This translator not only converts the raw water flow (bytes) into understandable language (characters) but also interprets the data according to specific data types (int, double, etc.)

4. Close all the connections.

```
clientSocket.close();
serverSocket.close();
    dis.close();
inputStream.close();
```

CLIENT-SIDE (MyClient.java)

1. Socket Class: Create a Socket object and connect to the server using its IP address and port.

```
Socket socket = new Socket("localhost",port);
```

2. Create output stream to send data to server: Get the OutputStream from the socket to send data to the server.

```
OutpututStream outputStream = socket.getOutputStream();
```

3. Send Data to server: Send data through the output stream, e.g. using a DataOutputStream.

```
DataOutputStream dos = new DataOutputStream(outputStream);
```

4. Close all the connections.

```
socket.close();
outputStream.close();
dos.close();
```

Q. Create a simple java socket programming where client sends a text and server receives and prints it.

(Myserver.java)

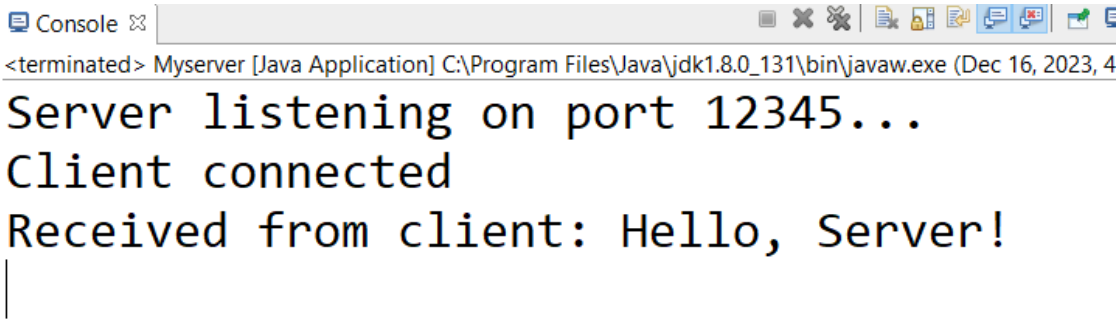
```
import java.io.*;
import java.net.*;
public class Myserver {
    public static void main(String[] args) {
        try {
           // Create a server socket listening on port 12345
   ServerSocket serverSocket = new ServerSocket(12345);
   System.out.println("Server listening on port 12345...");

            // Wait for a client to connect
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected");

         // Create input stream to read data from the client
  InputStream inputStream = clientSocket.getInputStream();
  DataInputStream dis = new DataInputStream(inputStream);

            // Read and print data from the client
       String clientMessage = dis.readUTF();
       System.out.println("From client: " + clientMessage);

            // Close the connection
            clientSocket.close();
            serverSocket.close();
            dis.close();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

(client.java)

```
import java.net.*;
import java.io.*;
public class client {
    public static void main(String[] args) {
        try {

        // Connect to the server on localhost and port 12345
            Socket socket = new Socket("localhost", 12345);

           // message to send to the server
         String messageToSend = "Hello, Server!";

          // Create output stream to send data to the server
      OutputStream outputStream = socket.getOutputStream();
DataOutputStream dos = new DataOutputStream(outputStream);
           dos.writeUTF(messageToSend);
            dos.flush();

            // Close the connection
            socket.close();
             dos.close();
        }
```

```
            catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```



Q. Create a TCP clinet/server program in which client sends an integer to the server and the server responds to client by sending square of the number sent by client.

Q. Create a TCP client/server program in which client and server communicate by sending a message to each other.

(MyServer.java)

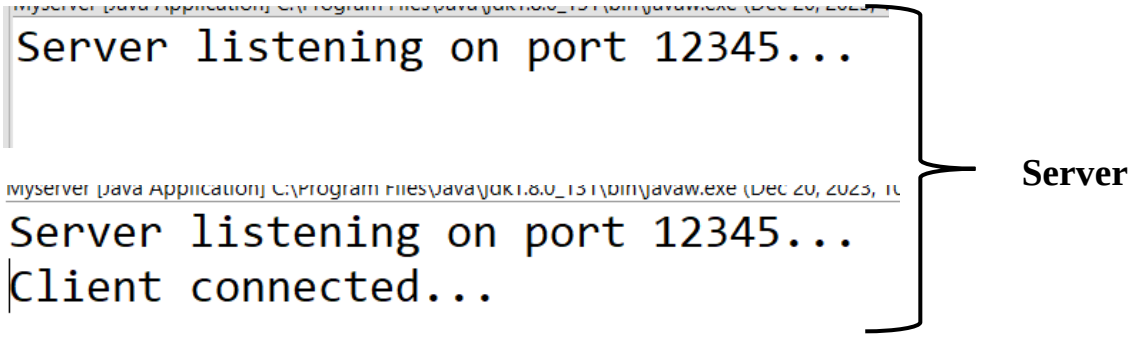
```
import java.io.*;
import java.net.*;
public class Myserver {
    public static void main(String[] args) {
        try {

            // Create a server socket listening on port 12345
   ServerSocket serverSocket = new ServerSocket(12345);
   System.out.println("Server listening on port 12345...");

            // Wait for a client to connect
```

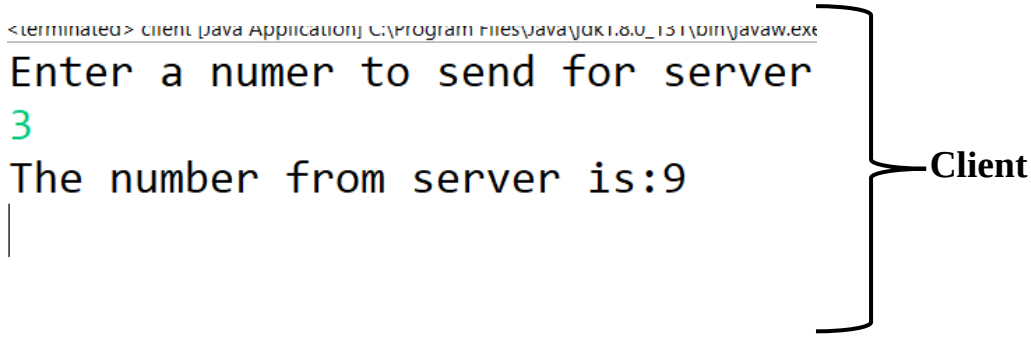
```
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected...");

          // Create input stream to read data from the client
  InputStream inputStream = clientSocket.getInputStream();
  DataInputStream dis = new DataInputStream(inputStream);

     //create output stream to send number to client
OutputStream outputStream = clientSocket.getOutputStream();
 DataOutputStream dos = new DataOutputStream(outputStream);

            // Read  number from the client
                  int num = dis.readInt();
              //send square of number to client
                  dos.writeInt(num*num);

            // Close the connection
            clientSocket.close();
            serverSocket.close();
            dis.close();
             dos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

(MyClient.java)

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class client {
    public static void main(String[] args) {
        try {

        // Connect to the server on localhost and port 12345
            Socket socket = new Socket("localhost", 12345);

           // number to send to the server
            Scanner sc = new Scanner(System.in);
    System.out.println("Enter a numer to send for server:");
            int num = sc.nextInt();

          // Create output stream to send data to the server
      OutputStream outputStream = socket.getOutputStream();
 DataOutputStream dos = new DataOutputStream(outputStream);

         // Create input stream to read data from the server
       InputStream inputStream = socket.getInputStream();
    DataInputStream dis = new DataInputStream(inputStream);

         //send number to server
           dos.writeInt(num);
            dos.flush();
```

```
            //receive square of number from client
              int numSquare = dis.readInt();
System.out.println("The number from server is:"+numSquare);

              // Close the connection
              socket.close();
          }
           catch (IOException e) {
              e.printStackTrace();
          }
      }
}
```





Q. Continuous chat app between server and client.

(MyServer.java)

```
import java.io.*;
import java.net.*;
public class Myserver {
    public static void main(String[] args) {
        try {

            // Create a server socket listening on port 12345
   ServerSocket serverSocket = new ServerSocket(12345);
   System.out.println("Server listening on port 12345...");

            // Wait for a client to connect
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected...");

      // Create input stream to read data from the client
  InputStream inputStream = clientSocket.getInputStream();
  DataInputStream dis = new DataInputStream(inputStream);

    //create output stream to send number to client
OutputStream outputStream = clientSocket.getOutputStream();
 DataOutputStream dos = new DataOutputStream(outputStream);

    //Continuous read from the client and send data
            String str;
            while(true) {
                str = dis.readUTF();
```

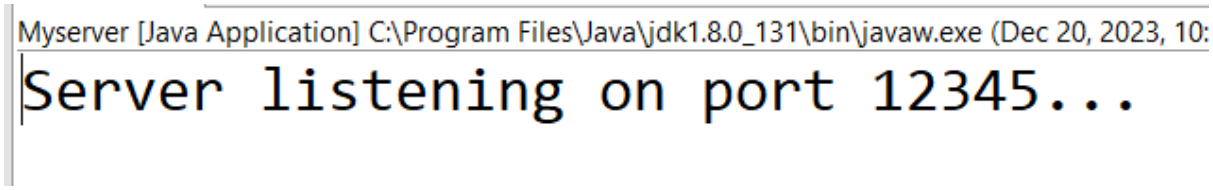
```
            dos.writeUTF("server: Hellow from server......");
                    if(str.equalsIgnoreCase("bye")) {
                        break;
                    }
                }
                dos.writeUTF("exit");
            System.out.println("Server Disconnected!!!!");
               // Close the connection
               clientSocket.close();
               serverSocket.close();
               dis.close();
               dos.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```

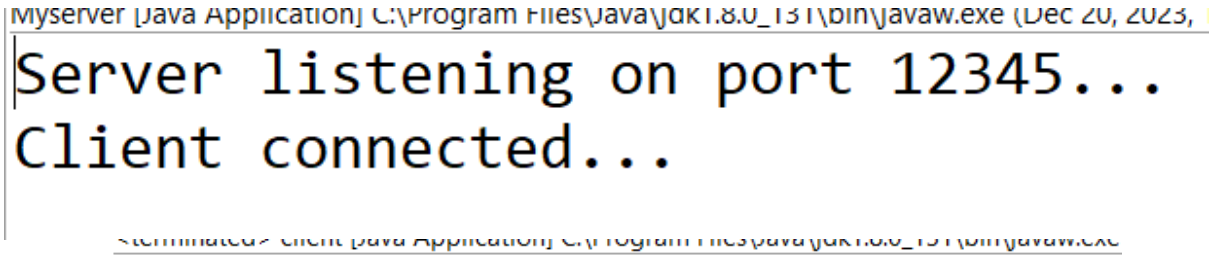
(MyClient.java)

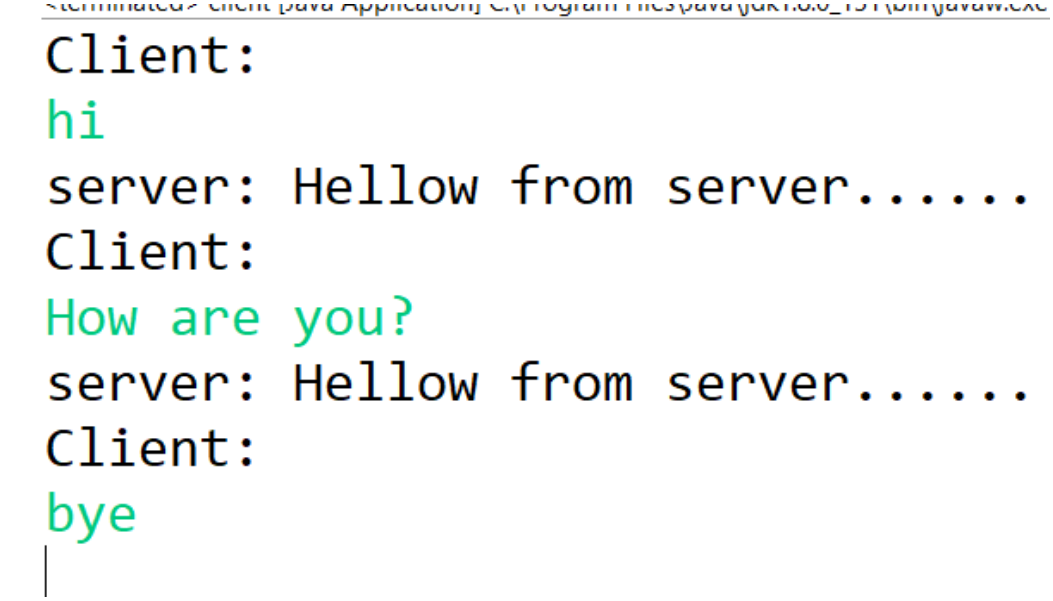
```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class client {
    public static void main(String[] args) {
        try {

        // Connect to the server on localhost and port 12345
            Socket socket = new Socket("localhost", 12345);
```

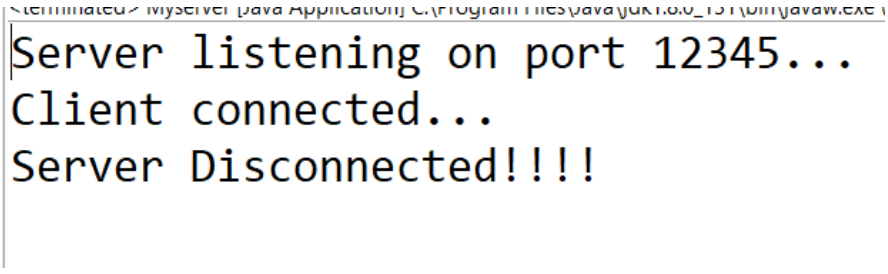
```
            // Create output stream to send data to the server
        OutputStream outputStream = socket.getOutputStream();
    DataOutputStream dos = new DataOutputStream(outputStream);

        // Create input stream to read data from the server
        InputStream inputStream = socket.getInputStream();
      DataInputStream dis = new DataInputStream(inputStream);

           //Continously send data to server
        Scanner sc = new Scanner(System.in);
        String str1, str2;
        while(true) {
                System.out.println("Client: ");
                str1 = sc.nextLine();
                dos.writeUTF(str1);
                dos.flush();
                str2 = dis.readUTF();
               if(str2.equalsIgnoreCase("exit")) {
                break;
               }
               System.out.println(str2);
        }
          // Close the connection
               socket.close();
               dis.close();
               dos.close();
          }
```

```
            catch (IOException e) {
                e.printStackTrace();
            }
        }
}
```









Multi Threaded TCP/IP Server(Each client sends a number to server. Server sends square of number to each client)

(Myserver.java)

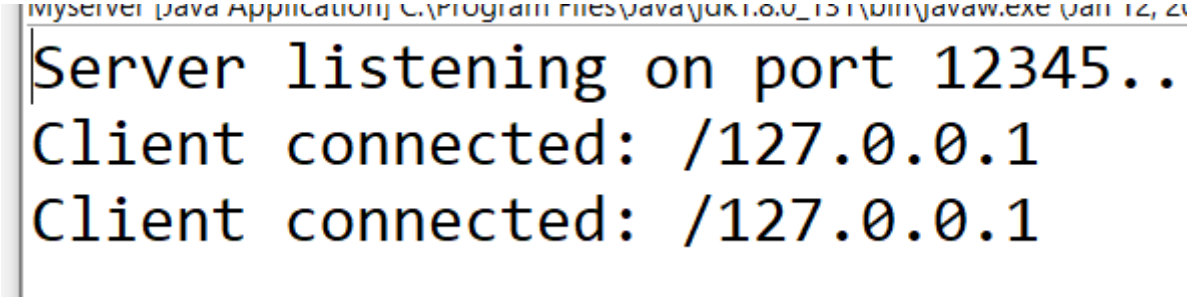
```
import java.io.*;
import java.net.*;
class ServerThread extends Thread {
    private Socket clientSocket;
    public ServerThread(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }
    public void run() {
        try {
 System.out.println("Client connected: " +
clientSocket.getInetAddress());

    // Create input stream to read data from the client
  InputStream inputStream = clientSocket.getInputStream();
DataInputStream dis = new DataInputStream(inputStream);

 // Create output stream to send number to client
OutputStream outputStream = clientSocket.getOutputStream();
DataOutputStream dos = new DataOutputStream(outputStream);

          // Read number from the client
             int num = dis.readInt();
             // Send square of the number to the client
             dos.writeInt(num * num);
             dos.flush();
```

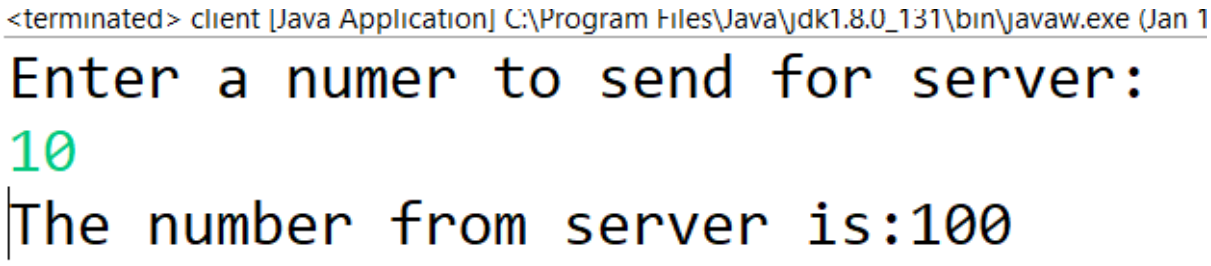
```
            // Close the connection for this client
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public class Myserver {
    public static void main(String[] args) {
        try {
       // Create a server socket listening on port 12345
      ServerSocket serverSocket = new ServerSocket(12345);
  System.out.println("Server listening on port 12345...");
     while (true) {
                // Wait for a client to connect
         Socket clientSocket = serverSocket.accept();

         // Start a new thread to handle the client
ServerThread serverThread = new ServerThread(clientSocket);
                serverThread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

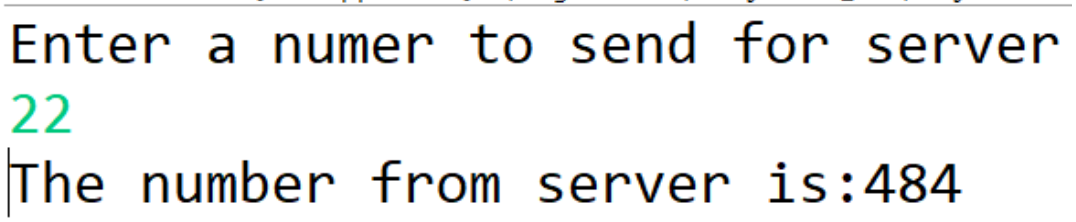
(Client.java)

```
import java.net.*;
import java.util.Scanner;
import java.io.*;
public class client {
    public static void main(String[] args) {
        try {

        // Connect to the server on localhost and port 12345
            Socket socket = new Socket("localhost", 12345);

           // number to send to the server
            Scanner sc = new Scanner(System.in);
    System.out.println("Enter a numer to send for server:");
            int num = sc.nextInt();

          // Create output stream to send data to the server
      OutputStream outputStream = socket.getOutputStream();
 DataOutputStream dos = new DataOutputStream(outputStream);

         // Create input stream to read data from the server
       InputStream inputStream = socket.getInputStream();
    DataInputStream dis = new DataInputStream(inputStream);

          //send number to server
           dos.writeInt(num);
            dos.flush();
```

```
            //receive square of number from client
              int numSquare = dis.readInt();
System.out.println("The number from server is:"+numSquare);

              // Close the connection
              socket.close();
          }
           catch (IOException e) {
              e.printStackTrace();
          }
      }
}
```







UDP [datagram](Connectionless)

Java **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming using the UDP instead of TCP.

Commonly used Constructors of DatagramPacket class

DatagramPacket(byte[ ] arr, int length): it creates a datagram packet. This constructor is used to receive the packets.

DatagramPacket(byte[ ] arr, int length, InetAddress address, int port): it creates a datagram packet. This constructor is used to send the packets.

(MyServer.java)

```
import java.net.*;
public class Myserver {
    public static void main(String[] args) {
        try {
     // Create a DatagramSocket to listen on a specific port
  DatagramSocket serverSocket = new DatagramSocket(12345);
 System.out.println("Server is listening on port 12345..");

          // Receive client message
          byte[] receiveData = new byte[1024];
DatagramPacket receivePacket = new DatagramPacket(receiveData,
1024);
          serverSocket.receive(receivePacket);
          // Extract client message and print
String clientMsg = new String(receivePacket.getData(),0,
receivePacket.getLength());
          System.out.println("From Client: "+clientMsg);
```

```
            // Close the socket
            serverSocket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

(MyClient.java)

```
import java.net.*;
public class MyClient {
    public static void main(String[] args) {
      try {
            // Create a DatagramSocket for client
       DatagramSocket clientSocket = new DatagramSocket();

            // Message to send to the server
            String messageToSend = "Hello, Server!";
            byte[] sendData = messageToSend.getBytes();

            // Create DatagramPacket to send data
DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.getlength(), InetAddress.getByName("localhost"),
12345);


            // Send the packet
            clientSocket.send(sendPacket);
```

```
            // Close the socket
            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

<terminated> MyServer [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (Dec 31, 2023, 11

```
Server is listening on port 12345....
From Client: Hello, Server!
```

Q. Difference between TCP(Stream Communication) and UDP(Byte/Datagram Communication)

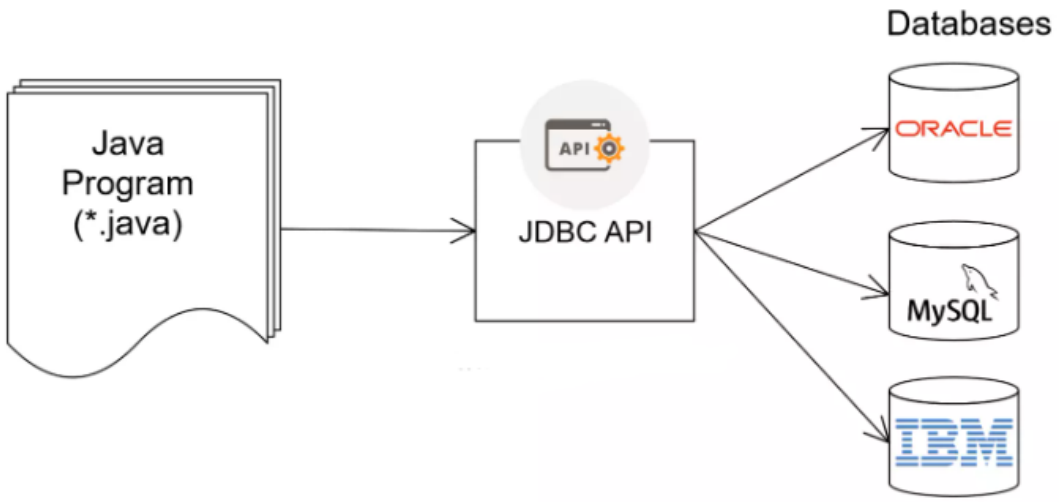
TCP	UDP
Requires an established connection before transmitting data	No connection is needed to start and end a data transfer
Delivery is guaranteed	Delivery is not guaranteed
TCP rearranges data packets in the specific order	UDP protocol has no fixed order because all packets are independent of each other.
TCP does error checking and also makes error recovery.	UDP performs error checking, but it discards erroneous packets.
Header size is 20 bytes	Header size is 8 bytes.
Examples: https, FTP, SMTP	Example: DNS, DHCP

CHAPTER: 8

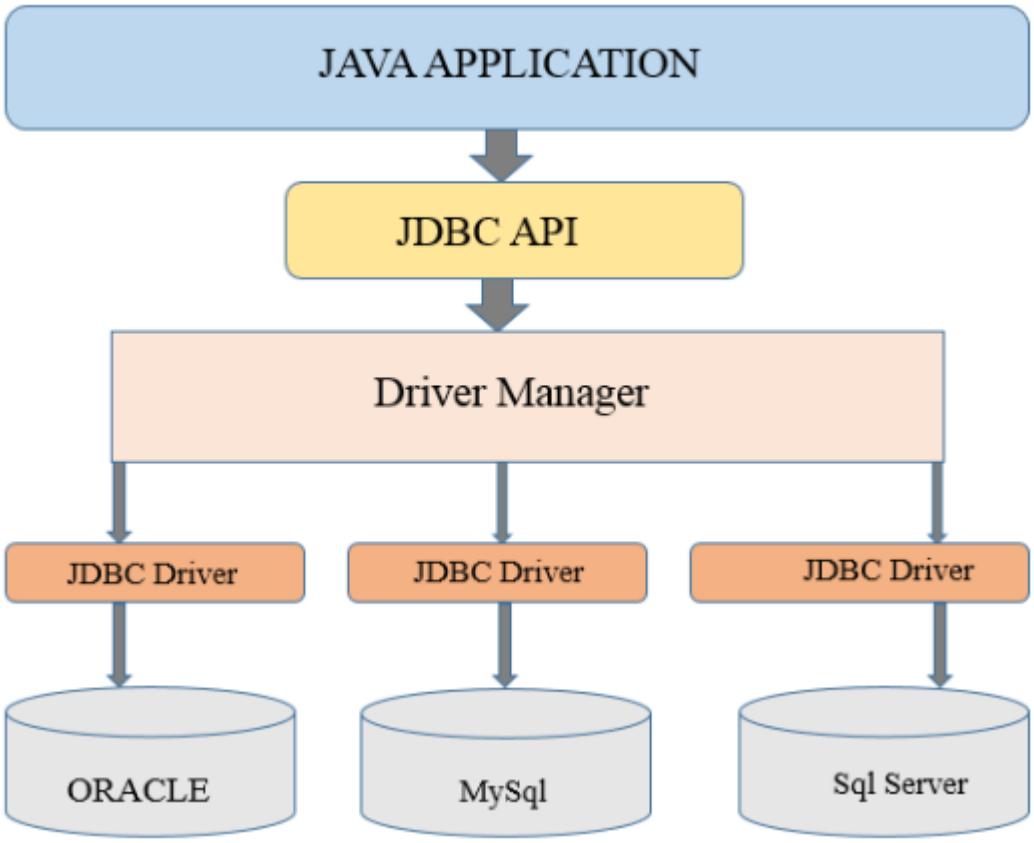
JDBC

JDBC

JDBC stands for **Java Database Connectivity. JDBC** is a **Java API** to connect and execute the query with the database. Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the **ODBC(Open database connectivity) driver**. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.



JDBC ARCHITECTURE



1. Application: Application in JDBC is a Java applet or a Servlet that communicates with a data source.

2. JDBC API: JDBC API provides classes, methods, and interfaces that allow Java programs to execute SQL statements and retrieve results from the database. Some important classes and interfaces defined in JDBC API are:

- DriverManager
- Driver
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- SQL data

3. Driver manager: When a database connection is requested, **DriverManager** helps in selecting an appropriate driver from the available ones.

4. Driver: Each JDBC driver translates JDBC calls into database-specific calls. It is the driver's responsibility to communicate with the database and perform the necessary operations.

JDBC DRIVER

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

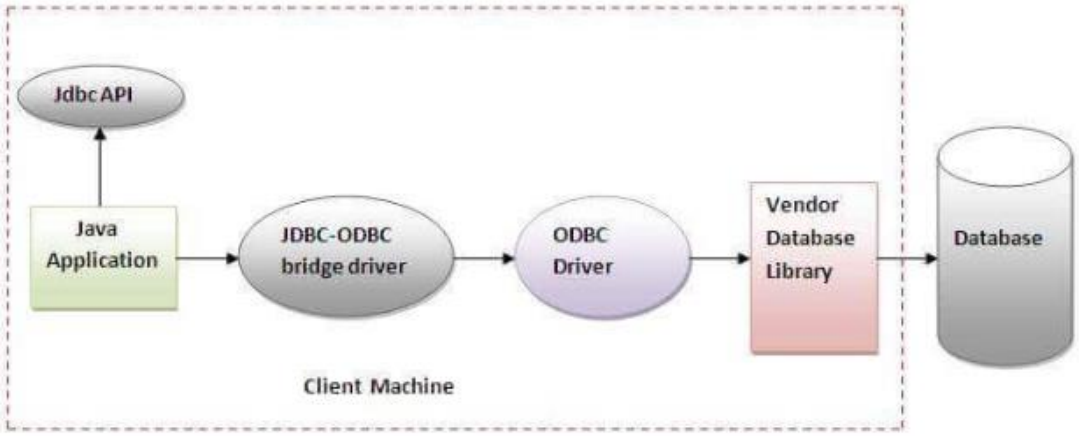


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that we use JDBC drivers provided by the vendor of our database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

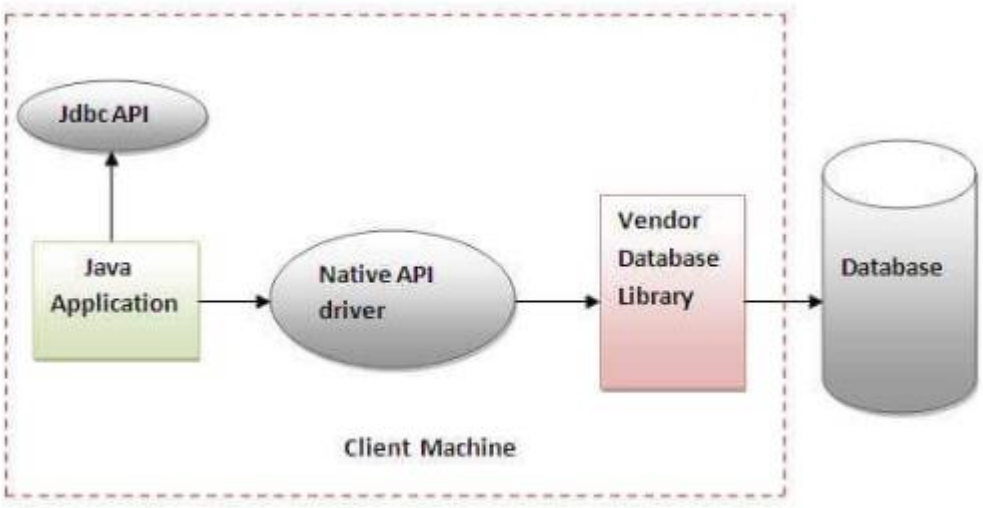


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

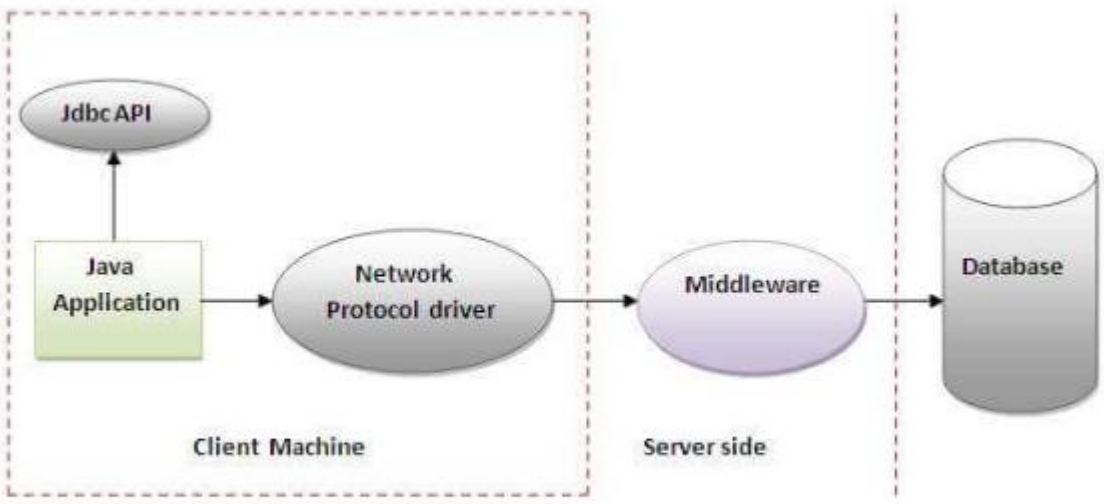


Figure- Network Protocol Driver

Advantage:

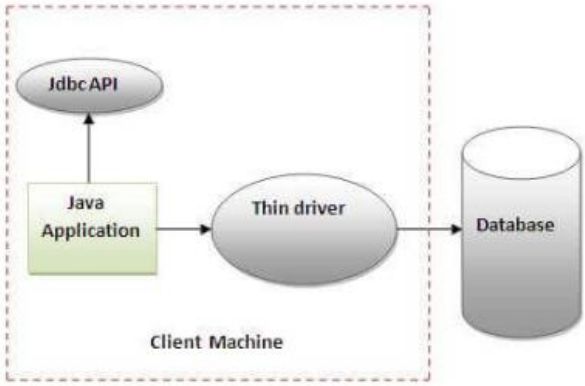
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.



Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

Q. Explain different types of JDBC drivers.

Steps to connect to database

1. Import JDBC Packages: `java.sql.*;`

2. Load and Register JDBC Driver.(for mysql)

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

3. Establish a Connection(Connection is an interface).

```
Connection con = DriverManager.getConnection(url, uname, pass);
```

4. Create a Statement: Statements are used to execute SQL queries.

Statement: Used for executing simple SQL queries where values are directly included in the query string.

```
String query = "select *from details where roll=95";

Statement smt = con.createStatement();
```

PreparedStaement: Used to execute SQL queries with placeholders for parameters.

```
String query = "select *from details where roll=?";

PreparedStatement psmt = con.prepareStatement(query);

psmt.setString(1, “Ankit”);
```

5. Execute SQL Queries.

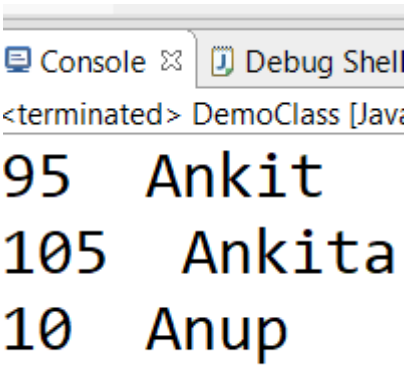
executeQuery() : Executes a SELECT SQL query and returns a ResultSet.

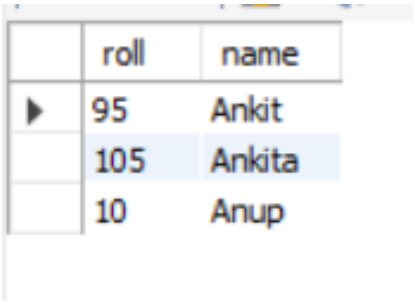
executeUpdate() : Executes an SQL UPDATE, INSERT, or DELETE query

6. Close Connection: Close all the connections.

Q. WAP to connect a databse using JDBC and display all records .Assume that the database name is Student and it has a table named Details with 2 colums "name" and "roll".

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {
      String url = "jdbc:mysql://localhost:3306/student";
      String uname = "root";
      String pass = "ankit@12345";
      String query = "SELECT *FROM details";
     Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(url,  uname,
pass);
     Statement smt = con.createStatement();
     ResultSet rset = smt.executeQuery(query);

     while(rset.next()) {
System.out.println(rset.getInt(1)+ "  "+rset.getString(2));
     }
     smt.close();
     con.close();
}
}
```





roll	name
95	Ankit
105	Ankita
10	Anup

Q. A databse *db_ std* contains a table *tbl_std* having field *roll, name, faculty and mark*. WAP that does following: Print the total number of students and find average marks.

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {
      String url = "jdbc:mysql://localhost:3306/db_std";
      String uname = "root";
      String pass = "root";
      String query = "SELECT *FROM tbl_std";
     Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(url,  uname,
pass);
     Statement smt = con.createStatement();
     ResultSet rset = smt.executeQuery(query);
     int count=0;
     float sum=0, avg;
     while(rset.next()) {
          count++;
          sum = sum + rset.getFloat("marks");
     }
     avg = sum/count;
     System.out.println("The Average marks: "+avg);
     System.out.println("The total students: "+count);
     smt.close();
     con.close();

}
}
```

Q. A database "pu" contains a table "Employee". WAP that asks user to enter a name, then query the databse and display total number of employess with that name.

```
import java.sql.*;
import java.util.Scanner;
public class DemoClass {
 public static void main(String[] args) throws Exception {

      String url = "jdbc:mysql://localhost:3306/pu";
      String uname = "root";
      String pass = "root";
     String query = "SELECT *FROM Employee WHERE name=?";
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection  con  =  DriverManager.getConnection(url,
     uname, pass);

     String name;
     Scanner sc = new Scanner(System.in);
     System.out.println("Enter name of student:");
     name = sc.next();
     PreparedStatement smt = con.prepareStatement(query);
     smt.setString(1, name);
     ResultSet rset = smt.executeQuery();

     int count=0;
     while(rset.next()) {
          count++;
     }
```

```
        System.out.println("Total employess with name "+name+"
        is:"+count);
        smt.close();
        con.close();
}
}
```

Q. A database "testdb" contains a table "Student" with fields id, name and address. WAP that asks user to enter the information and save in the databse. The program should prompt the user to pressy/n to either continue or exit program after every successful entry.

```
import java.sql.*;
import java.util.Scanner;
public class DemoClass {
 public static void main(String[] args) throws Exception {

        String url = "jdbc:mysql://localhost:3306/testdb";
        String uname = "root";
        String pass = "root";
       String query = "INSERT INTO student VALUES(?,?,?)";
       Class.forName("com.mysql.cj.jdbc.Driver");
       Connection   con   =   DriverManager.getConnection(url,
       uname, "");
       PreparedStatement smt = con.prepareStatement(query);

       String name, add;
       int id;
       Scanner sc = new Scanner(System.in);
```

```
    while(true) {
        System.out.println("Enter name of student:");
        name = sc.next();
        System.out.println("Enter ID of student:");
        id = sc.nextInt();
        System.out.println("Enter Address of student:");
        add = sc.next();

        smt.setInt(1, id);
        smt.setString(2, name);
        smt.setString(3, add);
        int row = smt.executeUpdate();

System.out.println("Number of rows affected: "+row);
  System.out.println("Do you wish to continue(Y/N): ");
    char ch = sc.next().charAt(0);
        if(ch!='y') {
            break;
        }
    }
    System.out.println("Program terminated!!!!!");
    smt.close();
    con.close();

}
}
```

Q. A database "testdb" contains a table "Employee" with some records having id, name, post, salary. WAP to update the salary to 50000 whose post is Manager.

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {

      String url = "jdbc:mysql://localhost:3306/testdb";
      String uname = "root";
      String pass = "root";
     String query = "UPDATE student SET salary=50000 WHERE
     post = 'Manager'";
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection con = DriverManager.getConnection(url,
     uname, pass);

     Statement smt = con.createStatement();
     int rows = smt.executeUpdate(query);
     System.out.println("Total rows affected "+rows);
     smt.close();
     con.close();
}
}
```

Q. WAP to steps to insert data on following table:

Table name : student

Column	DataType
Id	number
Name	varchar
Roll	number

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {

      String url = "jdbc:mysql://localhost:3306/db_std";
      String uname = "root";
      String pass = "root";
     String query = "INSERT INTO student VALUES(?,?,?)";
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection con = DriverManager.getConnection(url,
     uname, pass);
     PreparedStatement smt = con.prepareStatement(query);
          smt.setInt(1, 56);
          smt.setString(2, “Ankit Kharel”);
          smt.setInt(3, 1);
          int row = smt.executeUpdate();
System.out.println("Number of rows affected: "+row);
     smt.close();
     con.close();

}
}
```

Q. WAP to connect ms access databse and insert data in the table named "Student " with five fields named id, name, address, dob and class.

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {


Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
      String url =
"jdbc:ucanaccess://c:\\users\\dekstop\\std_db.accdb";
      Connection con = DriverManager.getConnection(url);

String query = "INSERT INTO Student VALUES (?, ?, ?, ?, ?)";
     PreparedStatement psmt = con.prepareStatement(query);
     psmt.setInt(1, 95);
     psmt.setString(2, "Ankit Kharel");
     psmt.setString(3, "Ktm");
     psmt.setString(4, "2052-04-28");
     psmt.setString(5, "BEIT");
     int row = psmt.executeUpdate();
     System.out.println("Total Rows Affeted:"+row);
     psmt.close();
     con.close();
}
}
```

Q.What is resultsetmetadata. Provide a sample program to illustrate.

The **ResultSetMetaData** is an interface that provides information about the obtained ResultSet object like, the number of columns, names of the columns, datatypes of the columns, name of the table etc.

Following are some methods of **ResultSetMetaData** .

Method	Description
getColumnCount()	Retrieves the number of columns in the current ResultSet object.
getColumnTypeName ()	Returns the database specific datatype of the column.
getColumnName()	Retrieves the name of the column.
getTableName()	Retrieves the name of the table.

```
import java.sql.*;
public class DemoClass {
 public static void main(String[] args) throws Exception {

      String url = "jdbc:mysql://localhost:3306/testdb";
      String uname = "root";
      String pass = "root";
     String query = "select *from student";
     Class.forName("com.mysql.cj.jdbc.Driver");
     Connection con = DriverManager.getConnection(url,
     uname, pass);
     Statement smt = con.createStatement();
     ResultSet rset = smt.executeQuery(query);
```

```
        ResultSetMetaData rsmd = rset.getMetaData();

        System.out.println("Total Columns: "
        +rsmd.getColumnCount());
        System.out.println("First Column Name: ”
        +rsmd.getColumnName(1));
        System.out.println("Data type: "
        +rsmd.getColumnTypeName(1));
        System.out.println("Second Column Name: ”
        +rsmd.getColumnName(2));
        System.out.println("Third Column Name: ”
        +rsmd.getColumnName(3));

        smt.close();
        con.close();

}
}
```

Q. Difference between normal statement and prepared statement.