Sahil Vahora

Q-1) Interpreter Vs Compiler.

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
Interpreters usually take less amount of time to analyse the source code. However, the overall execution time is comparatively slower than compilers.	Compilers usually take a large amount of time to analyse the source code. However, the overall execution time is comparatively faster than interpreters.
No Object Code is generated, hence are memory efficient.	Generates Object Code which further requires linking, hence requires more memory.
Programming languages like JavaScript, Python, and Ruby use interpreters.	Programming languages like C, C++, and Java use compilers.

Q-2) Procedural Programming Vs Object-oriented programming.

No.	On the basis of	Procedural Programming	Object-oriented programming
1.	Definition	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
2.	Security	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3.	Approach	It follows a top-down approach.	It follows a bottom-up approach.

4.	Data movement	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5.	Orientation	It is structure/procedure-oriented.	It is object-oriented.
6.	Access modifiers	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7.	Inheritance	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8.	Code reusability	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
9.	Overloading	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
10.	Importance	It gives importance to functions over data.	It gives importance to data over functions.
11.	Virtual class	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12.	Complex problems	It is not appropriate for complex problems.	It is appropriate for complex problems.
13.	Data hiding	There is not any proper way for data hiding.	There is a possibility of data hiding.
14.	Program division	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
15.	Examples	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

Q-3) Explain JDK, JVM, API, IDE, JIT.

Java Virtual Machine (JVM) is an abstract computing machine.

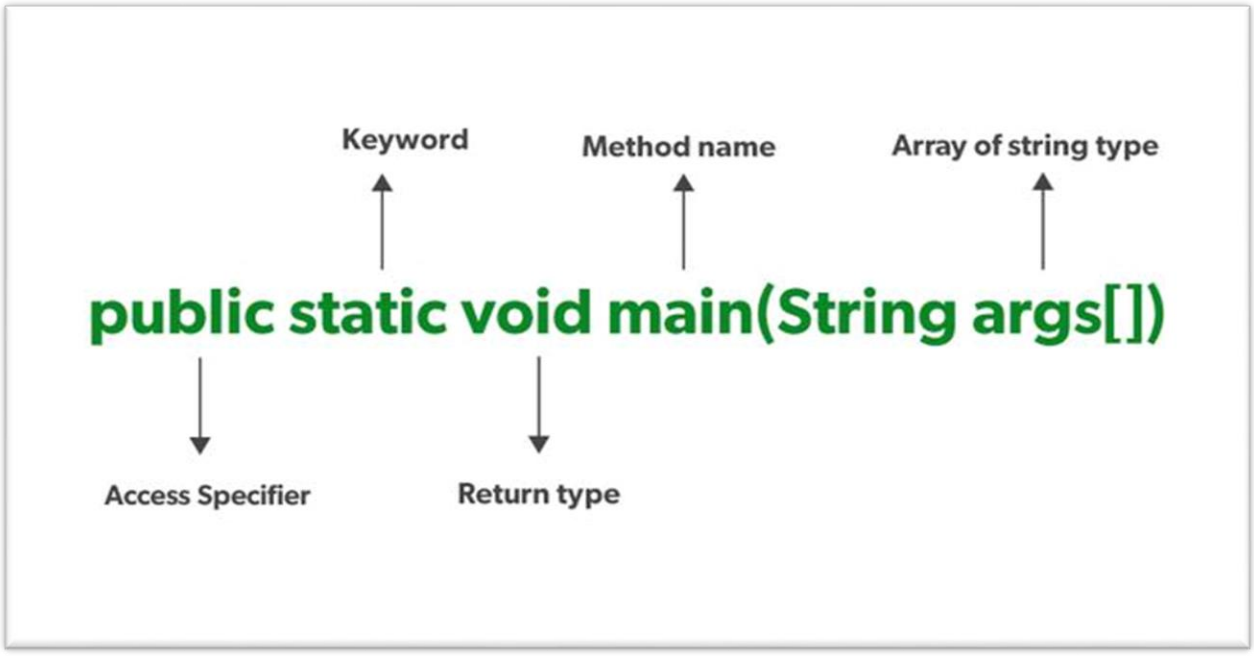
Java Runtime Environment (JRE) is an implementation of the JVM.

Java Development Kit (JDK) contains JRE along with various development tools like Java libraries, Java source compilers, Java debuggers, bundling and deployment tools.

Just in Time (JIT) is runs after the program has started executing, on the fly. It has access to runtime information and makes optimizations of the code for better performance.

API are important software components bundled with the JDK. APIs in Java include classes, interfaces, and user Interfaces. They enable developers to integrate various applications and websites and offer real-time information.

Q-4) Java main() Method – public static void main(String[] args) .



Example:

```
class Main {
   public static void main(String[] args)
   {
      System.out.println("I am a Geek");
   }
}
```

Output:

I am a Geek

Q-5) Explain Different Method to read Input in Java.

First Import Package: import java.util.Scanner;

Syntax of Java Scanner Class

```
Scanner a = new Scanner(System.in);
a.nextInt();
```

Example:

```
import java.util.Scanner;
class Main{
public static void main(String[]args) {
   Scanner a = new Scanner(System.in);
   a.nextInt();
   System.out.println(a);
}
      }
```

Methods of Java Scanner Class

Methods	Description
nextInt()	It takes int type input value from the user.
nextFloat()	It takes a float type input value from the user.
nextBoolean()	It takes a boolean type input value from the user.
nextLine()	t takes a line as an input value from the user.
next()	It takes a word as an input value from the user.
nextByte()	It takes a byte type of input value from the user.
nextDouble()	It takes a double type input value from the user.
nextShort()	It takes a short type input value from the user.
nextLong()	It takes a long type of input value from the user.

Q-6) Explain Data Types in detail with example.

Data Types in Java

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. Primitive data types: The primitive data types include Boolean, char, byte, short, int, long, float and double.
2. Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

There are 8 types of primitive data types:

- Boolean data type
- Byte data type
- Char data type
- Short data type
- Int data type

- Long data type
- Float data type
- Double data type

Example :

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

    boolean flag = true;
    System.out.println(flag);

byte range;
    range = 124;
    System.out.println(range);

short temperature;
    temperature = -200;
    System.out.println(temperature);

int a = -4250000;
    System.out.println(a);

long b = -42332200000L;
    System.out.println(b);

double number = -42.3;
    System.out.println(number);

float num = -42.3f;
    System.out.println(num);

char letter = 'Sahil';
    System.out.println(letter);

  }
}
```

Q-7) Explain Operators In Java With Example.

Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Assignment Operator.

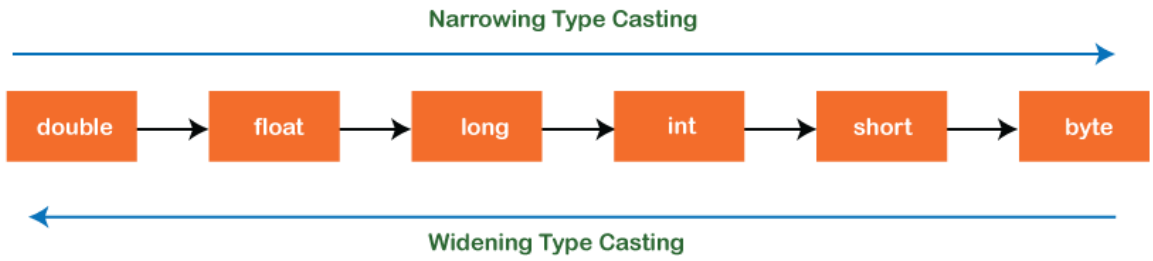
Operator Type	Category	Precedence
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Example:

```
public class Main{
   public static void main(String args[]){
      int x=10;
         System.out.println(x++);
         System.out.println(++x);
         System.out.println(x--);
         System.out.println(--x);
   }
}
```

Q-8) what is type casting? Explain widening and narrowing type casting.

Type casting is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.



Type Casting in Java

Widening Type Casting

Converting a lower data type into a higher one is called widening type casting. It is also known as implicit conversion or casting down. It is done automatically. It is safe because there is no chance to lose data.

Narrowing Type Casting

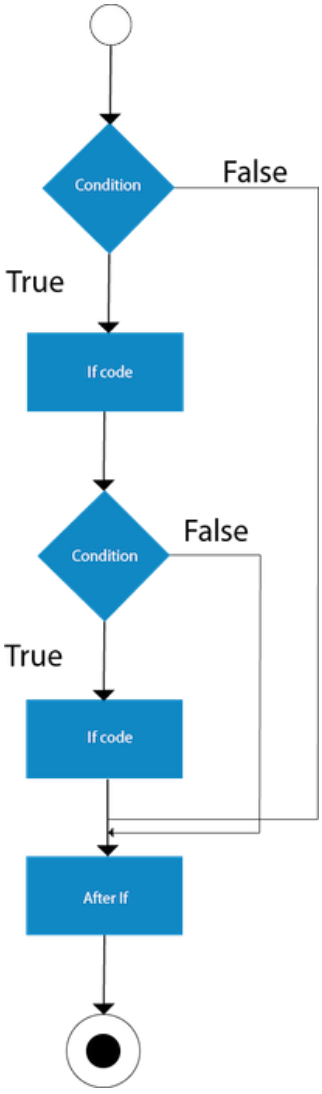
Converting a higher data type into a lower one is called narrowing type casting. It is also known as explicit conversion or casting up. It is done manually by the programmer.

Q-9) Explain nested If with Example.

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
        if(condition){
   }
}
```



Example:

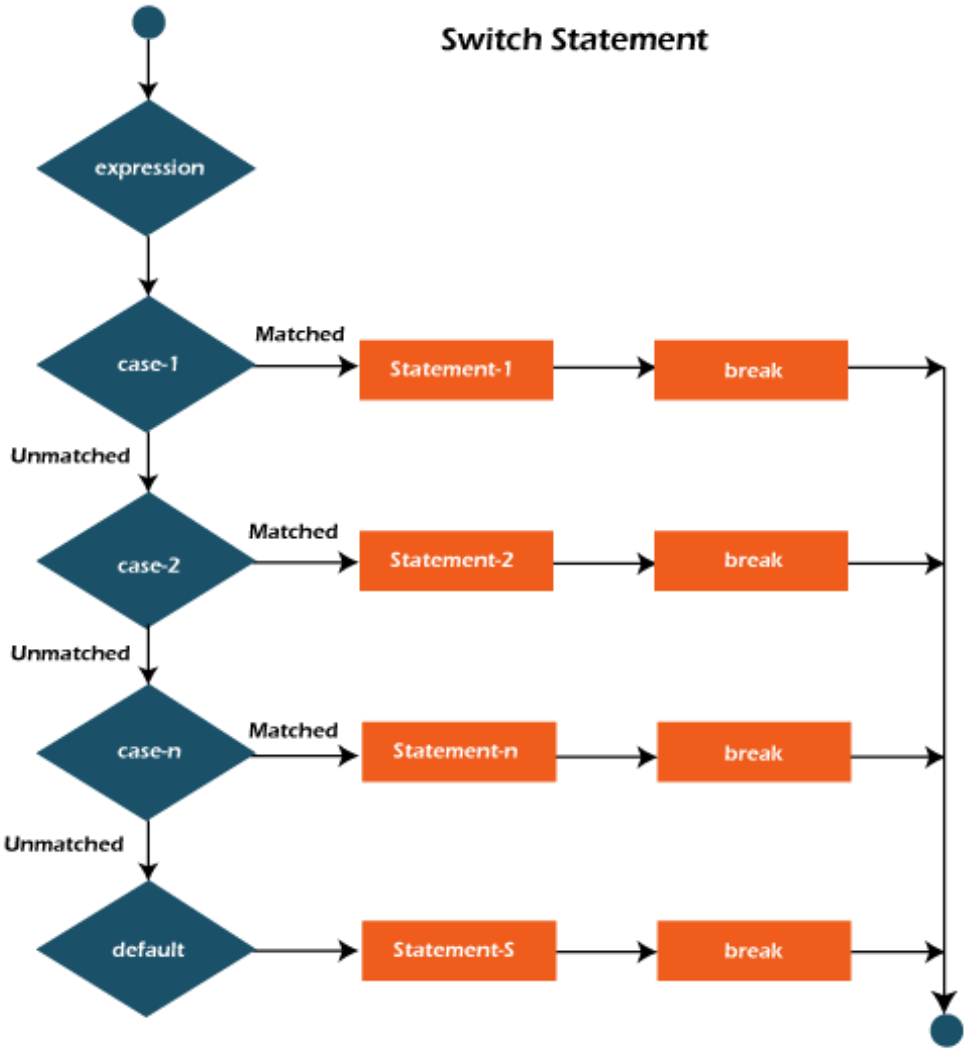
```
public class JavaNestedIfExample {
public static void main(String[] args) {
   int age=20;
   int weight=80;
   if(age>=18){
      if(weight>50){
         System.out.println("You are eligible to donate blood");
      }
   }
}
}
```

Q-10) Explain Switch Statement In Java.

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

Syntax:

```
switch(expression){
case value1:
 break;
case value2:
 break;
default:
}
```



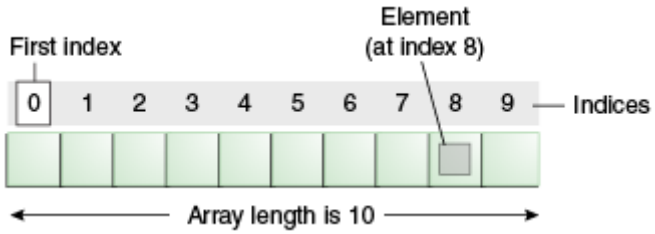
Example:

```
public class SwitchExample {
public static void main(String[] args) {
    int number=20;
    switch(number){
    case 10: System.out.println("10");
    break;
    case 20: System.out.println("20");
    break;
    case 30: System.out.println("30");
    break;
    default:System.out.println("Not in 10, 20 or 30");
    }
  }
}
```

Q-11) what is Array? Explain Array Types With Example.

Normally, an array is a collection of similar type of elements which has contiguous memory location.

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. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.



Advantages

- Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.
- Random access: We can get any data located at an index position.

Disadvantages

- Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Example:

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

   int a[] = new int[5];
   int b[][] = new int[2][2];
```

```
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;

    b[0][0] = 5;
    b[0][1] = 3;
    b[1][0] = 8;
    b[1][1] = 6;

    System.out.println(b[1][1]);

    System.out.println(a[2]);

}
     }
```

Q-12) How To pass Array in function and How return Array from Method?

In this section, we are going to learn how to return an array in Java.

Remember:

- A method can return a reference to an array.
- The return type of a method must be declared as an array of the correct data type.

Example:

```
import java.util.Arrays;
   public class Main{
      public static void main(String args[]){
         int[] a=numbers();
         for (int i = 0; i < a.length; i++)
         System.out.print( a[i]+ " ");
}
  public static int[] numbers()
{
int[] arr={5,6,7,8,9};
return arr;
}
}
```

Q-13) Explain Concept of Class & Object.

Object:

An object is an identifiable entity with some characteristics, state and behavior. Understanding the concept of objects is much easier when we consider real-life examples around us because an object is simply a real-world entity.

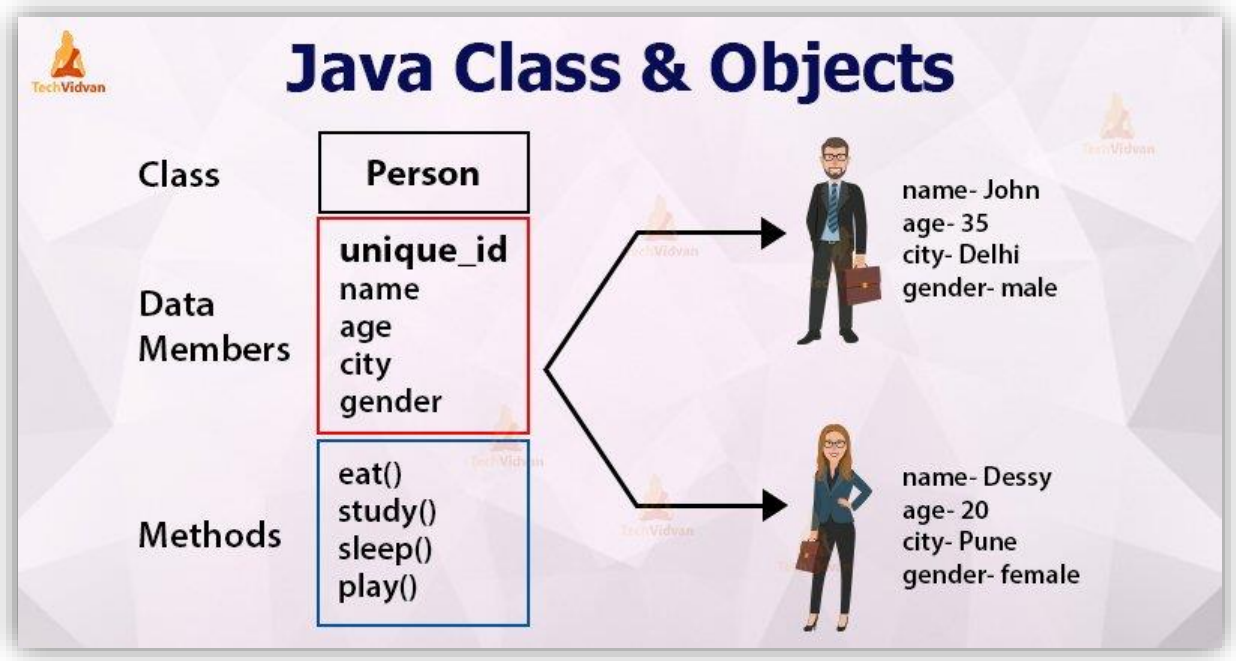
You will find yourself surrounded by the number of objects which have certain characteristics and behaviours.

For example, we can say 'Orange' is an object. Its characteristics are: spherical in shape and colour is orange. Its behaviour is: juicy and tastes sweet-sour.

Class:

A class is a group of objects that share common properties and behaviour.

For example, we can consider a car as a class that has characteristics like steering wheels, seats, brakes, etc. And its behaviour is mobility. But we can say Honda City having a reg.number 4654 is an 'object' that belongs to the class 'car'.
It was a brief description of objects and classes. Now we will understand the Java class in detail.



Q-14) Explain Access Modifiers or Visibility Modifiers.

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. If you do not make the child class, it cannot be accessed from outside the package.
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.

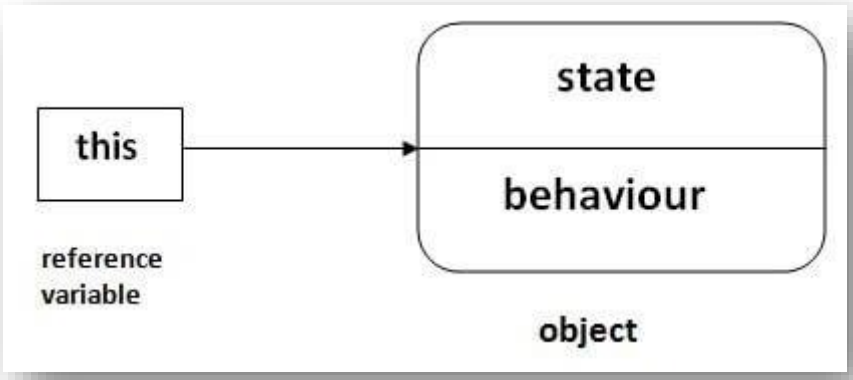
Understanding Java Access Modifiers

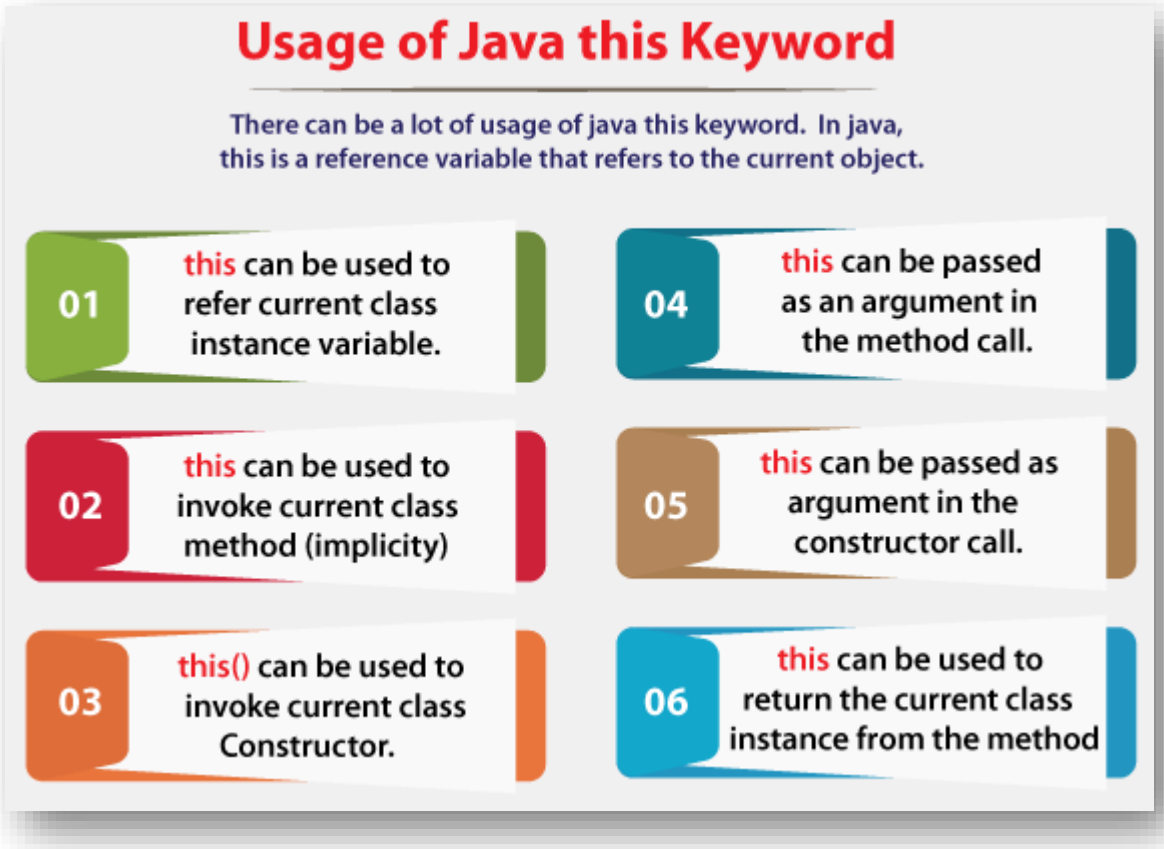
Let's understand the access modifiers in Java by a simple table.

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

Q-15) Explain Use Of this keyword in Java.

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.





Q-16) Abstraction Vs Encapsulation

Abstraction	Encapsulation
Abstraction is the process or method of gaining the information.	While encapsulation is the process or method to contain the information.
In abstraction, problems are solved at the design or interface level.	While in encapsulation, problems are solved at the implementation level.
Abstraction is the method of hiding the unwanted information.	Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.
We can implement abstraction using abstract class and interfaces.	Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.
In abstraction, implementation complexities are hidden using abstract classes and interfaces.	While in encapsulation, the data is hidden using methods of getters and setters.
The objects that help to perform abstraction are encapsulated.	Whereas the objects that result in encapsulation need not be abstracted.
Abstraction provides access to specific part of data.	Encapsulation hides data and the user cannot access same directly (data hiding.
Abstraction focus is on “what” should be done.	Encapsulation focus is on “How” it should be done.

Q-17) Primitive Data Types (Autoboxing) & Wrapper Data Types (Unboxing).

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

Wrapper Class vs Primitive Type in Java	
Wrapper class provides a mechanism to convert primitive type into object and object into primitive type.	A primitive type is a predefined data type provided by Java.
Associated Class	
A Wrapper class is used to create an object; therefore, it has a corresponding class.	A Primitive type is not an object so it does not belong to a class.
Null Values	
The wrapper class objects allow null values.	A primitive data type does not allow null values.
Memory Required	
Required memory is higher than the primitive types. The Clustered Index does not require an additional space.	Required memory is lower comparing to wrapper classes.
Collections	
A Wrapper class can be used with a collection such as ArrayList, etc.	A primitive type is not used with collections.

Q-18) Difference between StringBuffer and StringBuilder

No.	StringBuffer	StringBuilder
1)	StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5

Q-19) what is Inheritance list out its type and explain with example or explain the concept of superclass and subclass.

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

- Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

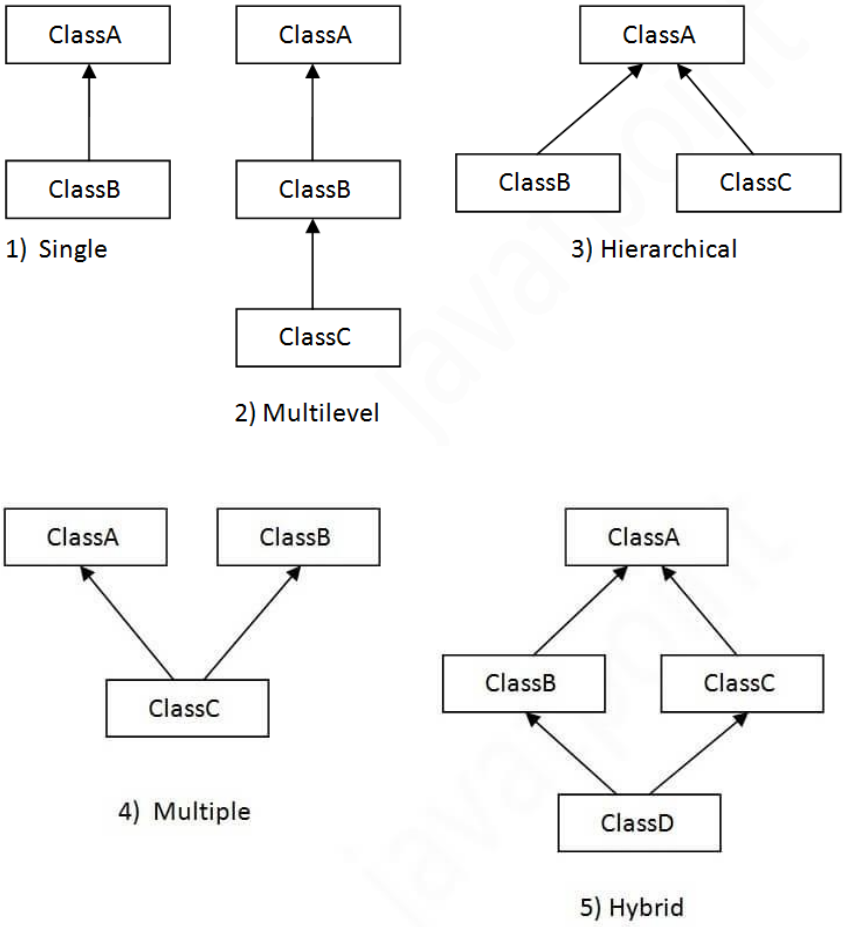
The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name

{

}
```

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

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

Types of inheritance in java



Example:

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Q-20) what is Constructor? Types of Constructor with Example.

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the Class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Rules for creating Java constructor

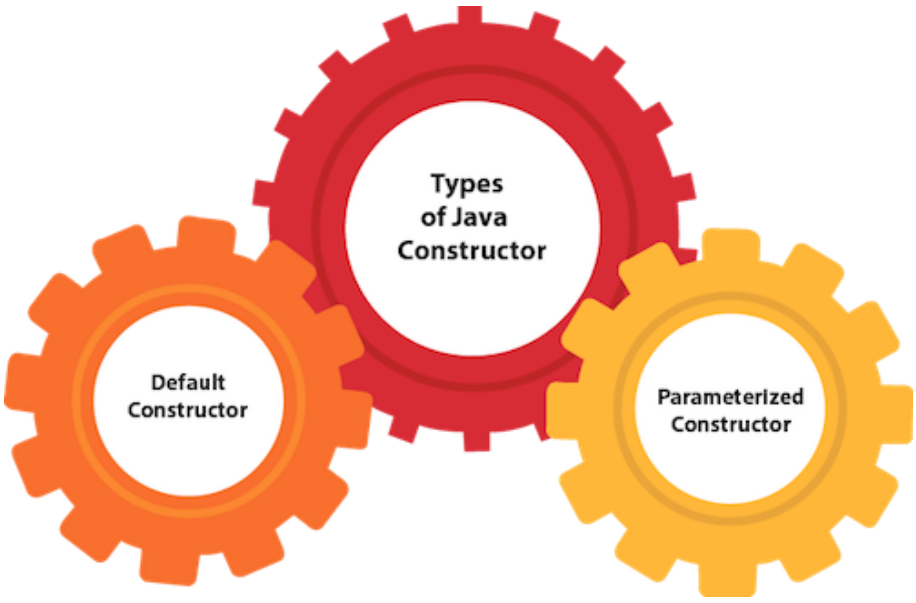
There are two rules defined for the constructor.

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

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Example:

```
class Bike1{
Bike1(){
System.out.println("Bike is created");}
public static void main(String args[]){
Bike1 b=new Bike1();
}
}
```

Q-21) what is Constructor Overloading?

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example:

```
class Student5{
   int id;
   String name;
   int age;
   r
   Student5(int i,String n){
   id = i;
   name = n;
   }

   Student5(int i,String n,int a){
   id = i;
   name = n;
   age=a;
   }
   void display(){
                System.out.println(id+" "+name+" "+age);
                }

   public static void main(String args[]){
   Student5 s1 = new Student5(111,"Karan");
   Student5 s2 = new Student5(222,"Aryan",25);
   s1.display();
   s2.display();
  }
}
```

Q-22) what is Method Overloading Explain with example.

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

If we have to perform only one operation, having same name of the methods increases the readability of the program

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

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

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods

so that we don't need to create instance for calling methods.

Example :

```
class Adder{
   static int add(int a,int b){
return a+b;
}
   static int add(int a,int b,int c){
return a+b+c;
}
 }
public static void main(String[] args){
       System.out.println(Adder.add(11,11));
        System.out.println(Adder.add(11,11,11));
    }
}
```

Output :

22

33

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type

The first add method receives two integer arguments and second add method receives two double arguments.

Example :

```
class Adder{
static int add(int a, int b){
return a+b;
}
static double add(double a, double b){
return a+b;
}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
    }
}
```

Output :

22

24.9

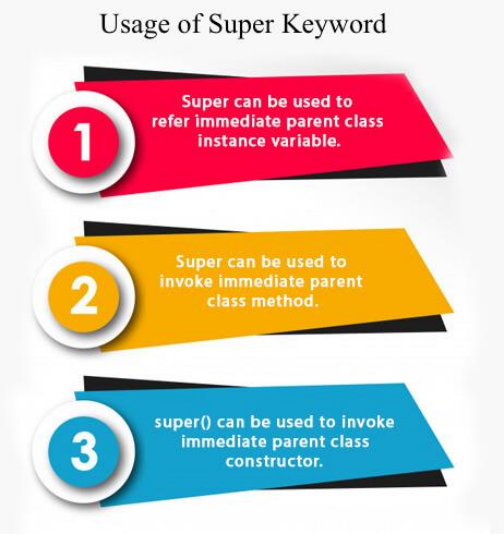
Q-23) Method Overloading (Compile time polymorphism) Vs Method Overriding (Run Time polymorphism).

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-24) Use of Super Keyword in Java with Example.

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.



Example:

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);
System.out.println(super.color);
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}
}
```

Q-25) Explain ArrayList Class in Java and Its Method.

ArrayList is an ordered sequence of elements

It provides random access to its elements. Random access means that we can grab any element at constant time.

It is dynamic and resizable.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

Syntax:

```
import java.util.ArrayList;

ArrayList<String> cars = new ArrayList<String>();
```

Example :

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();
     list.add("Mango");
     list.add("Apple");
     list.add("Banana");
     list.add("Grapes");
```

```
    System.out.println(list);
 }
}
```

Methods	Descriptions
size()	Returns the length of the arraylist.
sort()	Sort the arraylist elements.
clone()	Creates a new arraylist with the same element, size, andcapacity.
contains()	Searches the arraylist for the specified element and returns aboolean result.
ensureCapacity()	Specifies the total element the arraylist can contain.
indexOf()	Searches a specified element in an arraylist and returns the indexof the element.
isEmpty()	Checks if the arraylist is empty.

Q-26) Explain Static keyword.

The static keyword in Java is used for memory management mainly. We can apply static keyword with variables methods, blocks and nested classes.

The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

Q-27) what is exception? List out types of inbuilt exception. Explain with example.

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

In this tutorial, we will learn about Java exceptions, its types, and the difference between checked and unchecked exceptions.

What is Exception Handling?

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

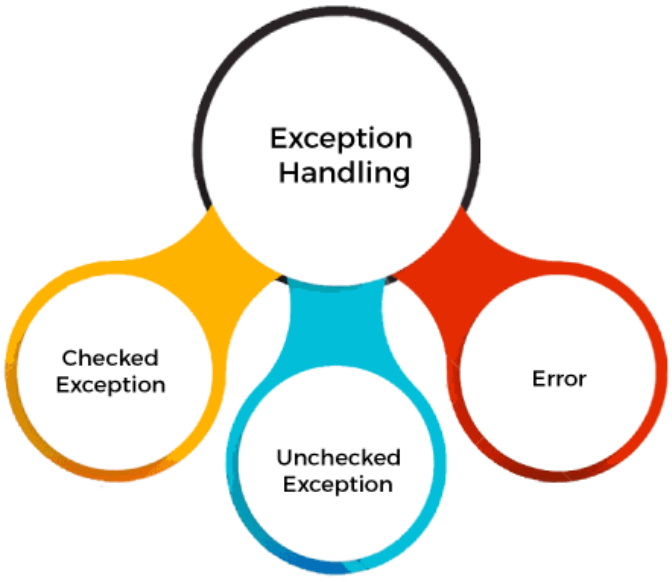
Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions. Let's consider a scenario:

Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. An error is considered as the unchecked exception. However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error



Difference between Checked and Unchecked Exceptions

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

Example:

```
public class Main {
   public static void main(String args[]) {
      try {
         int i = 1 / 100;
      }
         catch(ArithmeticException e){
            System.out.println(e);
         }
         System.out.println("Error");
      }

   }
```

Q-28) Explain Finally Clause. Final keyword & Final Method with Example.

- A finally block contains all the crucial statements that must be executed whether exception occurs or not.
- The statements present in this block will always execute regardless of whether exception occurs in try block or not such as closing a connection, stream etc.
- A finally block must be associated with a try block, you cannot use finally without a try block. You should place those statements in this block that must be executed always.
- Finally block is optional, as we have seen in previous tutorials that a try-catch block is sufficient for exception handling, however if you place a finally block then it will always run after the execution of try block.
- In normal case when there is no exception in try block then the finally block is executed after try block. However if an exception occurs then the catch block is executed before finally block.
- An exception in the finally block, behaves exactly like any other exception.
- The statements present in the finally block execute even if the try block contains control transfer statements like return, break or continue.

Example:

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

System.out.println(JavaFinally.myMethod());
}
public static int myMethod()
{
try {
return 112;
}
finally {
System.out.println("This is Finally block");
System.out.println("Finally block ran even after return statement");
        }
    }
}
```

Q-29) what is Interface? What is need of interface? Explain with example.

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

The interface in Java is *a mechanism to achieve abstraction*

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

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the IS-A relationship.

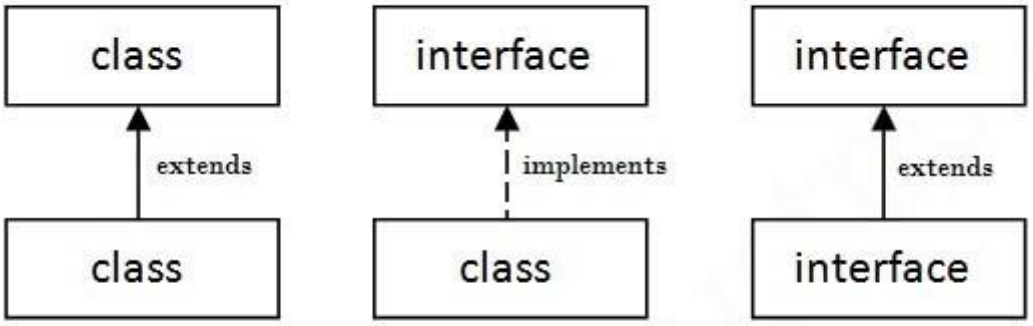
Why use Java interface?

Example:

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){
System.out.println("Hello");
}
public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

The relationship between classes and interfaces:

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



Q-30) Abstract Class Vs Abstract Method.

Abstract classes	Abstract methods
Abstract classes can't be instantiated.	Abstract method bodies must be empty.
Other classes extend abstract classes.	Sub-classes must implement the abstract class's abstract methods.
Can have both abstract and concrete methods.	Has no definition in the class.
Similar to interfaces, but can 1.Implement methods 2.Fields can have various 3.access modifiers 4.Subclasses can only extend one abstract class	Has to be implemented in a derived class.

Q-31) Class vs Interface.

Class	Interface
A class can be instantiated	An interface can never be instantiated
The class keyword is used to declare it	The interface keyword is used
The members of a class can be declared as private, public or protected	The members of an interface are always declared as public
Contains the concrete methods i.e methods with body	Contains abstract method i.e methods without the body
The extends keyword is used to inherit a class	The implements keyword is used to use an interface
Can contain final and static methods	Cannot contain final or static methods
A Java class can have constructors	An interface cannot have constructors
A class can extend only one class but can implement any number of interfaces	An interface can extend any number of interfaces but cannot implement any interface

Q-32) Abstract Class Vs Final Class

No.	ABSTRACT CLASS	FINAL CLASS
1.	Uses the “abstract” key word.	Uses the “final” key word.
2.	This helps to achieve abstraction.	This helps to restrict other classes from accessing its properties and methods.
3.	For later use, all the abstract methods should be overridden	Overriding concept does not arise as final class cannot be inherited
4.	A few methods can be implemented and a few cannot	All methods should have implementation
5.	Cannot create immutable objects (infact, no objects can be created)	Immutable objects can be created (eg. String class)
6.	Abstract class methods functionality can be altered in subclass	Final class methods should be used as it is by other classes
7.	Can be inherited	Cannot be inherited
8.	Cannot be instantiated	Can be instantiated

Q-33) Comparable Vs Comparator (Clonable) Interface.

Comparable	Comparator
Comparable provides compareTo() method to sort elements in Java.	Comparator provides compare() method to sort elements in Java.
Comparable interface is present in java.lang package.	Comparator interface is present in java.util package.
The logic of sorting must be in the same class whose object you are going to sort.	The logic of sorting should be in separate class to write different sorting based on different attributes of objects.
The class whose objects you want to sort must implement comparable interface.	Class, whose objects you want to sort, do not need to implement a comparator interface.
It provides single sorting sequences.	It provides multiple sorting sequences.
This method can sort the data according to the natural sorting order.	This method sorts the data according to the customized sorting order.
It affects the original class. i.e., actual class is altered.	It doesn't affect the original class, i.e., actual class is not altered.
Implemented frequently in the API by: Calendar, Wrapper classes, Date, and String.	It is implemented to sort instances of third-party classes.
All wrapper classes and String class implement comparable interface.	The only implemented classes of Comparator are Collator and RuleBasedColator.

Q-34) throw Vs throws with Example.

No.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.	Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
2.	Used	Type of exception Using throw keyword, we can only propagate unchecked exception i.e., the checked exception cannot be propagated using throw only.	Using throws keyword, we can declare both checked and unchecked exceptions. However, the throws keyword can be used to propagate checked exceptions only.
3.	Syntax	The throw keyword is followed by an instance of Exception to be thrown.	The throws keyword is followed by class names of Exceptions to be thrown.
4.	Declaration	throw is used within the method.	throws is used with the method signature.
5.	Internal implementation	We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.	We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

Example:

```
public class Main{
   static void method() throws ArithmeticException
   {
      System.out.println("Inside the method()");
      throw new ArithmeticException("throwing ArithmeticException");
   }
   public static void main(String args[]){
      try{
         method();
      }
      catch(ArithmeticException e){
         System.out.println("caught in main() method");
      }
   }
}
```

Q-35) Explain textarea , chechbox , radiobutton , textfield , label , passwordfield in JavaFX.

Creating a Button

A JavaFX Button control enables a JavaFX application to have some action executed when the application user clicks the button. The JavaFX Button control is represented by the class javafx.scene.control.Button . A JavaFX Button can have a text and an icon on it which indicate to the user what clicking the button will do.

You create a button control by creating an instance of the Button class. Here is a JavaFX Button instantiation example:

```
Button button = new Button("My Label");
```

The text to be displayed on the button is passed as parameters to the Button constructor.

Creating a RadioButton

The JavaFX RadioButton is represented by the class javafx.scene.control.RadioButton. The RadioButton class is a subclass of the ToggleButton class.

You create a JavaFX RadioButton using its constructor. Here is a JavaFX RadioButton instantiation example:

```
RadioButton radioButton1 = new RadioButton("Left");
```

The String passed as parameter to the RadioButton constructor is displayed next to the RadioButton.

Creating a CheckBox

You create a JavaFX CheckBox control via the CheckBox constructor. Here is a JavaFX CheckBox instantiation example:

```
CheckBox checkBox1 = new CheckBox("Green");
```

The String passed to the CheckBox constructor is displayed next to the CheckBox control.

Creating a ListView

The JavaFX ListView control enables users to choose one or more options from a predefined list of choices. The JavaFX ListView control is represented by the class javafx.scene.control.ListView . This JavaFX ListView tutorial will explain how to use the ListView class.

You create a ListView simply by creating a new instance of the ListView class. Here is a JavaFX ListView instantiation example:

```
ListView listView = new ListView();
```

Adding Items to a ListView

You can add items (options) to a ListView by obtaining its item collection and add items to it. Here is an example that adds items to a JavaFX ListView :

```
listView.getItems().add("Item 1");
listView.getItems().add("Item 2");
listView.getItems().add("Item 3");
```

Creating a ToggleButton

A JavaFX ToggleButton is a button that can be selected or not selected. Like a button that stays in when you press it, and when you press it the next time it comes out again. Toggled - not toggled. The JavaFX ToggleButton is represented by the class javafx.scene.control.ToggleButton .

You create a JavaFX ToggleButton by creating an instance of the ToggleButton class. Here is an example of creating a JavaFX ToggleButton instance:

```
ToggleButton toggleButton1 = new ToggleButton("Left");
```

ToggleButton Text

You can set or change the text of a JavaFX ToggleButton via its setText() method. Here is an example of changing the text of a JavaFX ToggleButton via setText():

```
ToggleButton toggleButton = new ToggleButton("Toggle This!");

toggleButton.setText("New Text");
```

Creating a TextField

A JavaFX TextField control enables users of a JavaFX application to enter text which can then be read by the application. The JavaFX TextField control is represented by the class javafx.scene.control.TextField .

You create a TextField control by creating an instance of the TextField class. Here is a JavaFX TextField instantiation example:

```
TextField textField = new TextField();
```

Setting the Text of a TextField

You can set the text of a TextField using its setText() method. This is often useful when you need to set the initial value for at text field that is part of a form. For instance, editing an existing object or record. Here is a simple example of setting the text of a JavaFX TextField:

```
textField.setText("Initial value");
```

Q-36) Different Layout Panes in JavaFX.

No	Shape & Description
1	HBox The HBox layout arranges all the nodes in our application in a single horizontal row. The class named HBox of the package javafx.scene.layout represents the text horizontal box layout.
2	VBox The VBox layout arranges all the nodes in our application in a single vertical column. The class named VBox of the package javafx.scene.layout represents the text Vertical box layout.
3	BorderPane The Border Pane layout arranges the nodes in our application in top, left, right, bottom and center positions. The class named BorderPane of the package javafx.scene.layout represents the border pane layout.

4	StackPane The stack pane layout arranges the nodes in our application on top of another just like in a stack. The node added first is placed at the bottom of the stack and the next node is placed on top of it. The class named StackPane of the package javafx.scene.layout represents the stack pane layout.
5	TextFlow The Text Flow layout arranges multiple text nodes in a single flow. The class named TextFlow of the package javafx.scene.layout represents the text flow layout.
6	AnchorPane The Anchor pane layout anchors the nodes in our application at a particular distance from the pane. The class named AnchorPane of the package javafx.scene.layout represents the Anchor Pane layout.
7	TilePane The Tile Pane layout adds all the nodes of our application in the form of uniformly sized tiles. The class named TilePane of the package javafx.scene.layout represents the TilePane layout.
8	GridPane The Grid Pane layout arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX. The class named GridPane of the package javafx.scene.layout represents the GridPane layout.
9	FlowPane The flow pane layout wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width. The class named FlowPane of the package javafx.scene.layout represents the Flow Pane layout.

Q-37) Explain colour class and its Method in JavaFX.

In JavaFX, we can fill the shapes with the colors. We have the flexibility to create our own color using the several methods and pass that as a Paint object into the setFill() method. Lets discuss the several methods of creating color in JavaFX.

RGB Color

RGB color system is the most popular method to create a color in graphics. It consists of three components named as RED → R, GREEN → G and BLUE → B. Each component uses 8 Bits that means every component can have the integer value from 0 to 22^8 - 1=255.

The computer screen can be seen as the collection of pixels. The set (R,G,B) actually represents the emission of their respective LEDs on the screen.

If the value of RED is set to 0 then it means that the Red LED is turned off while the value 255 indicates that the full emission of LED is being there. The combination of (0,0,0) represents the black color while (255,255,255) represents the white color. The middle values in that range can represent different colors.

Using the superimposition of RGB, we can represent 255*255*255 different colors. In JavaFX, the class javafx.scene.paint.Color class represents colors.

There is a static method named as rgb() of Color class. It accepts three integer arguments as Red, Green, Blue and one optional double argument called alpha. The value of alpha is proportional to the opacity of the color. The alpha value 0 means that the color is completely transparent while the value 1 means that the color is completely opaque.

Q-38) Image & ImageView Class in JavaFX.

JavaFX allows you to work with all popular image formats. Let's use class javafx.scene.image.Image to load images from hard drive or a network image sources. In order to display images on JavaFX, you use ImageView class.

javafx.scene.image.Image

```
Image image = new Image(input);
```

The Constructors of class Image help you to load image data:

```
Image(InputStream inputStream)

Image(InputStream is, double requestedWidth, double requestedHeight,
        boolean preserveRatio, boolean smooth)
```

Image(String url)

Image(String url, boolean backgroundLoading)

Image(String url, double requestedWidth, double requestedHeight,
boolean preserveRatio, boolean smooth)

Image(String url, double requestedWidth, double requestedHeight,
boolean preserveRatio, boolean smooth, boolean backgroundLoading)

ImageView is a component which helps you to display images on JavaFX. You can also apply effects to display images such as rotate, zoom in and out.

import javafx.scene.image.ImageView;

ImageView imageView = new ImageView(image);

Q-39) Explain Inner Class with Example.

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

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

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

Syntax of Inner class:

```
class Java_Outer_class{

class Java_Inner_class{

}

}
```

Advantage of Java inner classes

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

Example:

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}
```

Q-40) Write down Short note for I/O streamclass with Example.

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform file handling in Java by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

OutputStream vs InputStream

The explanation of OutputStream and InputStream classes are given below:

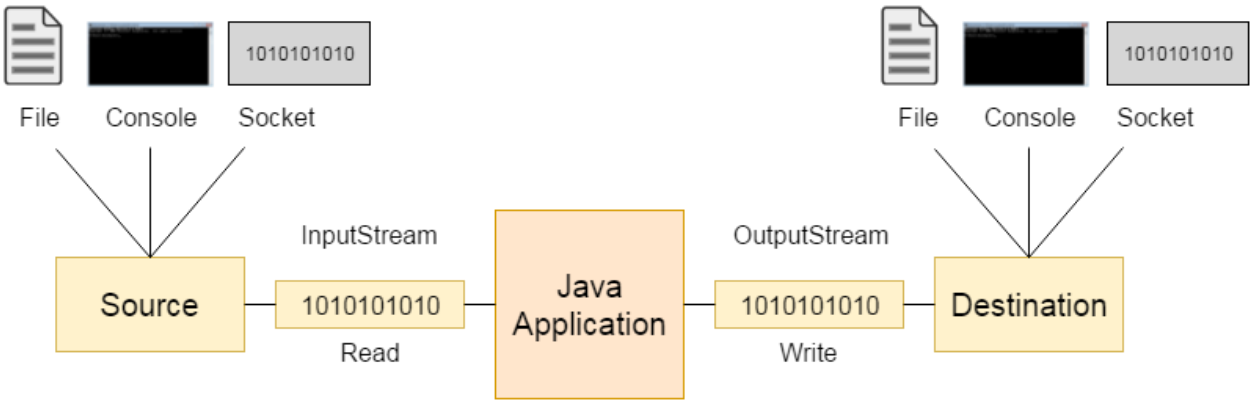
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream by the figure given below.



Example:

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;
public class BufferedInputStreamExample {
  public static void main(String args[]) throws IOException {

    BufferedInputStream inputStream = new BufferedInputStream(System.in);
    byte bytes[] = new byte[1024];
    System.out.println("Enter your data ");

    inputStream.read(bytes);
```

```
    FileOutputStream out= new FileOutputStream("D:/myFile.txt");
    BufferedOutputStream outputStream = new BufferedOutputStream(out);

    outputStream.write(bytes);
    outputStream.flush();
    System.out.println("Data successfully written in the specified file");
  }
}
```

Q-41) what is generic class and interface explain with example.

Generics make a class, interface and, method, consider all (reference) types that are given dynamically as parameters. This ensures type safety. Generic class parameters are specified in angle brackets “<>” after the class name as of the instance variable.

Generic constructors are the same as generic methods. For generic constructors after the public keyword and before the class name the type parameter must be placed. Constructors can be invoked with any type of a parameter after defining a generic constructor. A constructor is a block of code that initializes the newly created object. It is an instance method with no return type. The name of the constructor is same as the class name. Constructors can be Generic, despite its class is not Generic.

Example:

```
import java.io.*;
class GenericConstructor {

        private double v;

        <T extends Number> GenericConstructor(T t)
        {
                v = t.doubleValue();
        }
        void show()
        {
                System.out.println("v: " + v);
        }
}
class GFG {
        public static void main(String[] args)
        {
                System.out.println("Number to Double Conversion:");
```

```
            GenericConstructor obj1 = new GenericConstructor(10);
            GenericConstructor obj2 = new GenericConstructor(136.8F);
            obj1.show();
            obj2.show();
        }
}
```

Q-42) AWT Vs Swing

Context	AWT	Swing
API Package	The AWT Component classes are provided by the java.awt package.	The Swing component classes are provided by the javax.swing package.
Operating System	The Components used in AWT are mainly dependent on the operating system.	The Components used in Swing are not dependent on the operating system. It is completely scripted in Java.
Weightiness	The AWT is heavyweight since it uses the resources of the operating system.	The Swing is mostly lightweight since it doesn't need any Operating system object for processing. The Swing Components are built on the top of AWT.
Appearance	The Appearance of AWT Components is mainly not configurable. It generally depends on the operating system's look and feels.	The Swing Components are configurable and mainly support pluggable look and feel.
Number of Components	The Java AWT provides a smaller number of components in comparison to Swing.	Java Swing provides a greater number of components than AWT, such as list, scroll panes, tables, color choosers, etc.
Full-Form	Java AWT stands for Abstract Window Toolkit.	Java Swing is mainly referred to as Java Foundation Classes (JFC).
Peers	Java AWT has 21 peers. There is one peer for each control and one peer for the dialogue. Peers are provided by the operating system in the form of widgets themselves.	Java Swing has only one peer in the form of OS's window object, which provides the drawing surface used to draw the Swing's widgets (label, button, entry fields, etc.) developed directly by Java Swing Package.

Functionality and Implementation	Java AWT many features that are completely developed by the developer. It serves as a thin layer of development on the top of the OS.	Swing components provide the higher-level inbuilt functions for the developer that facilitates the coder to write less code.
Memory	Java AWT needs a higher amount of memory for the execution.	Java Swing needs less memory space as compared to Java AWT.
Speed	Java AWT is slower than swing in terms of performance.	Java Swing is faster than the AWT

Q-43) what is thread. Describe the lifecycle of thread.

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread that is provided by the JVM or Java Virtual Machine at the starting of the program's execution.

In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

Explanation of Different Thread States

New: Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is runnable, and the other is running.

- Runnable: A thread that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.

- Running: When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Blocked or Waiting: Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state. Of India | List of Prime Minister of India (1947-2020)

For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the same time, the other thread (let's say its name is B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state. A thread in the blocked state is unable to perform any execution and thus never consume any cycle of the Central Processing Unit (CPU). Hence, we can say that thread A remains idle until the thread scheduler reactivates thread A, which is in the waiting or blocked state.

When the main thread invokes the join() method then, it is said that the main thread is in the waiting state. The main thread then waits for the child threads to complete their tasks. When the child threads complete their job, a notification is sent to the main thread, which again moves the thread from waiting to the active state.

If there are a lot of threads in the waiting or blocked state, then it is the duty of the thread scheduler to determine which thread to choose and which one to reject, and the chosen thread is then given the opportunity to run.

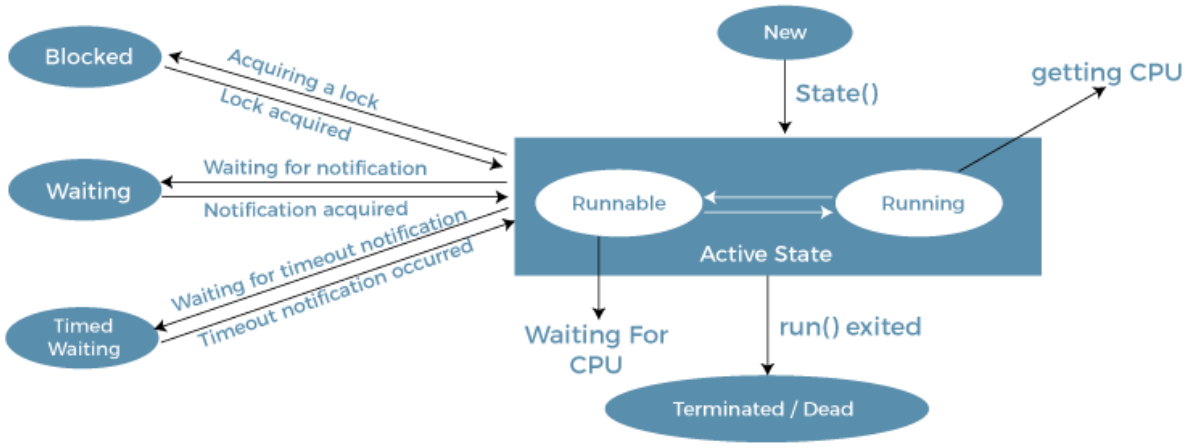
Timed Waiting: Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

Terminated: A thread reaches the termination state because of the following reasons:

- When a thread has finished its job, then it exists or terminates normally.
- Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

The following diagram shows the different states involved in the life cycle of a thread.



Life Cycle of a Thread

Q-44) List Vs Set.

List	Set
1. The List is an indexed sequence.	1. The Set is a non-indexed sequence.
2. List allows duplicate elements	2. Set doesn't allow duplicate elements.
3. Elements by their position can be accessed.	3. Position access to elements is not allowed.
4. Multiple null elements can be stored.	4. Null element can store only once.
5. List implementations are ArrayList, LinkedList, Vector, Stack	5. Set implementations are HashSet, LinkedHashSet.

Q-45) Explain static Variable and static method with example.

The static variable is a class level variable and it is common to all the class objects i.e. a single copy of the static variable is shared among all the class objects.

A static method manipulates the static variables in a class. It belongs to the class instead of the class objects and can be invoked without using a class object.

The static initialization blocks can only initialize the static instance variables. These blocks are only executed once when the class is loaded.

Example:

```
public class Demo {
  static int x = 10;
  static int y;
  static void func(int z) {
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("z = " + z);
  }
  static {
    System.out.println("Running static initialization block.");
    y = x + 5;
  }
  public static void main(String args[]) {
    func(8);
  }
}
```

Q-46) List out the limitation & Restriction of Generic Programing.

Disadvantages:

Generics don't work with primitive types.
We can't create Generic Arrays
We cannot create an instance of a type parameter.
We cannot create, catch, or throw generic types.
Due to type erasure, you cannot use instance of with generic types.
We cannot declare static fields of a type parameter.

Advantages:

There are basically three main advantages of generics in Java.

Generics provide type-safety. So, we can hold a single type of object in generics.
It provides a compile-time checker so it doesn't show an error at runtime.
There is no necessity to typecast an object.

Q-47) Thread Synchronization.

Synchronization means coordination between multiple processes/threads.

Types of synchronization:

There are two types of synchronization that are as follows:

1. Process synchronization
2. Thread synchronization

Here we will be mainly focusing on thread synchronization.

Thread synchronization basically refers to the concept of one thread execute at a time and the rest of the threads are in waiting state. This process is known as thread synchronization. It prevents the thread interference and inconsistency problem.

Synchronization is build using locks or monitor. In Java, a monitor is an object that is used as a mutually exclusive lock. Only a single thread at a time has the right to own a monitor. When a thread gets a lock then all other threads will get suspended which are trying to acquire the locked monitor. So, other threads are said to be waiting for the monitor, until the first thread exits the monitor. In a simple way, when a thread request a resource then that resource gets locked so that no other thread can work or do any modification until the resource gets released.

Thread Synchronization are of two types:

1. Mutual Exclusive
2. Inter-Thread Communication

A. Mutual Exclusive

While sharing any resource, this will keep the thread interfering with one another i.e. mutual exclusive. We can achieve this via
Synchronized Method
Synchronized Block
Static Synchronization

Synchronized Method

We can declare a method as synchronized using the “synchronized” keyword. This will make the code written inside the method thread-safe so that no other thread will execute while the resource is shared.

Implementation:

We will be proposing prints the two threads simultaneously showing the asynchronous behaviour without thread synchronization.

Example:

```
public class PrintTest extends Thread {

	public void printThread(int n)
	{

		for (int i = 1; i <= 10; i++) {
			System.out.println("Thread " + n + " is working...");

			try {
				Thread.sleep(600);
			}
			catch (Exception ex) {
				System.out.println(ex.toString());
			}
		}
		System.out.println("-------------------------");

		try {
			Thread.sleep(1000);
		}

		catch (Exception ex) {
			System.out.println(ex.toString());
```

```
                }
        }
}
public class Thread1 extends Thread {
        PrintTest test;

        Thread1(PrintTest p) { test = p; }

        public void run()
        {

                test.printThread(1);
        }
}

public class Thread2 extends Thread {

        PrintTest test;

        Thread2(PrintTest p) { test = p; }

        public void run() { test.printThread(2); }
}

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

                PrintTest p = new PrintTest();
                Thread1 t1 = new Thread1(p);
                Thread2 t2 = new Thread2(p);
                t1.start();
                t2.start();
        }
}
```