1) What is Java?

Java is the high-level, object-oriented, robust, secure programming language, platform-independent, high performance, Multithreaded, and portable programming language. It was developed by James Gosling in June 1991. It can also be known as the platform as it provides its own JRE and API.

---

2) What are the differences between C++ and Java?

The differences between C++ and Java are given in the following table.

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of C programming language.	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed with a goal of being easy to use and accessible to a broader audience.

Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in Java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses compiler and interpreter both. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.

Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses a single inheritance tree always because all classes are the child of Object class in java. The object class is the root of the inheritance tree in java.
Hardware	C++ is nearer to hardware.	Java is not so interactive with hardware.

Object-oriented	C++ is an object-oriented language. However, in C language, single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

3) List the features of Java Programming language.

There are the following features in Java Programming Language.

- **Simple:** Java is easy to learn. The syntax of Java is based on C++ which makes easier to write the program in it.

- **Object-Oriented:** Java follows the object-oriented paradigm which allows us to maintain our code as the combination of different type of objects that incorporates both data and behavior.

- **Portable:** Java supports read-once-write-anywhere approach. We can execute the Java program on every machine. Java program (.java) is converted to bytecode (.class) which can be easily run on every machine.

- **Platform Independent:** Java is a platform independent programming language. It is different from other programming languages like C and C++ which needs a platform to be executed. Java comes with its platform on which its code is executed. Java doesn't depend upon the operating system to be executed.

- **Secured:** Java is secured because it doesn't use explicit pointers. Java also provides the concept of ByteCode and Exception handling which makes it more secured.

- **Robust:** Java is a strong programming language as it uses strong memory management. The concepts like Automatic garbage collection, Exception handling, etc. make it more robust.

- **Architecture Neutral:** Java is architectural neutral as it is not dependent on the architecture. In C, the size of data types may vary according to the architecture (32 bit or 64 bit) which doesn't exist in Java.

- **Interpreted:** Java uses the Just-in-time (JIT) interpreter along with the compiler for the program execution.

- **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++).

- **Multithreaded:** 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.

- **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.

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

4) What do you understand by Java virtual machine?

Java Virtual Machine is a virtual machine that enables the computer to run the Java program. JVM acts like a run-time engine which calls the main method present in the Java code. JVM is the specification which must be implemented in the computer system. The Java code is compiled by JVM to be a Bytecode which is machine independent and close to the native code.

5) What is the difference between JDK, JRE, and JVM?

JVM

JVM is an acronym for Java Virtual Machine; it is an abstract machine which provides the runtime environment in which Java bytecode can be executed. It is a specification which specifies the working of Java Virtual Machine. Its implementation has been provided by Oracle and other companies. Its implementation is known as JRE.

JVMs are available for many hardware and software platforms (so JVM is platform dependent). It is a runtime instance which is created when we run the Java class. There are three notions of the JVM: specification, implementation, and instance.

JRE

JRE stands for Java Runtime Environment. It is the implementation of JVM. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. It is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools. JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

More Details.

6) How many types of memory areas are allocated by JVM?

Many types:

1. **Class(Method) Area:** Class Area stores per-class structures such as the runtime constant pool, field, method data, and the code for methods.
2. **Heap:** It is the runtime data area in which the memory is allocated to the objects
3. **Stack:** Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return. Each thread has a private JVM stack, created at the same time as the thread. A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

4. **Program Counter Register:** PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.
5. **Native Method Stack:** It contains all the native methods used in the application.

More Details.

7) What is JIT compiler?

Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the bytecode that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) What is the platform?

A platform is the hardware or software environment in which a piece of software is executed. There are two types of platforms, software-based and hardware-based. Java provides the software-based platform.

9) What are the main differences between the Java platform and other platforms?

There are the following differences between the Java platform and other platforms.

- o Java is the software-based platform whereas other platforms may be the hardware platforms or software-based platforms.
- o Java is executed on the top of other hardware platforms whereas other platforms can only have the hardware components.

10) What gives Java its 'write once and run anywhere' nature?

The bytecode. Java compiler converts the Java programs into the class file (Byte Code) which is the intermediate language between source code and machine code. This bytecode is not platform specific and can be executed on any computer.

11) What is classloader?

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the superclass of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes, etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside $JAVA_HOME/jre/lib/ext directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the class files from the classpath. By default, the classpath is set to the current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

12) Is Empty .java file name a valid source file name?

Yes, Java allows to save our java file by **.java** only, we need to compile it by **javac .java** and run by **java classname** Let's take a simple example:

```
1.      //save by .java only
2.      class A{
3.      public static void main(String args[]){
4.      System.out.println("Hello java");
5.      }
6.      }
7.      //compile by javac .java
8.      //run by     java A
```

compile it by **javac .java**

run it by **java A**

13) Is delete, next, main, exit or null keyword in java?

No.

14) If I don't provide any arguments on the command line, then what will the value stored in the String array passed into the main() method, empty or NULL?

It is empty, but not null.

15) What if I write static public void instead of public static void?

The program compiles and runs correctly because the order of specifiers doesn't matter in Java.

16) What is the default value of the local variables?

The local variables are not initialized to any default value, neither primitives nor object references.

17) What are the various access specifiers in Java?

In Java, access specifiers are the keywords which are used to define the access scope of the method, class, or a variable. In Java, there are four access specifiers given below.

- o **Public** The classes, methods, or variables which are defined as public, can be accessed by any class or method.
- o **Protected** Protected can be accessed by the class of the same package, or by the sub-class of this class, or within the same class.
- o **Default** Default are accessible within the package only. By default, all the classes, methods, and variables are of default scope.

○ **Private** The private class, methods, or variables defined as private can be accessed within the class only.

18) What is the purpose of static methods and variables?

The methods or variables defined as static are shared among all the objects of the class. The static is the part of the class and not of the object. The static variables are stored in the class area, and we do not need to create the object to access such variables. Therefore, static is used in the case, where we need to define variables or methods which are common to all the objects of the class.

For example, In the class simulating the collection of the students in a college, the name of the college is the common attribute to all the students. Therefore, the college name will be defined as **static**.

19) What are the advantages of Packages in Java?

There are various advantages of defining packages in Java.

○ Packages avoid the name clashes.

○ The Package provides easier access control.

○ We can also have the hidden classes that are not visible outside and used by the package.

○ It is easier to locate the related classes.

20) What is the output of the following Java program?

```
1.        class Test
2.        {
3.          public static void main (String args[])
4.          {
5.            System.out.println(10 + 20 + "Javatpoint");
6.            System.out.println("Javatpoint" + 10 + 20);
7.          }
8.        }
```

The output of the above code will be

```
30Javatpoint

Javatpoint1020
```

Explanation

In the first case, 10 and 20 are treated as numbers and added to be 30. Now, their sum 30 is treated as the string and concatenated with the string **Javatpoint**. Therefore, the output will be **30Javatpoint**.

In the second case, the string Javatpoint is concatenated with 10 to be the string **Javatpoint10** which will then be concatenated with 20 to be **Javatpoint1020**.

21) What is the output of the following Java program?

```
1.      class Test
2.      {
3.          public static void main (String args[])
4.          {
5.              System.out.println(10 * 20 + "Javatpoint");
6.              System.out.println("Javatpoint" + 10 * 20);
7.          }
8.      }
```

The output of the above code will be

```
200Javatpoint

Javatpoint200
```

Explanation

In the first case, The numbers 10 and 20 will be multiplied first and then the result 200 is treated as the string and concatenated with the string **Javatpoint** to produce the output **200Javatpoint**.

In the second case, The numbers 10 and 20 will be multiplied first to be 200 because the precedence of the multiplication is higher than addition. The result 200 will be treated as the string and concatenated with the string **Javatpoint**to produce the output as **Javatpoint200**.

22) What is the output of the following Java program?

```
1.          class Test
2.          {
3.             public static void main (String args[])
4.             {
5.                for(int i=0; 0; i++)
6.                {
7.                   System.out.println("Hello Javatpoint");
8.                }
9.             }
10.         }
```

The above code will give the compile-time error because the for loop demands a boolean value in the second part and we are providing an integer value, i.e., 0.

There is given more than 50 OOPs (Object-Oriented Programming and System) interview questions. However, they have been categorized in many sections such as constructor interview questions, static interview questions, Inheritance Interview questions, Abstraction interview question, Polymorphism interview questions, etc. for better understanding.

23) What is object-oriented paradigm?

It is a programming paradigm based on objects having data and methods defined in the class to which it belongs. Object-oriented paradigm aims to incorporate the advantages of modularity and reusability. Objects are the instances of classes which interacts with one another to design applications and programs. There are the following features of the object-oriented paradigm.

- o Follows the bottom-up approach in program design.
- o Focus on data with methods to operate upon the object's data
- o Includes the concept like Encapsulation and abstraction which hides the complexities from the user and show only functionality.
- o Implements the real-time approach like inheritance, abstraction, etc.
- o The examples of the object-oriented paradigm are C++, Simula, Smalltalk, Python, C#, etc.

24) What is an object?

The Object is the real-time entity having some state and behavior. In Java, Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the **new** keyword.

25) What is the difference between an object-oriented programming language and object-based programming language?

There are the following basic differences between the object-oriented language and object-based language.

- o Object-oriented languages follow all the concepts of OOPs whereas, the object-based language doesn't follow all the concepts of OOPs like inheritance and polymorphism.
- o Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.
- o Examples of object-oriented programming are Java, C#, Smalltalk, etc. whereas the examples of object-based languages are JavaScript, VBScript, etc.

26) What will be the initial value of an object reference which is defined as an instance variable?

All object references are initialized to null in Java.

Core Java - OOPs Concepts: Constructor Interview Questions
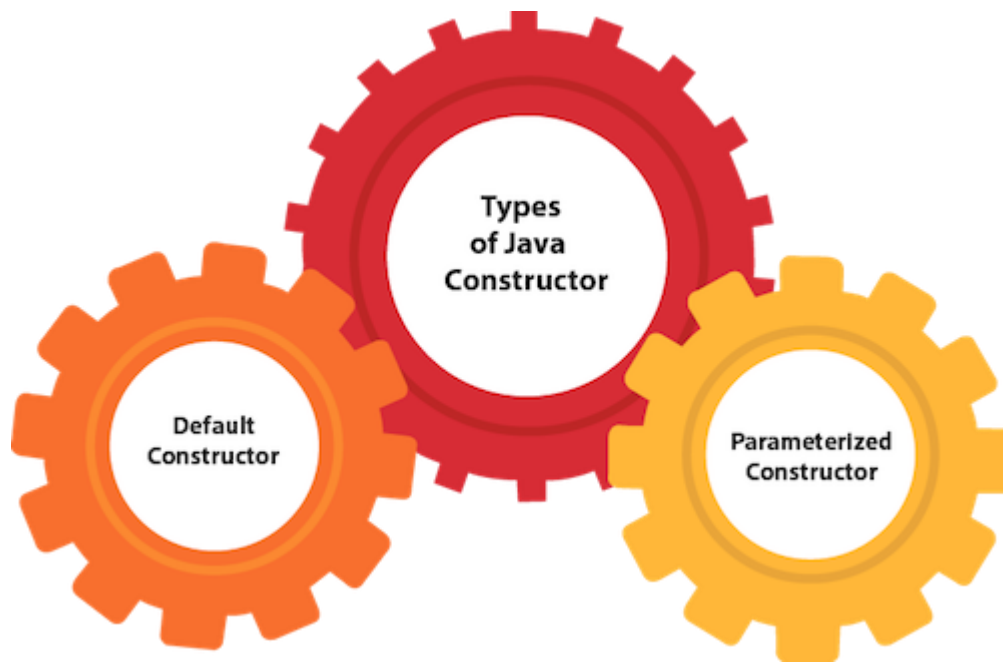
27) What is the constructor?

The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

More Details.

28) How many types of constructors are used in Java?

Based on the parameters passed in the constructors, there are two types of constructors in Java.

- o **Default Constructor:** default constructor is the one which does not accept any value. The default constructor is mainly used to initialize the instance variable with the default values. It can also be used for performing some useful task on object creation. A default constructor is invoked implicitly by the compiler if there is no constructor defined in the class.
- o **Parameterized Constructor:** The parameterized constructor is the one which can initialize the instance variables with the given values. In other words, we can say that the constructors which can accept the arguments are called parameterized constructors.

29) What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

1.   **class** Student3{
2.   **int** id;

3.   String name;

4.

5.   **void** display(){System.out.println(id+" "+name);}

6.

7.   **public static void** main(String args[]){

8.   Student3 s1=**new** Student3();

9.   Student3 s2=**new** Student3();

10.   s1.display();

11.   s2.display();
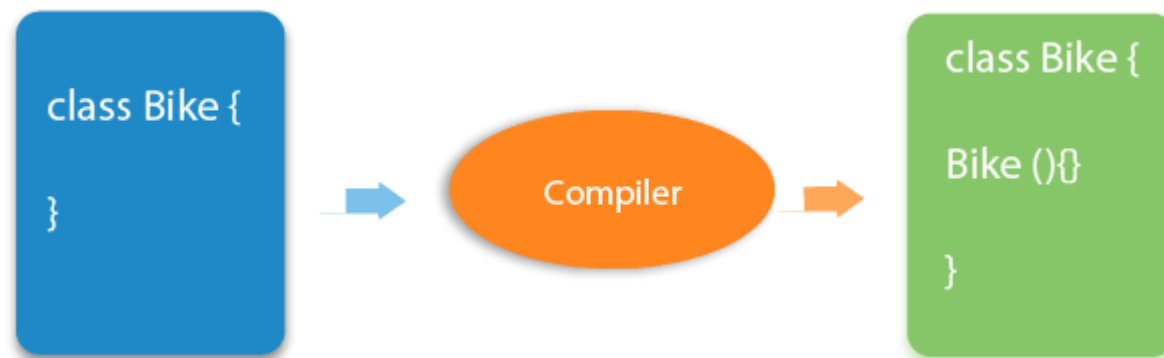
12.   }

13.   }

Output:

```
0 null

0 null
```

Explanation: In the above class, you are not creating any constructor, so compiler provides you a default constructor. Here 0 and null values are provided by default constructor.

30) Does constructor return any value?

Ans: yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor).

31)Is constructor inherited?

No, The constructor is not inherited.

32) Can you make a constructor final?

No, the constructor can't be final.

33) Can we overload the constructors?

Yes, the constructors can be overloaded by changing the number of arguments accepted by the constructor or by changing the data type of the parameters. Consider the following example.

```
1.    class Test
2.    {
3.        int i;
4.        public Test(int k)
5.        {
6.            i=k;
7.        }
8.        public Test(int k, int m)
9.        {
10.           System.out.println("Hi I am assigning the value max(k, m) to i");
11.           if(k>m)
12.           {
```

```
13.                i=k;
14.              }
15.            else
16.            {
17.                i=m;
18.            }
19.         }
20.      }
21.   public class Main
22.   {
23.      public static void main (String args[])
24.      {
25.         Test test1 = new Test(10);
26.         Test test2 = new Test(12, 15);
27.         System.out.println(test1.i);
28.         System.out.println(test2.i);
29.      }
30.   }
31.
```

In the above program, The constructor Test is overloaded with another constructor. In the first call to the constructor, The constructor with one argument is called, and i will be initialized with the value 10. However, In the second call to the constructor, The constructor with the 2 arguments is called, and i will be initialized with the value 15.

34) What do you understand by copy constructor in Java?

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

- o By constructor
- o By assigning the values of one object into another
- o By clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

```
1.    //Java program to initialize the values from one object to another
2.    class Student6{
3.        int id;
4.        String name;
5.        //constructor to initialize integer and string
6.        Student6(int i,String n){
7.        id = i;
8.        name = n;
9.        }
```

```
10.            //constructor to initialize another object
11.            Student6(Student6 s){
12.            id = s.id;
13.            name =s.name;
14.            }
15.            void display(){System.out.println(id+" "+name);}
16.
17.            public static void main(String args[]){
18.            Student6 s1 = new Student6(111,"Karan");
19.            Student6 s2 = new Student6(s1);
20.            s1.display();
21.            s2.display();
22.            }
23.        }
```

Output:

```
111 Karan
111 Karan
```

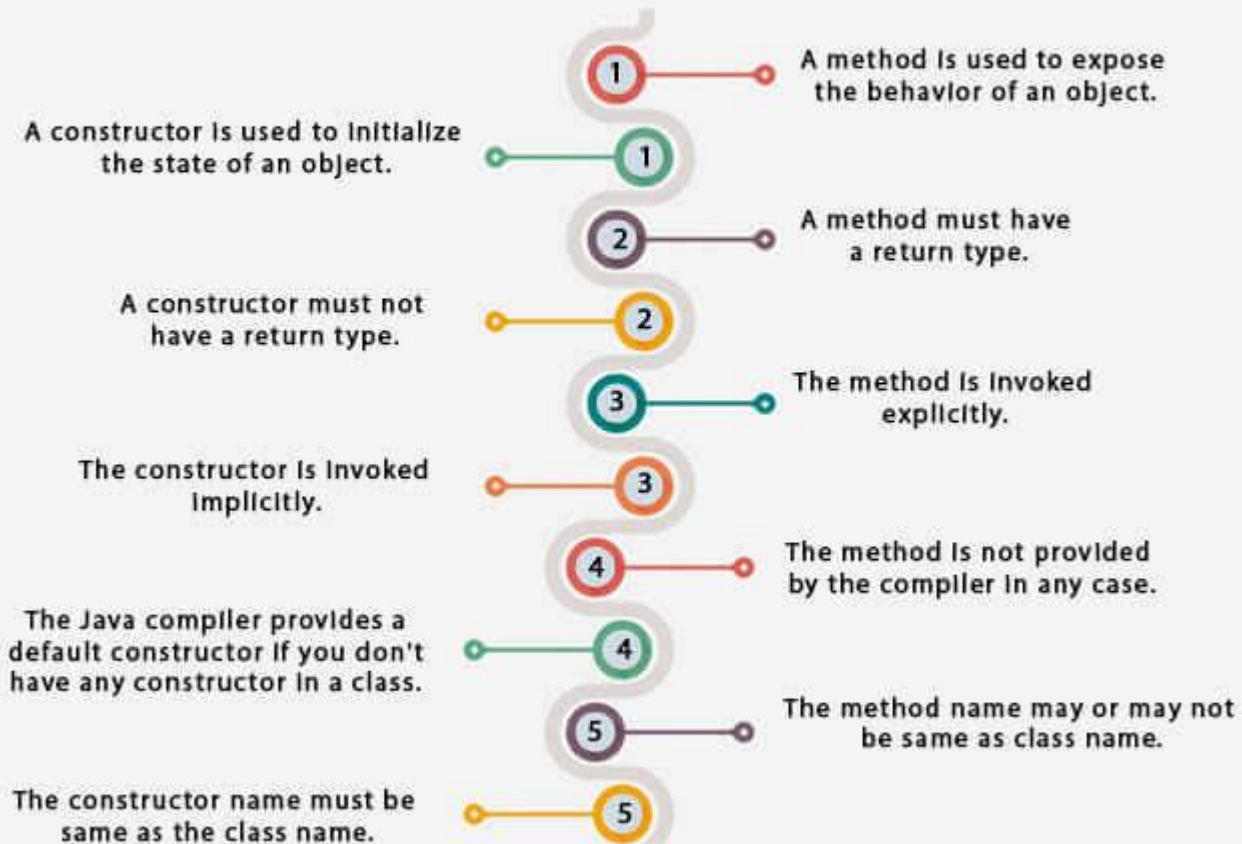## 35) What are the differences between the constructors and methods?

There are many differences between constructors and methods. They are given below.

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as class name.

Difference between constructor and method in Java

1 — A method is used to expose the behavior of an object.

1 — A constructor is used to initialize the state of an object.

2 — A method must have a return type.

2 — A constructor must not have a return type.

3 — The method is invoked explicitly.

3 — The constructor is invoked implicitly.

4 — The method is not provided by the compiler in any case.

4 — The Java compiler provides a default constructor if you don't have any constructor in a class.

5 — The method name may or may not be same as class name.

5 — The constructor name must be same as the class name.

36) What is the output of the following Java program?

```java
1.      public class Test
2.      {
3.          Test(int a, int b)
4.          {
5.              System.out.println("a = "+a+" b = "+b);
6.          }
7.          Test(int a, float b)
8.          {
9.              System.out.println("a = "+a+" b = "+b);
10.         }
11.         public static void main (String args[])
12.         {
13.             byte a = 10;
14.             byte b = 15;
15.             Test test = new Test(a,b);
16.         }
17.     }
```

The output of the following program is:

```
a = 10 b = 15
```

Here, the data type of the variables a and b, i.e., byte gets promoted to int, and the first parameterized constructor with the two integer parameters is called.

37) What is the output of the following Java program?

```
1.      class Test
2.      {
3.          int i;
4.      }
5.      public class Main
6.      {
7.          public static void main (String args[])
8.          {
9.              Test test = new Test();
10.             System.out.println(test.i);
11.         }
12.     }
```

The output of the program is 0 because the variable i is initialized to 0 internally. As we know that a default constructor is invoked implicitly if there is no constructor in the class, the variable i is initialized to 0 since there is no constructor in the class.

38) What is the output of the following Java program?

```
1.      class Test
2.      {
3.          int test_a, test_b;
4.          Test(int a, int b)
5.          {
6.          test_a = a;
7.          test_b = b;
8.          }
9.          public static void main (String args[])
10.         {
11.             Test test = new Test();
12.             System.out.println(test.test_a+" "+test.test_b);
13.         }
14.     }
```

There is a **compiler error** in the program because there is a call to the default constructor in the main method which is not present in the class. However, there is only one parameterized constructor in the class Test. Therefore, no default constructor is invoked by the constructor implicitly.

39) What is the static variable?

The static variable is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc. Static variable gets memory only once in the class area at the time of class loading. Using a static variable makes your program more memory efficient (it saves memory). Static variable belongs to the class rather than the object.

```
1.      //Program of static variable
2.
3.      class Student8{
4.         int rollno;
5.         String name;
6.         static String college ="ITS";
7.
8.         Student8(int r,String n){
```

```
9.          rollno = r;
10.          name = n;
11.          }
12.     void display (){System.out.println(rollno+" "+name+" "+college);}
13.
14.     public static void main(String args[]){
15.     Student8 s1 = new Student8(111,"Karan");
16.     Student8 s2 = new Student8(222,"Aryan");
17.
18.     s1.display();
19.     s2.display();
20.     }
21.       }
```
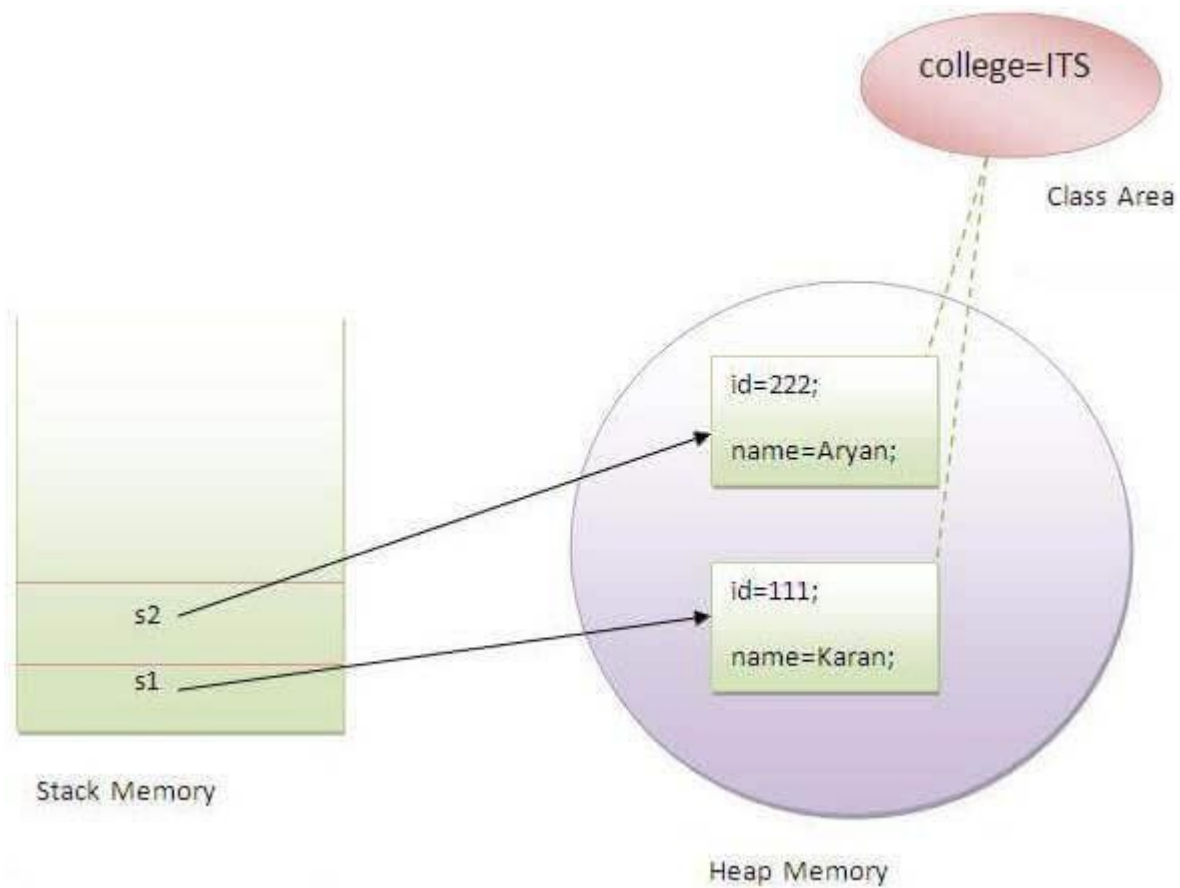
```
Output:111 Karan ITS
        222 Aryan ITS
```

college=ITS

Class Area

id=222;

name=Aryan;

id=111;

name=Karan;

s2

s1

Stack Memory

Heap Memory

More Details.

40) What is the static method?

- o   A static method belongs to the class rather than the object.
- o   There is no need to create the object to call the static methods.
- o   A static method can access and change the value of the static variable.

More Details.

41) What are the restrictions that are applied to the Java static methods?

Two main restrictions are applied to the static methods.

- o   The static method can not use non-static data member or call the non-static method directly.
- o   this and super cannot be used in static context as they are non-static.

42) Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation. More Details.

43) Can we override the static methods?

No, we can't override static methods.

44) What is the static block?

Static block is used to initialize the static data member. It is executed before the main method, at the time of classloading.

1.   **class** A2{
2.   **static**{System.out.println("static block is invoked");}
3.   **public static void** main(String args[]){
4.   System.out.println("Hello main");
5.   }
6.   }

Test it Now

Output: static block is invoked

        Hello main

More Details.

45) Can we execute a program without main() method?

Ans) Yes, one of the ways to execute the program without the main method is using static block. More Details.

46) What if the static modifier is removed from the signature of the main method?

Program compiles. However, at runtime, It throws an error "NoSuchMethodError."

47) What is the difference between static (class) method and instance method?

static or class method	instance method
1)A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
2)We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3)Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.
4)For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

48) Can we make constructors static?

As we know that the static context (method, block, or variable) belongs to the class, not the object. Since Constructors are invoked only when the object is created, there is no sense to make the constructors static. However, if you try to do so, the compiler will show the compiler error.

49) Can we make the abstract methods static in Java?

In Java, if we make the abstract methods static, It will become the part of the class, and we can directly call it which is unnecessary. Calling an undefined method is completely useless therefore it is not allowed.

50) Can we declare the static variables and methods in an abstract class?

Yes, we can declare static variables and methods in an abstract method. As we know that there is no requirement to make the object to access the static context, therefore, we can access the static context declared inside the abstract class by using the name of the abstract class. Consider the following example.

```java
1.        abstract class Test
2.        {
3.            static int i = 102;
4.            static void TestMethod()
5.            {
6.                System.out.println("hi !! I am good !!");
7.            }
8.        }
9.        public class TestClass extends Test
10.        {
11.            public static void main (String args[])
```

```
12.            {
13.               Test.TestMethod();
14.               System.out.println("i = "+Test.i);
15.            }
16.         }
```

Output

```
hi !! I am good !!
i = 102
```

51) What is this keyword in java?

The **this** keyword is a reference variable that refers to the current object. There are the various uses of this keyword in Java. It can be used to refer to current class properties such as instance methods, variable, constructors, etc. It can also be passed as an argument into the methods or constructors. It can also be returned from the method as the current class instance.
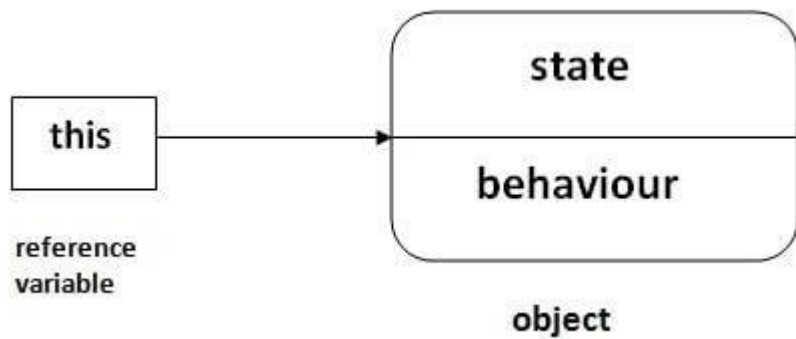
More Details.

52) What are the main uses of this keyword?

There are the following uses of **this** keyword.

- **this** can be used to refer to the current class instance variable.
- **this** can be used to invoke current class method (implicitly)
- **this()** can be used to invoke the current class constructor.
- **this** can be passed as an argument in the method call.
-
- **this** can be passed as an argument in the constructor call.
- **this** can be used to return the current class instance from the method.

53) Can we assign the reference to this variable?

No, this cannot be assigned to any value because it always points to the current class object and this is the final reference in Java. However, if we try to do so, the compiler error will be shown. Consider the following example.

```
1.    public class Test
2.    {
3.       public Test()
4.       {
5.          this = null;
6.          System.out.println("Test class constructor called");
7.       }
8.       public static void main (String args[])
9.       {
10.         Test t = new Test();
11.      }
12.    }
```

Output

```
Test.java:5: error: cannot assign a value to final variable this
            this = null;
```

```
          ^

1 error
```

54) Can this keyword be used to refer static members?

Yes, It is possible to use this keyword to refer static members because this is just a reference variable which refers to the current class object. However, as we know that, it is unnecessary to access static variables through objects, therefore, it is not the best practice to use this to refer static members. Consider the following example.

```
1.      public class Test
2.      {
3.          static int i = 10;
4.          public Test ()
5.          {
6.              System.out.println(this.i);
7.          }
8.          public static void main (String args[])
9.          {
10.             Test t = new Test();
11.         }
12.     }
```

Output

10

55) How can constructor chaining be done using this keyword?

Constructor chaining enables us to call one constructor from another constructor of the class with respect to the current class object. We can use this keyword to perform constructor chaining within the same class. Consider the following example which illustrates how can we use this keyword to achieve constructor chaining.

```
1.    public class Employee
2.    {
3.        int id,age;
4.        String name, address;
5.        public Employee (int age)
6.        {
7.            this.age = age;
8.        }
9.        public Employee(int id, int age)
10.       {
11.           this(age);
12.           this.id = id;
```

```java
13.         }
14.         public Employee(int id, int age, String name, String address)
15.         {
16.             this(id, age);
17.             this.name = name;
18.             this.address = address;
19.         }
20.         public static void main (String args[])
21.         {
22.             Employee emp = new Employee(105, 22, "Vikas", "Delhi");
23.             System.out.println("ID: "+emp.id+" Name:"+emp.name+" age:"+emp.age+" address: "+emp.address);
24.         }
25.
26.     }
```

Output

```
ID: 105 Name:Vikas age:22 address: Delhi
```

56) What are the advantages of passing this into a method instead of the current class object itself?

As we know, that this refers to the current class object, therefore, it must be similar to the current class object. However, there can be two main advantages of passing this into a method instead of the current class object.

- o this is a final variable. Therefore, this cannot be assigned to any new value whereas the current class object might not be final and can be changed.
- o this can be used in the synchronized block.

57) What is the Inheritance?

Inheritance is a mechanism by which one object acquires all the properties and behavior of another object of another class. It is used for Code Reusability and Method Overriding. The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

There are five types of inheritance in Java.

- o Single-level inheritance
- o Multi-level inheritance
- o Multiple Inheritance
- o Hierarchical Inheritance
- o Hybrid Inheritance

Multiple inheritance is not supported in Java through class.

More Details.

58) Why is Inheritance used in Java?

There are various advantages of using inheritance in Java that is given below.

- o Inheritance provides code reusability. The derived class does not need to redefine the method of base class unless it needs to provide the specific implementation of the method.
- o Runtime polymorphism cannot be achieved without using inheritance.
- o We can simulate the inheritance of classes with the real-time objects which makes OOPs more realistic.
- o Inheritance provides data hiding. The base class can hide some data from the derived class by making it private.
- o Method overriding cannot be achieved without inheritance. By method overriding, we can give a specific implementation of some basic method contained by the base class.

59) Which class is the superclass for all the classes?

The object class is the superclass of all other classes in Java.

60) Why is multiple inheritance not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java. Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

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

```
1.      class A{
2.      void msg(){System.out.println("Hello");}
3.      }
4.      class B{
5.      void msg(){System.out.println("Welcome");}
6.      }
7.      class C extends A,B{//suppose if it were
8.
9.       Public Static void main(String args[]){
10.        C obj=new C();
11.        obj.msg();//Now which msg() method would be invoked?
12.      }
13.      }
```

Test it Now

```
Compile Time Error
```

61) What is aggregation?

Aggregation can be defined as the relationship between two classes where the aggregate class contains a reference to the class it owns. Aggregation is best described as a **has-a** relationship. For example, The aggregate class Employee having various fields such as age, name, and salary also contains an object of Address class having various fields such as Address-Line 1, City, State, and pin-code. In other words, we can say that Employee (class) has an object of Address class. Consider the following example.

Address.java

```
1.    public class Address {
2.    String city,state,country;
3.
4.    public Address(String city, String state, String country) {
5.        this.city = city;
6.        this.state = state;
7.        this.country = country;
8.    }
9.
10.   }
```

Employee.java

```
1.    public class Emp {
2.    int id;
```

```java
3.          String name;
4.          Address address;
5.
6.          public Emp(int id, String name,Address address) {
7.              this.id = id;
8.              this.name = name;
9.              this.address=address;
10.         }
11.
12.         void display(){
13.         System.out.println(id+" "+name);
14.         System.out.println(address.city+" "+address.state+" "+address.country);
15.         }
16.
17.         public static void main(String[] args) {
18.         Address address1=new Address("gzb","UP","india");
19.         Address address2=new Address("gno","UP","india");
20.
21.         Emp e=new Emp(111,"varun",address1);
22.         Emp e2=new Emp(112,"arun",address2);
23.
24.         e.display();
```

```
25.        e2.display();
26.
27.     }
28.     }
```

Output

```
111 varun
gzb UP india
112 arun
gno UP india
```

62) What is composition?

Holding the reference of a class within some other class is known as composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. In other words, we can say that composition is the particular case of aggregation which represents a stronger relationship between two objects. Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.

63) What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

64) Why does Java not support pointers?

The pointer is a variable that refers to the memory address. They are not used in Java because they are unsafe(unsecured) and complex to understand.

65) What is super in java?

The **super** keyword in Java is a reference variable that is used to refer to the immediate parent class object. Whenever you create the instance of the subclass, an instance of the parent class is created implicitly which is referred by super reference variable. The super() is called in the class constructor implicitly by the compiler if there is no super or this.

```
1.    class Animal{
2.    Animal(){System.out.println("animal is created");}
3.    }
4.    class Dog extends Animal{
5.    Dog(){
6.    System.out.println("dog is created");
7.    }
8.    }
9.    class TestSuper4{
```

```
10.       public static void main(String args[]){
11.       Dog d=new Dog();
12.       }
13.       }
```

Output:

```
animal is created

dog is created
```
More Details.

66) How can constructor chaining be done by using the super keyword?

```
1.        class Person
2.        {
3.           String name,address;
4.           int age;
5.           public Person(int age, String name, String address)
6.           {
7.              this.age = age;
8.              this.name = name;
```

```java
9.              this.address = address;
10.         }
11.     }
12.     class Employee extends Person
13.     {
14.         float salary;
15.         public Employee(int age, String name, String address, float salary)
16.         {
17.             super(age,name,address);
18.             this.salary = salary;
19.         }
20.     }
21.     public class Test
22.     {
23.         public static void main (String args[])
24.         {
25.             Employee e = new Employee(22, "Mukesh", "Delhi", 90000);
26.             System.out.println("Name: "+e.name+" Salary: "+e.salary+" Age: "+e.age+" Address: "+e.address);
27.         }
28.     }
```

Output

```
Name: Mukesh Salary: 90000.0 Age: 22 Address: Delhi
```

67) What are the main uses of the super keyword?

There are the following uses of super keyword.

- o   super can be used to refer to the immediate parent class instance variable.
- o   super can be used to invoke the immediate parent class method.
- o   super() can be used to invoke immediate parent class constructor.

68) What are the differences between this and super keyword?

There are the following differences between this and super keyword.

- o   The super keyword always points to the parent class contexts whereas this keyword always points to the current class context.
- o   The super keyword is primarily used for initializing the base class variables within the derived class constructor whereas this keyword primarily used to differentiate between local and instance variables when passed in the class constructor.
- o   The super and this must be the first statement inside constructor otherwise the compiler will throw an error.

69) What is the output of the following Java program?

```
1.      class Person
2.      {
3.          public Person()
4.          {
5.              System.out.println("Person class constructor called");
6.          }
7.      }
8.      public class Employee extends Person
9.      {
10.         public Employee()
11.         {
12.             System.out.println("Employee class constructor called");
13.         }
14.         public static void main (String args[])
15.         {
16.             Employee e = new Employee();
17.         }
18.     }
```

Output

```
Person class constructor called
Employee class constructor called
```

Explanation

The super() is implicitly invoked by the compiler if no super() or this() is included explicitly within the derived class constructor. Therefore, in this case, The Person class constructor is called first and then the Employee class constructor is called.

70) Can you use this() and super() both in a constructor?

No, because this() and super() must be the first statement in the class constructor.

Example:

```
1.      public class Test{
2.         Test()
3.          {
4.             super();
5.             this();
6.          System.out.println("Test class object is created");
7.          }
8.          public static void main(String []args){
9.          Test t = new Test();
10.         }
11.      }
```

Output:

```
Test.java:5: error: call to this must be first statement in constructor
```

71)What is object cloning?

The object cloning is used to create the exact copy of an object. The clone() method of the Object class is used to clone an object. The **java.lang.Cloneable** interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

1.   **protected** Object clone() **throws** CloneNotSupportedException
2.

More Details.

Core Java - OOPs Concepts: Method Overloading Interview Questions

72) What is method overloading?

Method overloading is the polymorphism technique which allows us to create multiple methods with the same name but different signature. We can achieve method overloading in two ways.

- o   Changing the number of arguments
- o   Changing the return type

Method overloading increases the readability of the program. Method overloading is performed to figure out the program quickly.

More Details.

73) Why is method overloading not possible by changing the return type in java?

In Java, method overloading is not possible by changing the return type of the program due to avoid the ambiguity.

1.   **class** Adder{
2.   **static int** add(**int** a,**int** b){**return** a+b;}
3.   **static double** add(**int** a,**int** b){**return** a+b;}
4.   }
5.   **class** TestOverloading3{
6.   **public static void** main(String[] args){
7.   System.out.println(Adder.add(11,11));//ambiguity
8.   }}

Test it Now

Output:

```
Compile Time Error: method add(int, int) is already defined in class Adder
```
More Details.

74) Can we overload the methods by making them static?

No, We cannot overload the methods by just applying the static keyword to them(number of parameters and types are the same). Consider the following example.

```java
1.      public class Animal
2.      {
3.          void consume(int a)
4.          {
5.              System.out.println(a+" consumed!!");
6.          }
7.          static void consume(int a)
8.          {
9.              System.out.println("consumed static "+a);
10.         }
11.         public static void main (String args[])
12.         {
13.             Animal a = new Animal();
14.             a.consume(10);
15.             Animal.consume(20);
16.         }
17.     }
```

Output

```
Animal.java:7: error: method consume(int) is already defined in class Animal
    static void consume(int a)
                ^
Animal.java:15: error: non-static method consume(int) cannot be referenced from a static context
        Animal.consume(20);
              ^
2 errors
```
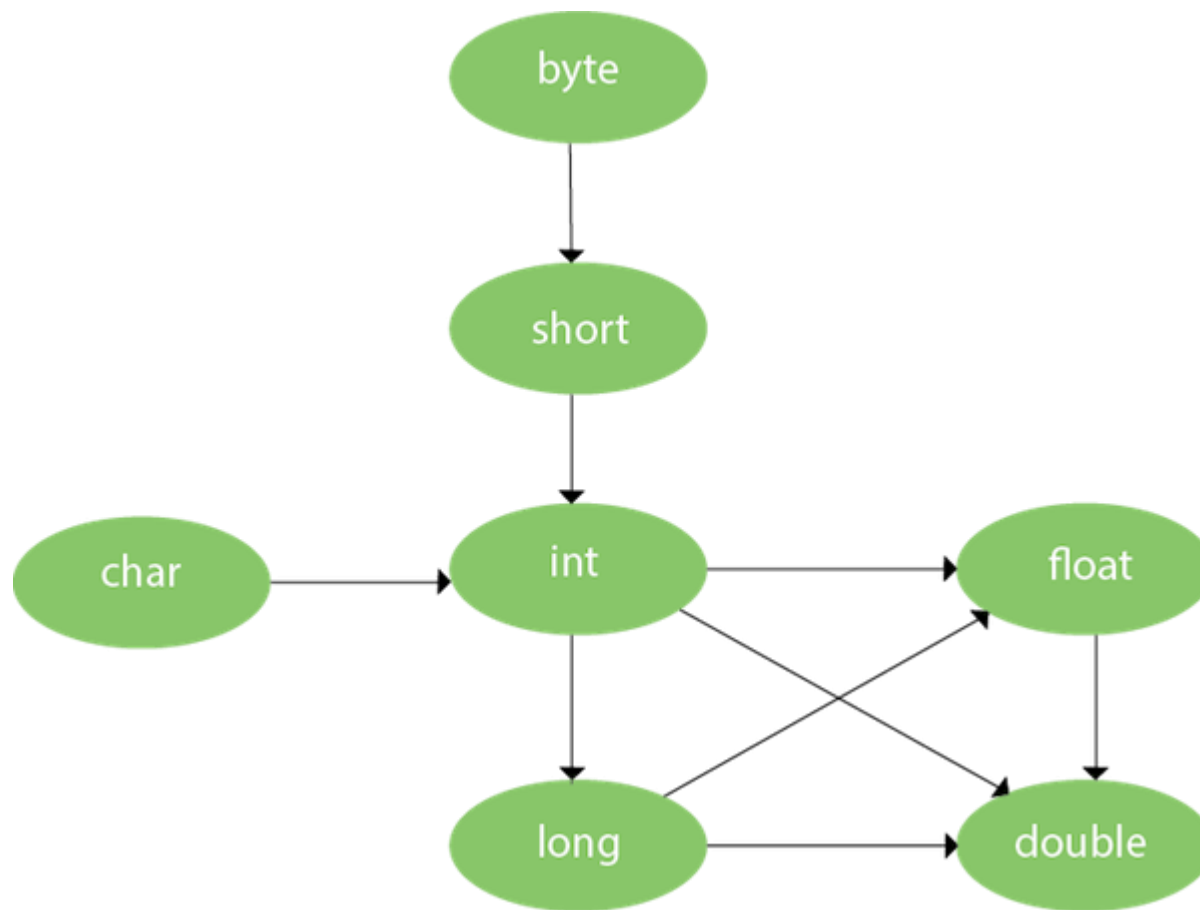
75) Can we overload the main() method?

Yes, we can have any number of main methods in a Java program by using method overloading.

More Details.

76) What is method overloading with type promotion?

By Type promotion is method overloading, we mean that one data type can be promoted to another implicitly if no exact matching is found.

As displayed in the above diagram, the byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on. Consider the following example.

```
1.          class OverloadingCalculation1{
2.            void sum(int a,long b){System.out.println(a+b);}
3.            void sum(int a,int b,int c){System.out.println(a+b+c);}
4.
5.            public static void main(String args[]){
6.            OverloadingCalculation1 obj=new OverloadingCalculation1();
7.            obj.sum(20,20);//now second int literal will be promoted to long
8.            obj.sum(20,20,20);
9.            }
10.          }
```

Test it Now

Output

40

60

77) What is the output of the following Java program?

```
1.          class OverloadingCalculation3{
2.            void sum(int a,long b){System.out.println("a method invoked");}
3.            void sum(long a,int b){System.out.println("b method invoked");}
```

4.
5.   **public static void** main(String args[]){
6.   OverloadingCalculation3 obj=**new** OverloadingCalculation3();
7.   obj.sum(20,20);//now ambiguity
8.   }
9.   }

Output

```
OverloadingCalculation3.java:7: error: reference to sum is ambiguous

obj.sum(20,20);//now ambiguity

     ^

     both method sum(int,long) in OverloadingCalculation3

     and method sum(long,int) in OverloadingCalculation3 match

1 error
```

Explanation

There are two methods defined with the same name, i.e., sum. The first method accepts the integer and long type whereas the second method accepts long and the integer type. The parameter passed that are a = 20, b = 20. We can not tell that which method will be called as there is no clear differentiation mentioned between integer literal and long literal. This is the case of ambiguity. Therefore, the compiler will throw an error.

78) What is method overriding:

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to implement the interface methods.

Rules for Method overriding

- o The method must have the same name as in the parent class.
- o The method must have the same signature as in the parent class.
- o Two classes must have an IS-A relationship between them.

More Details.

79) Can we override the static method?

No, you can't override the static method because they are the part of the class, not the object.

80) Why can we not override static method?

It is because the static method is the part of the class, and it is bound with class whereas instance method is bound with the object, and static gets memory in class area, and instance gets memory in a heap.

81) Can we override the overloaded method?

Yes.

82) Difference between method Overloading and Overriding.

Method Overloading	Method Overriding
1) Method overloading increases the readability of the program.	Method overriding provides the specific implementation of the method that is already provided by its superclass.
2) Method overloading occurs within the class.	Method overriding occurs in two classes that have IS-A relationship between them.
3) In this case, the parameters must be different.	In this case, the parameters must be the same.

83) Can we override the private methods?

No, we cannot override the private methods because the scope of private methods is limited to the class and we cannot access them outside of the class.

84) Can we change the scope of the overridden method in the subclass?

Yes, we can change the scope of the overridden method in the subclass. However, we must notice that we cannot decrease the accessibility of the method. The following point must be taken care of while changing the accessibility of the method.

- o   The private can be changed to protected, public, or default.
- o   The protected can be changed to public or default.
- o   The default can be changed to public.
- o   The public will always remain public.

85) Can we modify the throws clause of the superclass method while overriding it in the subclass?

Yes, we can modify the throws clause of the superclass method while overriding it in the subclass. However, there are some rules which are to be followed while overriding in case of exception handling.

- o   If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception, but it can declare the unchecked exception.
- o   If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

86) What is the output of the following Java program?

```
1.      class Base
2.      {
3.          void method(int a)
4.          {
5.              System.out.println("Base class method called with integer a = "+a);
6.          }
7.
8.          void method(double d)
9.          {
10.             System.out.println("Base class method called with double d ="+d);
11.         }
12.     }
13.
14.     class Derived extends Base
15.     {
16.         @Override
17.         void method(double d)
18.         {
19.             System.out.println("Derived class method called with double d ="+d);
```

```
20.          }
21.       }
22.
23.       public class Main
24.       {
25.          public static void main(String[] args)
26.          {
27.             new Derived().method(10);
28.          }
29.       }
```

Output

```
Base class method called with integer a = 10
```

Explanation

The method() is overloaded in class Base whereas it is derived in class Derived with the double type as the parameter. In the method call, the integer is passed.

87) Can you have virtual functions in Java?

Yes, all functions in Java are virtual by default.

88) What is covariant return type?

Now, since java5, it is possible to override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type. The covariant return type specifies that the return type may vary in the same direction as the subclass.

```
1.    class A{
2.    A get(){return this;}
3.    }
4.
5.    class B1 extends A{
6.    B1 get(){return this;}
7.    void message(){System.out.println("welcome to covariant return type");}
8.
9.    public static void main(String args[]){
10.   new B1().get().message();
11.   }
12.   }
```

Test it Now

Output: welcome to covariant return type

More Details.

89) What is the output of the following Java program?

```
1.      class Base
2.      {
3.         public void baseMethod()
4.         {
5.             System.out.println("BaseMethod called ...");
6.         }
7.      }
8.      class Derived extends Base
9.      {
10.        public void baseMethod()
11.        {
12.            System.out.println("Derived method called ...");
13.        }
14.     }
15.     public class Test
16.     {
17.        public static void main (String args[])
18.        {
19.            Base b = new Derived();
```

```
20.            b.baseMethod();
21.        }
22.    }
```

Output

`Derived method called ...`

Explanation

The method of Base class, i.e., baseMethod() is overridden in Derived class. In Test class, the reference variable b (of type Base class) refers to the instance of the Derived class. Here, Runtime polymorphism is achieved between class Base and Derived. At compile time, the presence of method baseMethod checked in Base class, If it presence then the program compiled otherwise the compiler error will be shown. In this case, baseMethod is present in Base class; therefore, it is compiled successfully. However, at runtime, It checks whether the baseMethod has been overridden by Derived class, if so then the Derived class method is called otherwise Base class method is called. In this case, the Derived class overrides the baseMethod; therefore, the Derived class method is called.

Core Java - OOPs Concepts: final keyword Interview Questions

90) What is the final variable?

In Java, the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its value. In other words, we can say that the final variable once assigned to a value, can never be changed after that. The final variable which is not assigned to any value can only be assigned through the class constructor.

Java Final Keyword

⇨ Stop Value Change

⇨ Stop Method Overridding

⇨ Stop Inheritance

```
1.      class Bike9{
2.       final int speedlimit=90;//final variable
3.       void run(){
4.        speedlimit=400;
5.       }
6.       public static void main(String args[]){
7.       Bike9 obj=new  Bike9();
8.       obj.run();
9.       }
```

10.   }//end of class

Output:Compile Time Error
More Details.

91) What is the final method?

If we change any method to a final method, we can't override it. More Details.

1.   **class** Bike{
2.   **final void** run(){System.out.println("running");}
3.   }
4.
5.   **class** Honda **extends** Bike{
6.   **void** run(){System.out.println("running safely with 100kmph");}
7.
8.   **public static void** main(String args[]){
9.   Honda honda= **new** Honda();
10.   honda.run();
11.   }
12.   }

Output:Compile Time Error

92) What is the final class?

If we make any class final, we can't inherit it into any of the subclasses.

1.   **final class** Bike{}

2.

3.   **class** Honda1 **extends** Bike{

4.   **void** run(){System.out.println("running safely with 100kmph");}

5.

6.   **public static void** main(String args[]){

7.   Honda1 honda= **new** Honda1();

8.   honda.run();

9.   }

10.   }

Output:Compile Time Error

More Details.

93) What is the final blank variable?

A final variable, not initialized at the time of declaration, is known as the final blank variable. We can't initialize the final blank variable directly. Instead, we have to initialize it by using the class constructor. It is useful in the case when the user has some data which must not be changed by others, for example, PAN Number. Consider the following example:

1.   **class** Student{
2.   **int** id;
3.   String name;
4.   **final** String PAN_CARD_NUMBER;
5.   ...
6.   }
More Details.

94) Can we initialize the final blank variable?

Yes, if it is not static, we can initialize it in the constructor. If it is static blank final variable, it can be initialized only in the static block.
More Details.

95) Can you declare the main method as final?

Yes, We can declare the main method as public static final void main(String[] args){}.

96) What is the output of the following Java program?

1.   class Main {
2.   public static void main(String args[]){
3.   final int i;
4.   i = 20;
5.   System.out.println(i);
6.   }
7.   }

Output

20

Explanation

Since i is the blank final variable. It can be initialized only once. We have initialized it to 20. Therefore, 20 will be printed.

97) What is the output of the following Java program?

1.   class Base

```java
2.      {
3.          protected final void getInfo()
4.          {
5.              System.out.println("method of Base class");
6.          }
7.      }
8.
9.      public class Derived extends Base
10.     {
11.         protected final void getInfo()
12.         {
13.             System.out.println("method of Derived class");
14.         }
15.         public static void main(String[] args)
16.         {
17.             Base obj = new Base();
18.             obj.getInfo();
19.         }
20.     }
```

Output

```
Derived.java:11: error: getInfo() in Derived cannot override getInfo() in Base
```

```
    protected final void getInfo()
                             ^

  overridden method is final

1 error
```

Explanation

The getDetails() method is final; therefore it can not be overridden in the subclass.

98) Can we declare a constructor as final?

The constructor can never be declared as final because it is never inherited. Constructors are not ordinary methods; therefore, there is no sense to declare constructors as final. However, if you try to do so, The compiler will throw an error.

99) Can we declare an interface as final?

No, we cannot declare an interface as final because the interface must be implemented by some class to provide its definition. Therefore, there is no sense to make an interface final. However, if you try to do so, the compiler will show an error.

100) What is the difference between the final method and abstract method?

The main difference between the final method and abstract method is that the abstract method cannot be final as we need to override them in the subclass to give its definition.

101) What is the difference between compile-time polymorphism and runtime polymorphism?

There are the following differences between compile-time polymorphism and runtime polymorphism.

SN	compile-time polymorphism	Runtime polymorphism
1	In compile-time polymorphism, call to a method is resolved at compile-time.	In runtime polymorphism, call to an overridden method is resolved at runtime.
2	It is also known as static binding, early binding, or overloading.	It is also known as dynamic binding, late binding, overriding, or dynamic method dispatch.
3	Overloading is a way to achieve compile-time polymorphism in which, we can define multiple methods or constructors with different signatures.	Overriding is a way to achieve runtime polymorphism in which, we can redefine some particular method or variable in the derived class. By using overriding, we can give some specific implementation to the base class properties in the derived class.
4	It provides fast execution because the type	It provides slower execution as compare to compile-time because

		of an object is determined at compile-time.	the type of an object is determined at run-time.
	5	Compile-time polymorphism provides less flexibility because all the things are resolved at compile-time.	Run-time polymorphism provides more flexibility because all the things are resolved at runtime.

102) What is Runtime Polymorphism?

Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

```java
1.      class Bike{
2.        void run(){System.out.println("running");}
3.       }
4.      class Splendor extends Bike{
5.        void run(){System.out.println("running safely with 60km");}
6.        public static void main(String args[]){
7.          Bike b = new Splendor();//upcasting
8.          b.run();
9.         }
```

10.   }

Output:

```
running safely with 60km.
```

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

More details.

103) Can you achieve Runtime Polymorphism by data members?

No, because method overriding is used to achieve runtime polymorphism and data members cannot be overridden. We can override the member functions but not the data members. Consider the example given below.

1.   class Bike{
2.   int speedlimit=90;
3.   }
4.   class Honda3 extends Bike{
5.   int speedlimit=150;
6.   public static void main(String args[]){
7.   Bike obj=new Honda3();

8.   System.out.println(obj.speedlimit);//90
9.   }

Output:

90
More details.

104) What is the difference between static binding and dynamic binding?

In case of the static binding, the type of the object is determined at compile-time whereas, in the dynamic binding, the type of the object is determined at runtime.

Static Binding

1.   class Dog{
2.   private void eat(){System.out.println("dog is eating...");}
3.
4.   public static void main(String args[]){
5.   Dog d1=new Dog();
6.   d1.eat();
7.   }
8.   }

Dynamic Binding

1.   **class** Animal{
2.   **void** eat(){System.out.println("animal is eating...");}
3.   }
4.
5.   **class** Dog **extends** Animal{
6.   **void** eat(){System.out.println("dog is eating...");}
7.
8.   **public static void** main(String args[]){
9.   Animal a=**new** Dog();
10.   a.eat();
11.   }
12.   }
    More details.

105) What is the output of the following Java program?

1.   **class** BaseTest
2.   {

```java
3.        void print()
4.          {
5.            System.out.println("BaseTest:print() called");
6.          }
7.        }
8.      public class Test extends BaseTest
9.        {
10.       void print()
11.         {
12.           System.out.println("Test:print() called");
13.         }
14.       public static void main (String args[])
15.         {
16.           BaseTest b = new Test();
17.           b.print();
18.         }
19.       }
```

Output

```
  Test:print() called
```

Explanation

It is an example of Dynamic method dispatch. The type of reference variable b is determined at runtime. At compile-time, it is checked whether that method is present in the Base class. In this case, it is overridden in the child class, therefore, at runtime the derived class method is called.

106) What is Java instanceOf operator?

The instanceof in Java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has a null value, it returns false. Consider the following example.

1.   **class** Simple1{
2.   **public static void** main(String args[]){
3.   Simple1 s=**new** Simple1();
4.   System.out.println(s **instanceof** Simple1);//true
5.   }
6.   }
Test it Now

Output

```
true
```

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

107) What is the abstraction?

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It displays just the essential things to the user and hides the internal information, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery. Abstraction enables you to focus on what the object does instead of how it does it. Abstraction lets you focus on what the object does instead of how it does it.

In Java, there are two ways to achieve the abstraction.

- o   Abstract Class
- o   Interface

More details.

108) What is the difference between abstraction and encapsulation?

Abstraction hides the implementation details whereas encapsulation wraps code and data into a single unit.

More details.

109) What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method. Consider the following example.

1. **abstract class** Bike{
2.   **abstract void** run();
3.   }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println(**"running safely"**);}
6. **public static void** main(String args[]){
7.   Bike obj = **new** Honda4();
8.   obj.run();
9.   }
10.   }

Test it Now

Output

```
running safely
```
More details.

110) Can there be an abstract method without an abstract class?

No, if there is an abstract method in a class, that class must be abstract.

111) Is the following program written correctly? If yes then what will be the output of the program?

```
1.    abstract class Calculate
2.    {
3.        abstract int multiply(int a, int b);
4.    }
5.
6.    public class Main
7.    {
8.        public static void main(String[] args)
9.        {
10.           int result = new Calculate()
11.           {
12.               @Override
13.               int multiply(int a, int b)
14.               {
15.                   return a*b;
16.               }
```

```
17.              }.multiply(12,32);
18.              System.out.println("result = "+result);
19.          }
20.      }
```

Yes, the program is written correctly. The Main class provides the definition of abstract method multiply declared in abstract class Calculation. The output of the program will be:

Output

```
384
```

112) Can you use abstract and final both with a method?

No, because we need to override the abstract method to provide its implementation, whereas we can't override the final method.

113) Is it possible to instantiate the abstract class?

No, the abstract class can never be instantiated even if it contains a constructor and all of its methods are implemented.

114) What is the interface?

The interface is a blueprint for a class that has static constants and abstract methods. It can be used to achieve full abstraction and multiple inheritance. It 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. Java Interface also represents the IS-A relationship. It cannot be instantiated just like the abstract class. However, we need to implement it to define its methods. Since Java 8, we can have the default, static, and private methods in an interface.

More details.

115) Can you declare an interface method static?

No, because methods of an interface are abstract by default, and we can not use static and abstract together.

116) Can the Interface be final?

No, because an interface needs to be implemented by the other class and if it is final, it can't be implemented by any class.

117) What is a marker interface?

A Marker interface can be defined as the interface which has no data member and member functions. For example, Serializable, Cloneable are marker interfaces. The marker interface can be declared as follows.

1.   **public interface** Serializable{

2.   }

118) What are the differences between abstract class and interface?

Abstract class	Interface
An abstract class can have a method body (non-abstract methods).	The interface has only abstract methods.
An abstract class can have instance variables.	An interface cannot have instance variables.
An abstract class can have the constructor.	The interface cannot have the constructor.
An abstract class can have static methods.	The interface cannot have static methods.
You can extend one abstract class.	You can implement multiple interfaces.
The abstract class **can provide the implementation of the interface**.	The Interface **can't provide the implementation of the abstract class**.
The **abstract keyword** is used to declare an abstract class.	The **interface keyword** is used to declare an interface.
An **abstract class** can extend another Java class and	An **interface** can extend another Java interface only.

implement multiple Java interfaces.	
An **abstract class** can be extended using keyword **extends**	An **interface class** can be implemented using keyword **implements**
A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

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

119) Can we define private and protected modifiers for the members in interfaces?

No, they are implicitly public.

120) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

121) How to make a read-only class in Java?

A class can be made read-only by making all of the fields private. The read-only class will have only getter methods which return the private property of the class to the main method. We cannot modify this property because there is no setter method available in the class. Consider the following example.

```java
1.        //A Java class which has only getter methods.
2.        public class Student{
3.        //private data member
4.        private String college="AKG";
5.        //getter method for college
6.        public String getCollege(){
7.        return college;
8.        }
9.        }
```

122) How to make a write-only class in Java?

A class can be made write-only by making all of the fields private. The write-only class will have only setter methods which set the value passed from the main method to the private fields. We cannot read the properties of the class because there is no getter method in this class. Consider the following example.

```
1.        //A Java class which has only setter methods.
2.        public class Student{
3.        //private data member
4.        private String college;
5.        //getter method for college
6.        public void setCollege(String college){
7.        this.college=college;
8.        }
9.        }
```

123) What are the advantages of Encapsulation in Java?

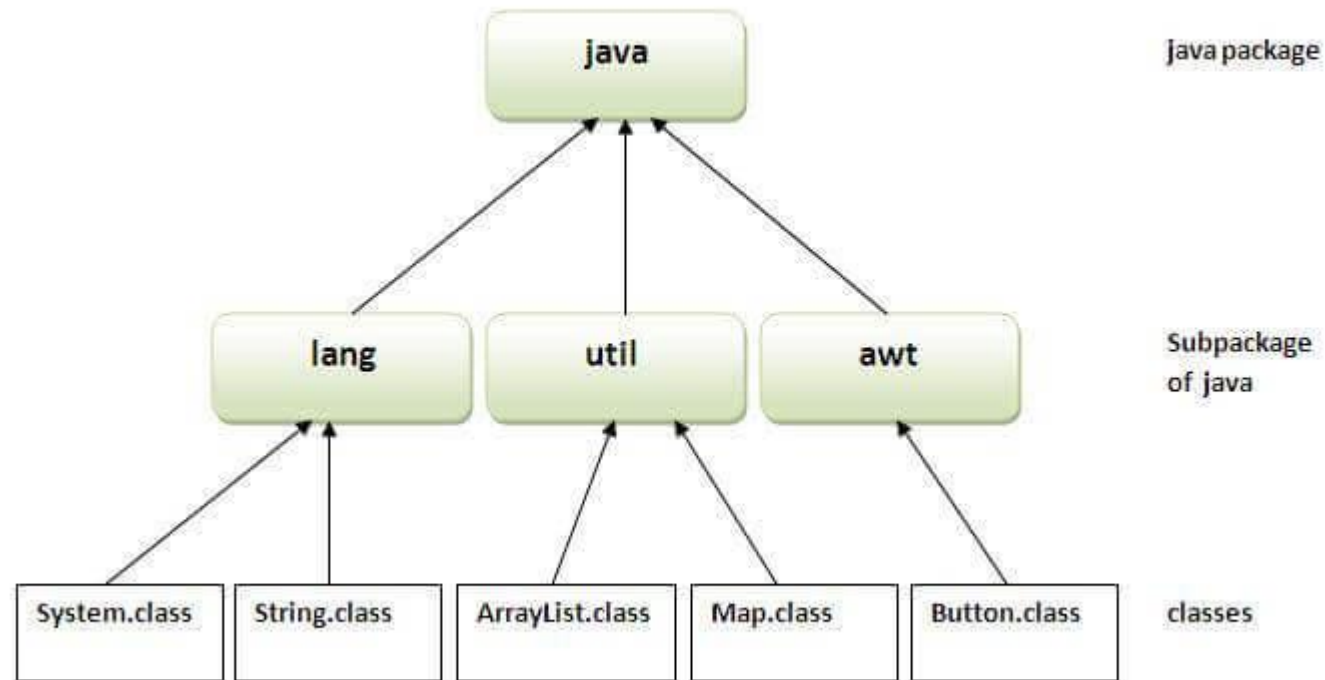There are the following advantages of Encapsulation in Java?

- By providing only the setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
- It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is easy to test. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

124) What is the package?

A package is a group of similar type of classes, interfaces, and sub-packages. It provides access protection and removes naming collision. The packages in Java can be categorized into two forms, inbuilt package, and user-defined package. There are many built-in packages such as Java, lang, awt, javax, swing, net, io, util, sql, etc. Consider the following example to create a package in Java.

```java
1.    //save as Simple.java
2.    package mypack;
3.    public class Simple{
4.     public static void main(String args[]){
5.        System.out.println("Welcome to package");
6.       }
7.     }
```

lang   util   awt — Subpackage of java

java — Java package

System.class   String.class   ArrayList.class   Map.class   Button.class — classes

More details.

125) What are the advantages of defining packages in Java?

By defining packages, we can avoid the name conflicts between the same class names defined in different packages. Packages also enable the developer to organize the similar classes more effectively. For example, one can clearly understand that the classes present in java.io package are used to perform io related operations.

126) How to create packages in Java?

If you are using the programming IDEs like Eclipse, NetBeans, MyEclipse, etc. click on **file->new->project** and eclipse will ask you to enter the name of the package. It will create the project package containing various directories such as src, etc. If you are using an editor like notepad for java programming, use the following steps to create the package.

- Define a package **package_name**. Create the class with the name **class_name** and save this file with **your_class_name.java**.

- Now compile the file by running the following command on the terminal.

1. javac -d . your_class_name.java

   The above command creates the package with the name **package_name** in the present working directory.

- Now, run the class file by using the absolute class file name, like following.

1. java package_name.class_name

127) How can we access some class in another class in Java?

There are two ways to access a class in another class.

- **By using the fully qualified name:** To access a class in a different package, either we must use the fully qualified name of that class, or we must import the package containing that class.
- **By using the relative path**, We can use the path of the class that is related to the package that contains our class. It can be the same or subpackage.

128) Do I need to import java.lang package any time? Why?

No. It is by default loaded internally by the JVM.

129) Can I import same package/class twice? Will the JVM load the package twice at runtime?

One can import the same package or the same class multiple times. Neither compiler nor JVM complains about it. However, the JVM will internally load the class only once no matter how many times you import the same class.

130) What is the static import?

By static import, we can access the static members of a class directly, and there is no to qualify it with the class name.

More details.

Java: Exception Handling Interview Questions

There is given a list of exception handling interview questions with answers. If you know any exception handling interview question, kindly post it in the comment section.

131) How many types of exception can occur in a Java program?

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

- **Checked Exception:** Checked exceptions are the one which are checked at compile-time. For example, SQLException, ClassNotFoundException, etc.

- **Unchecked Exception:** Unchecked exceptions are the one which are handled at runtime because they can not be checked at compile-time. For example, ArithmaticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.

- **Error:** Error cause the program to exit since they are not recoverable. For Example, OutOfMemoryError, AssertionError, etc.
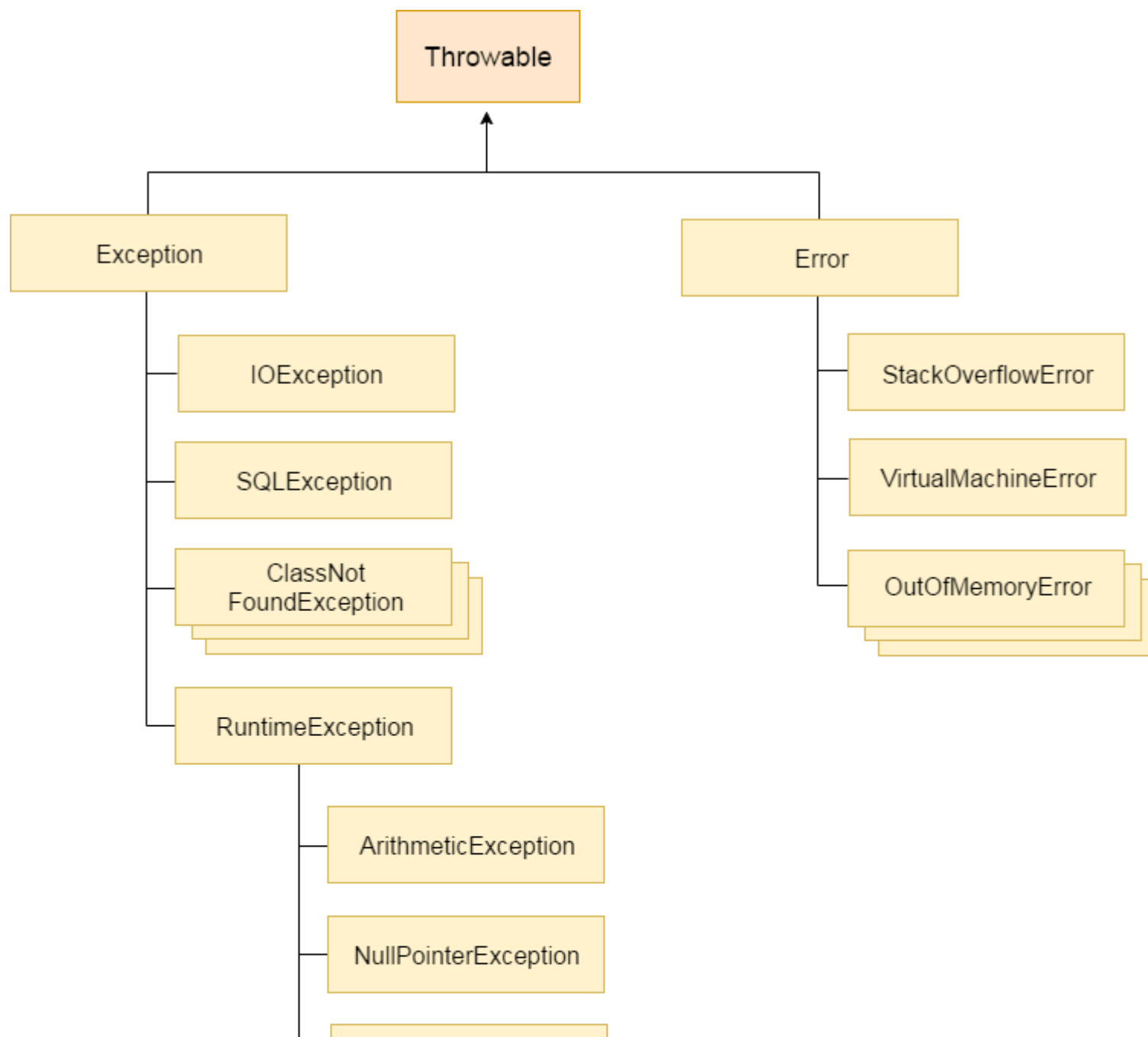
132) What is Exception Handling?

Exception Handling is a mechanism that is used to handle runtime errors. It is used primarily to handle checked exceptions. Exception handling maintains the normal flow of the program. There are mainly two types of exceptions: checked and unchecked. Here, the error is considered as the unchecked exception.

More details.

133) Explain the hierarchy of Java Exception classes?

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:

```
                          ┌──────────────┐
                          │   Throwable  │
                          └──────────────┘
                                 ▲
          ┌──────────────────────┴──────────────────────┐
  ┌──────────────┐                              ┌──────────────┐
  │  Exception   │                              │     Error    │
  └──────────────┘                              └──────────────┘
        │                                              │
        │    ┌──────────────┐                          │    ┌──────────────────┐
        ├────│  IOException  │                          ├────│ StackOverflowError│
        │    └──────────────┘                          │    └──────────────────┘
        │                                              │
        │    ┌──────────────┐                          │    ┌──────────────────┐
        ├────│  SQLException │                          ├────│VirtualMachineError│
        │    └──────────────┘                          │    └──────────────────┘
        │                                              │
        │    ┌──────────────┐                          │    ┌──────────────────┐
        ├────│   ClassNot    │                          └────│ OutOfMemoryError  │
        │    │ FoundException│                               └──────────────────┘
        │    └──────────────┘
        │
        │    ┌──────────────┐
        └────│RuntimeException│
             └──────────────┘
                   │
                   │    ┌──────────────────┐
                   ├────│ ArithmeticException│
                   │    └──────────────────┘
                   │
                   │    ┌──────────────────┐
                   └────│NullPointerException│
                        └──────────────────┘
```

134) What is the difference between Checked Exception and Unchecked Exception?

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions, e.g., IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions, e.g., ArithmeticException, NullPointerException, etc. Unchecked exceptions are not checked at compile-time.

More details.

135) What is the base class for Error and Exception?

The Throwable class is the base class for Error and Exception.

136) Is it necessary that each try block must be followed by a catch block?

It is not necessary that each try block must be followed by a catch block. It should be followed by either a catch block OR a finally block. So whatever exceptions are likely to be thrown should be declared in the throws clause of the method. Consider the following example.

```
1.      public class Main{
2.          public static void main(String []args){
3.            try{
4.                int a = 1;
5.                System.out.println(a/0);
6.            }
7.            finally
8.            {
9.                System.out.println("rest of the code...");
10.           }
11.          }
12.      }
13.
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...
```

137) What is the output of the following Java program?

```java
1.      public class ExceptionHandlingExample {
2.      public static void main(String args[])
3.      {
4.          try
5.          {
6.              int a = 1/0;
7.              System.out.println("a = "+a);
8.          }
9.          catch(Exception e){System.out.println(e);}
10.         catch(ArithmeticException ex){System.out.println(ex);}
11.     }
12.     }
```

Output

```
ExceptionHandlingExample.java:10: error: exception ArithmeticException has already been caught
        catch(ArithmeticException ex){System.out.println(ex);}
        ^
1 error
```

Explanation

ArithmaticException is the subclass of Exception. Therefore, it can not be used after Exception. Since Exception is the base class for all the exceptions, therefore, it must be used at last to handle the exception. No class can be used after this.

138) What is finally block?

The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. In other words, we can say that finally block is the block which is always executed. Finally block follows try or catch block. If you don't handle the exception, before terminating the program, JVM runs finally block, (if any). The finally block is mainly used to place the cleanup code such as closing a file or closing a connection. Here, we must know that for each try block there can be zero or more catch blocks, but only one finally block. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

Program code

exception occurred?

no

yes

exception handled?

no

yes

finally block is executed
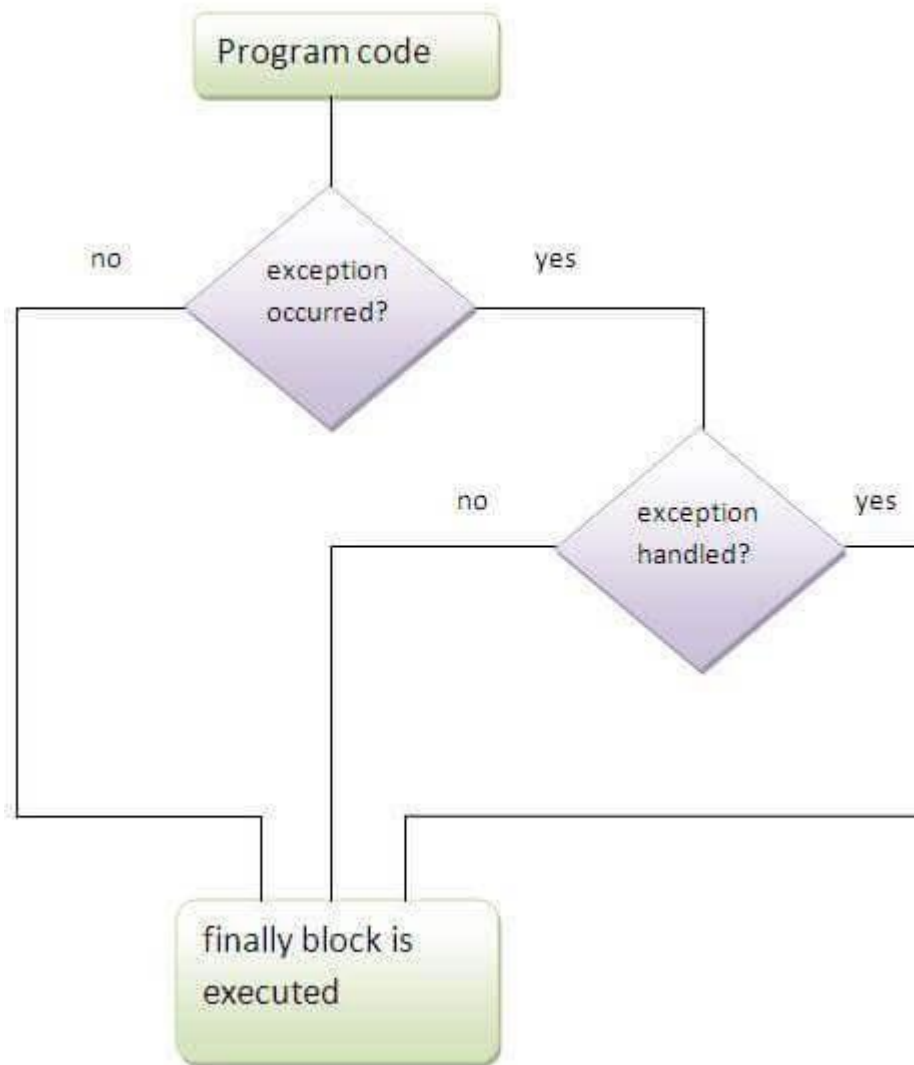
139) Can finally block be used without a catch?

Yes, According to the definition of finally block, it must be followed by a try or catch block, therefore, we can use try block instead of catch.

140) Is there any case when finally will not be executed?

Finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort).

141) What is the difference between throw and throws?

throw keyword	throws keyword
1) The **throw** keyword is used to throw an exception explicitly.	The **throws** keyword is used to declare an exception.
2) The checked exceptions cannot be propagated with throw only.	The checked exception can be propagated with throws

3) The **throw** keyword is followed by an instance.	The **throws** keyword is followed by class.
4) The **throw** keyword is used within the method.	The **throws** keyword is used with the method signature.
5) You cannot throw multiple exceptions.	You can declare multiple exceptions, e.g., public void method()throws IOException, SQLException.

More details.

142) What is the output of the following Java program?

```
1.      public class Main{
2.        public static void main(String []args){
3.          try
4.          {
5.            throw 90;
6.          }
7.          catch(int e){
8.            System.out.println("Caught the exception "+e);
9.          }
```

10.
11.   }
12.   }

Output

```
Main.java:6: error: incompatible types: int cannot be converted to Throwable
             throw 90;
             ^
Main.java:8: error: unexpected type
          catch(int e){
                 ^
  required: class
  found:    int
2 errors
```

Explanation

In Java, the throwable objects can only be thrown. If we try to throw an integer object, The compiler will show an error since we can not throw basic data type from a block of code.

143) What is the output of the following Java program?

```java
1.      class Calculation extends Exception
2.      {
3.          public Calculation()
4.          {
5.              System.out.println("Calculation class is instantiated");
6.          }
7.          public void add(int a, int b)
8.          {
9.              System.out.println("The sum is "+(a+b));
10.         }
11.     }
12.     public class Main{
13.         public static void main(String []args){
14.             try
15.             {
16.                 throw new Calculation();
17.             }
18.             catch(Calculation c){
19.                 c.add(10,20);
20.             }
```

```
21.          }
22.        }
```

Output

```
Calculation class is instantiated

The sum is 30
```

Explanation

The object of Calculation is thrown from the try block which is caught in the catch block. The add() of Calculation class is called with the integer values 10 and 20 by using the object of this class. Therefore there sum 30 is printed. The object of the Main class can only be thrown in the case when the type of the object is throwable. To do so, we need to extend the throwable class.

144) Can an exception be rethrown?

Yes.

145) Can subclass overriding method declare an exception if parent class method doesn't throw an exception?

Yes but only unchecked exception not checked.

More details.

146) What is exception propagation?

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This procedure is called exception propagation. By default, checked exceptions are not propagated.

```java
1.    class TestExceptionPropagation1{
2.      void m(){
3.        int data=50/0;
4.      }
5.      void n(){
6.        m();
7.      }
8.      void p(){
9.       try{
10.        n();
11.       }catch(Exception e){System.out.println("exception handled");}
12.      }
13.      public static void main(String args[]){
14.       TestExceptionPropagation1 obj=new TestExceptionPropagation1();
15.       obj.p();
```
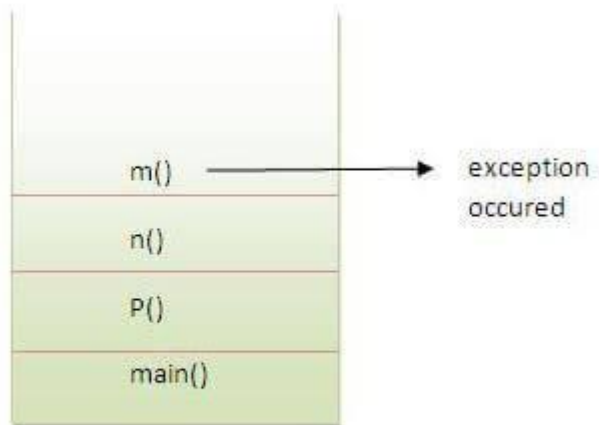
16.   System.out.println("normal flow...");
17.   }
18.   }

Output:

```
exception handled
normal flow...
```


Call Stack

More details.

147) What is the output of the following Java program?

```java
1.      public class Main
2.      {
3.         void a()
4.         {
5.            try{
6.            System.out.println("a(): Main called");
7.            b();
8.            }catch(Exception e)
9.            {
10.              System.out.println("Exception is caught");
11.           }
12.        }
13.        void b() throws Exception
14.        {
15.         try{
16.            System.out.println("b(): Main called");
17.            c();
18.        }catch(Exception e){
19.            throw new Exception();
```

```
20.            }
21.             finally
22.            {
23.                System.out.println("finally block is called");
24.            }
25.            }
26.            void c() throws Exception
27.            {
28.                throw new Exception();
29.            }
30.
31.            public static void main (String args[])
32.            {
33.                Main m = new Main();
34.                m.a();
35.            }
36.        }
```

Output

```
a(): Main called
b(): Main called
finally block is called
```

```
Exception is caught
```

Explanation

In the main method, a() of Main is called which prints a message and call b(). The method b() prints some message and then call c(). The method c() throws an exception which is handled by the catch block of method b. However, It propagates this exception by using **throw Exception()** to be handled by the method a(). As we know, finally block is always executed therefore the finally block in the method b() is executed first and prints a message. At last, the exception is handled by the catch block of the method a().

148) What is the output of the following Java program?

1.   **public class** Calculation
2.   {
3.   **int** a;
4.   **public** Calculation(**int** a)
5.   {
6.   **this**.a = a;
7.   }
8.   **public int** add()
9.   {
10.   a = a+10;
11.   **try**

```java
12.             {
13.                 a = a+10;
14.                 try
15.                 {
16.                     a = a*10;
17.                     throw new Exception();
18.                 }catch(Exception e){
19.                     a = a - 10;
20.                 }
21.             }catch(Exception e)
22.             {
23.                 a = a - 10;
24.             }
25.             return a;
26.         }
27.
28.         public static void main (String args[])
29.         {
30.             Calculation c = new Calculation(10);
31.             int result = c.add();
32.             System.out.println("result = "+result);
33.         }
```

34.   }

Output

```
result = 290
```

Explanation

The instance variable a of class Calculation is initialized to 10 using the class constructor which is called while instantiating the class. The add method is called which returns an integer value result. In add() method, a is incremented by 10 to be 20. Then, in the first try block, 10 is again incremented by 10 to be 30. In the second try block, a is multiplied by 10 to be 300. The second try block throws the exception which is caught by the catch block associated with this try block. The catch block again alters the value of a by decrementing it by 10 to make it 290. Thus the add() method returns 290 which is assigned to result. However, the catch block associated with the outermost try block will never be executed since there is no exception which can be handled by this catch block.

Java: String Handling Interview Questions

There is given a list of string handling interview questions with short and pointed answers. If you know any string handling interview question, kindly post it in the comment section.

149) What is String Pool?

String pool is the space reserved in the heap memory that can be used to store the strings. The main advantage of using the String pool is whenever we create a string literal; the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. Therefore, it saves the memory by avoiding the duplicacy.

"Welcome"

s2

s1

String constant pool
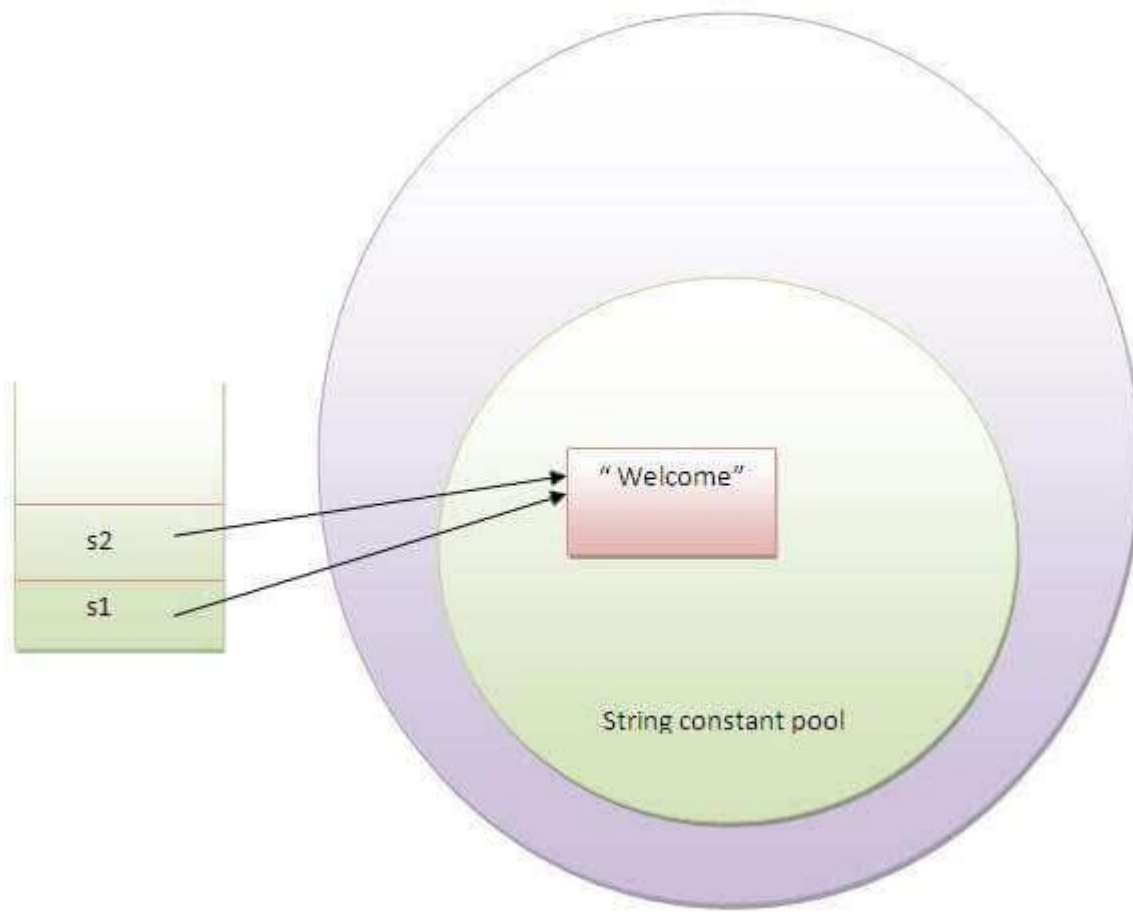
Heap

150) What is the meaning of immutable regarding String?

The simple meaning of immutable is unmodifiable or unchangeable. In Java, String is immutable, i.e., once string object has been created, its value can't be changed. Consider the following example for better understanding.

```
1.      class Testimmutablestring{
2.       public static void main(String args[]){
3.        String s="Sachin";
4.        s.concat(" Tendulkar");//concat() method appends the string at the end
5.        System.out.println(s);//will print Sachin because strings are immutable objects
6.       }
7.      }
```

Test it Now

Output:

```
Sachin
```
More details.

151) Why are the objects immutable in java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

"Sachin"

"Sachin Tendulkar"

String constant pool

s

Heap
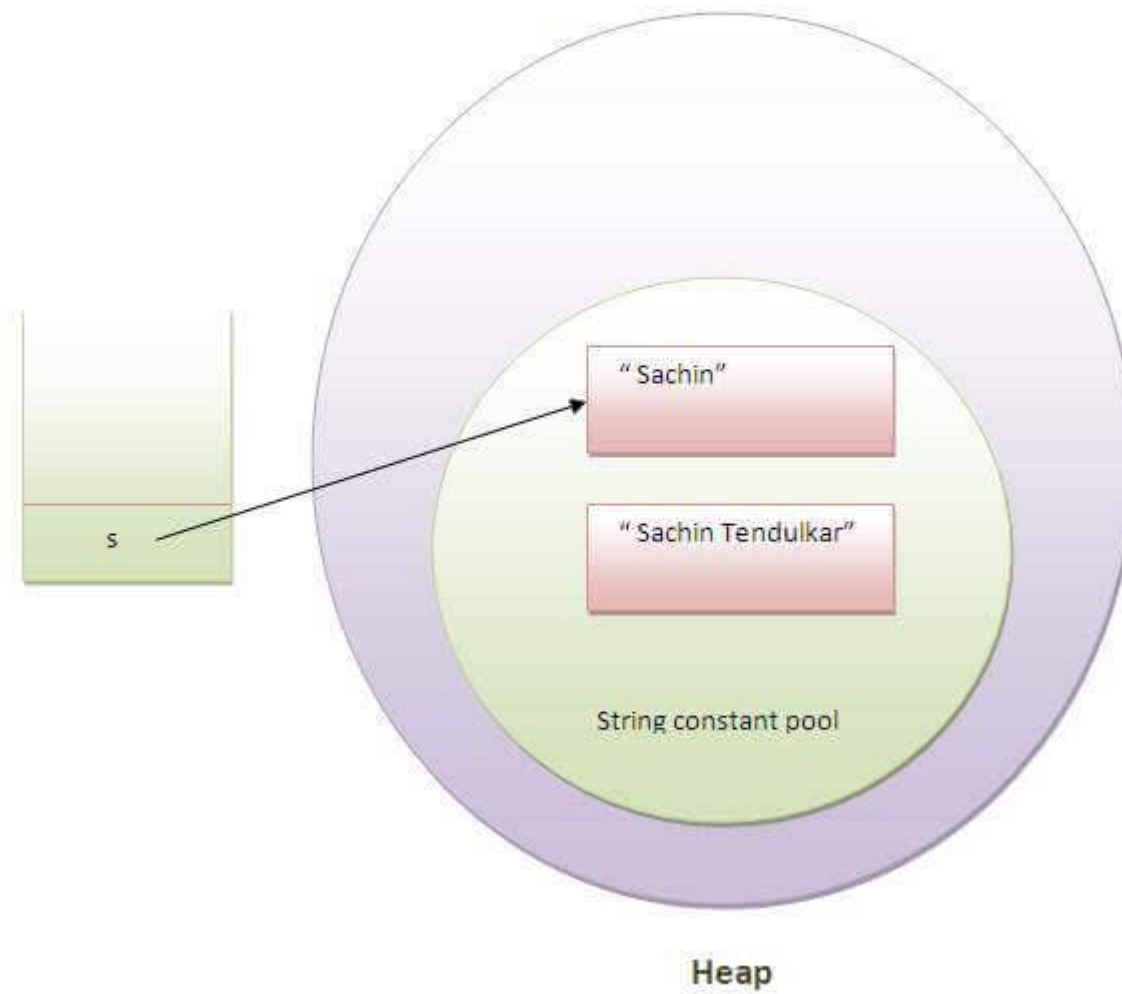
More details.

152) How many ways can we create the string object?

1) String Literal

Java String literal is created by using double quotes. For Example:

1.   String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. String objects are stored in a special memory area known as the **string constant pool** For example:

1.   String s1="Welcome";
2.   String s2="Welcome";//It doesn't create a new instance

2) By new keyword

1.   String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the constant string pool. The variable s will refer to the object in a heap (non-pool).

153) How many objects will be created in the following code?

1.   String s1="Welcome";
2.   String s2="Welcome";
3.   String s3="Welcome";

Only one object will be created using the above code because strings in Java are immutable.

More details.

154) Why java uses the concept of the string literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

More details.

155) How many objects will be created in the following code?

1.   String s = **new** String("Welcome");

Two objects, one in string constant pool and other in non-pool(heap).

156) What is the output of the following Java program?

```
1.      public class Test
2.
3.       public static void main (String args[])
4.        {
5.           String a = new String("Sharma is a good player");
6.           String b = "Sharma is a good player";
7.           if(a == b)
8.           {
9.              System.out.println("a == b");
10.          }
11.          if(a.equals(b))
12.          {
13.             System.out.println("a equals b");
14.          }
15.       }
```

Output

```
a equals b
```

Explanation

The operator **==** also check whether the references of the two string objects are equal or not. Although both of the strings contain the same content, their references are not equal because both are created by different ways(Constructor and String literal) therefore, **a == b** is unequal. On the other hand, the equal() method always check for the content. Since their content is equal hence, **a equals b** is printed.

157) What is the output of the following Java program?

1.   **public class** Test

2.   {

3.   **public static void** main (String args[])

4.   {

5.   String s1 = "Sharma is a good player";

6.   String s2 = **new** String("Sharma is a good player");

7.   s2 = s2.intern();

8.   System.out.println(s1 ==s2);

9.   }

10.   }

Output

```
true
```

Explanation

The intern method returns the String object reference from the string pool. In this case, s1 is created by using string literal whereas, s2 is created by using the String pool. However, s2 is changed to the reference of s1, and the operator **==** returns true.

158) What are the differences between String and StringBuffer?

The differences between the String and StringBuffer is given in the table below.

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	The String is slow and consumes more memory when you concat too many strings because every time it creates a new instance.	The StringBuffer is fast and consumes less memory when you cancat strings.
3)	The String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	The StringBuffer class doesn't override the equals() method of Object class.

159) What are the differences between StringBuffer and StringBuilder?

The differences between the StringBuffer and StringBuilder is given below.

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.

160) How can we create an immutable class in Java?

We can create an immutable class by defining a final class having all of its members as final. Consider the following example.

```
1.      public final class Employee{
2.      final String pancardNumber;
3.
4.      public Employee(String pancardNumber){
5.      this.pancardNumber=pancardNumber;
6.      }
7.
8.      public String getPancardNumber(){
```

```
9.          return pancardNumber;
10.         }
11.
12.         }
```
More details.

161) What is the purpose of toString() method in Java?

The toString() method returns the string representation of an object. If you print any object, java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object, etc. depending upon your implementation. By overriding the toString() method of the Object class, we can return the values of the object, so we don't need to write much code. Consider the following example.

```
1.          class Student{
2.           int rollno;
3.           String name;
4.           String city;
5.
6.           Student(int rollno, String name, String city){
7.            this.rollno=rollno;
8.            this.name=name;
9.            this.city=city;
```

```
10.        }
11.
12.        public String toString(){//overriding the toString() method
13.         return rollno+" "+name+" "+city;
14.        }
15.        public static void main(String args[]){
16.          Student s1=new Student(101,"Raj","lucknow");
17.          Student s2=new Student(102,"Vijay","ghaziabad");
18.
19.          System.out.println(s1);//compiler writes here s1.toString()
20.          System.out.println(s2);//compiler writes here s2.toString()
21.        }
22.         }
```

Output:

```
101 Raj lucknow

102 Vijay ghaziabad
```
More details.

162) Why CharArray() is preferred over String to store the password?

String stays in the string pool until the garbage is collected. If we store the password into a string, it stays in the memory for a longer period, and anyone having the memory-dump can extract the password as clear text. On the other hand, Using CharArray allows us to set it to blank whenever we are done with the password. It avoids the security threat with the string by enabling us to control the memory.

163) Write a Java program to count the number of words present in a string?

Program:

```
1.        public class Test
2.      {
3.        public static void main (String args[])
4.         {
5.            String s = "Sharma is a good player and he is so punctual";
6.            String words[] = s.split(" ");
7.            System.out.println("The Number of words present in the string are : "+words.length);
8.         }
9.      }
```

168) What are the advantages of Java inner classes?

There are two types of advantages of Java inner classes.

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

169) What is a nested class?

The nested class can be defined as the class which is defined inside another class or interface. We use the nested class to logically group classes and interfaces in one place so that it can be more readable and maintainable. A nested class can access all the data members of the outer class including private data members and methods. The syntax of the nested class is defined below.

```
1.    class Java_Outer_class{
2.     //code
3.     class Java_Nested_class{
4.      //code
5.     }
6.    }
7.
```

There are two types of nested classes, static nested class, and non-static nested class. The non-static nested class can also be called as inner-class

170) What are the disadvantages of using inner classes?

There are the following main disadvantages of using inner classes.

- o Inner classes increase the total number of classes used by the developer and therefore increases the workload of JVM since it has to perform some routine operations for those extra classes which result in slower performance.
- o IDEs provide less support to the inner classes as compare to the top level classes and therefore it annoys the developers while working with inner classes.

171) What are the types of inner classes (non-static nested class) used in Java?

There are mainly three types of inner classes used in Java.

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within the method.

172) Is there any difference between nested classes and inner classes?

Yes, inner classes are non-static nested classes. In other words, we can say that inner classes are the part of nested classes.

More details.

173) Can we access the non-final local variable, inside the local inner class?

No, the local variable must be constant if you want to access it in the local inner class.

175) What are anonymous inner classes?

Anonymous inner classes are the classes that are automatically declared and instantiated within an expression. We cannot apply different access modifiers to them. Anonymous class cannot be static, and cannot define any static fields, method, or class. In other words, we can say that it a class without the name and can have only one object that is created by its definition. Consider the following example.

1.   **abstract class** Person{
2.   **abstract void** eat();
3.   }
4.   **class** TestAnonymousInner{
5.   **public static void** main(String args[]){
6.   Person p=**new** Person(){

7.   **void** eat(){System.out.println("nice fruits");}

8.   };

9.   p.eat();

10.   }

11.   }

Output:

```
nice fruits
```

Consider the following example for the working of the anonymous class using interface.

1.   **interface** Eatable{

2.   **void** eat();

3.   }

4.   **class** TestAnnonymousInner1{

5.   **public static void** main(String args[]){

6.   Eatable e=**new** Eatable(){

7.   **public void** eat(){System.out.println("nice fruits");}

8.   };

9.   e.eat();

10.   }

11.   }

Output:

```
nice fruits
```

176) What is the nested interface?

An Interface that is declared inside the interface or class is known as the nested interface. It is static by default. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The external interface or class must refer to the nested interface. It can't be accessed directly. The nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class. The syntax of the nested interface is given as follows.

1.   **interface** interface_name{
2.   …
3.   **interface** nested_interface_name{
4.   …
5.   }
6.   }
7.

More details.

177) Can a class have an interface?

Yes, an interface can be defined within the class. It is called a nested interface.

More details.

178) Can an Interface have a class?

Yes, they are static implicitly.

More details.

179) What is Garbage Collection?

Garbage collection is a process of reclaiming the unused runtime objects. It is performed for memory management. In other words, we can say that It is the process of removing unused objects from the memory to free up space and make this space available for Java Virtual Machine. Due to garbage collection java gives 0 as output to a variable whose value is not set, i.e., the variable has been defined but not initialized. For this purpose, we were using free() function in the C language and delete() in C++. In Java, it is performed automatically. So, java provides better memory management.

More details.

180) What is gc()?

The gc() method is used to invoke the garbage collector for cleanup processing. This method is found in System and Runtime classes. This function explicitly makes the Java Virtual Machine free up the space occupied by the unused objects so that it can be utilized or reused. Consider the following example for the better understanding of how the gc() method invoke the garbage collector.

```
1.   public class TestGarbage1{
2.    public void finalize(){System.out.println("object is garbage collected");}
3.    public static void main(String args[]){
4.     TestGarbage1 s1=new TestGarbage1();
5.     TestGarbage1 s2=new TestGarbage1();
6.      s1=null;
7.      s2=null;
8.      System.gc();
9.     }
10.    }
```

Test it Now

```
        object is garbage collected
        object is garbage collected
```

181) How is garbage collection controlled?

Garbage collection is managed by JVM. It is performed when there is not enough space in the memory and memory is running low. We can externally call the System.gc() for the garbage collection. However, it depends upon the JVM whether to perform it or not.

182) How can an object be unreferenced?

There are many ways:

- By nulling the reference
- By assigning a reference to another
- By anonymous object etc.

1) By nulling a reference:

1. Employee e=**new** Employee();
2. e=**null**;

2) By assigning a reference to another:

1. Employee e1=**new** Employee();
2. Employee e2=**new** Employee();
3. e1=e2;//now the first object referred by e1 is available for garbage collection

3) By anonymous object:

1.   **new** Employee();

183) What is the purpose of the finalize() method?

The finalize() method is invoked just before the object is garbage collected. It is used to perform cleanup processing. The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created an object without new, you can use the finalize method to perform cleanup processing (destroying remaining objects). The cleanup processing is the process to free up all the resources, network which was previously used and no longer needed. It is essential to remember that it is not a reserved keyword, finalize method is present in the object class hence it is available in every class as object class is the superclass of every class in java. Here, we must note that neither finalization nor garbage collection is guaranteed. Consider the following example.

```
1.    public class FinalizeTest {
2.        int j=12;
3.        void add()
4.        {
5.            j=j+12;
6.            System.out.println("J="+j);
7.        }
8.        public void finalize()
9.        {
```

```
10.         System.out.println("Object is garbage collected");
11.     }
12.     public static void main(String[] args) {
13.         new FinalizeTest().add();
14.         System.gc();
15.         new FinalizeTest().add();
16.     }
17. }
18.
```

184) Can an unreferenced object be referenced again?

Yes,

185) What kind of thread is the Garbage collector thread?

Daemon thread.

186) What is the difference between final, finally and finalize?

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method, and variable. The final class can't be inherited, final method can't be overridden, and final variable value can't be changed.	Finally is used to place important code, it will be executed whether an exception is handled or not.	Finalize is used to perform clean up processing just before an object is garbage collected.
2)	Final is a keyword.	Finally is a block.	Finalize is a method.

187) What is the purpose of the Runtime class?

Java Runtime class is used to interact with a java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory, etc. There is only one instance of java.lang.Runtime class is available for one java application. The Runtime.getRuntime() method returns the singleton instance of Runtime class.

190) What do you understand by an IO stream?

The stream is a sequence of data that flows from source to destination. It is composed of bytes. In Java, three streams are created for us automatically.

- o System.out: standard output stream
- o System.in: standard input stream
- o System.err: standard error stream

191) What is the difference between the Reader/Writer class hierarchy and the InputStream/OutputStream class hierarchy?

The Reader/Writer class hierarchy is character-oriented, and the InputStream/OutputStream class hierarchy is byte-oriented. The ByteStream classes are used to perform input-output of 8-bit bytes whereas the CharacterStream classes are used to perform the input/output for the 16-bit Unicode system. There are many classes in the ByteStream class hierarchy, but the most frequently used classes are FileInputStream and FileOutputStream. The most frequently used classes CharacterStream class hierarchy is FileReader and FileWriter.

192) What are the super most classes for all the streams?

All the stream classes can be divided into two types of classes that are ByteStream classes and CharacterStream Classes. The ByteStream classes are further divided into InputStream classes and OutputStream classes. CharacterStream classes are also divided into Reader classes and Writer classes. The SuperMost classes for all the InputStream classes is java.io.InputStream and for all the output stream classes is java.io.OutPutStream. Similarly, for all the reader classes, the super-most class is java.io.Reader, and for all the writer classes, it is java.io.Writer.

193) What are the FileInputStream and FileOutputStream?

Java FileOutputStream is an output stream used for writing data to a file. If you have some primitive values to write into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through the FileOutputStream class. However, for character-oriented data, it is preferred to use FileWriter than FileOutputStream. Consider the following example of writing a byte into a file.

```java
1.       import java.io.FileOutputStream;
2.       public class FileOutputStreamExample {
3.          public static void main(String args[]){
4.              try{
5.                FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
6.                fout.write(65);
7.                fout.close();
8.                System.out.println("success...");
9.              }catch(Exception e){System.out.println(e);}
10.         }
11.      }
```

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video, etc. You can also read character-stream data. However, for reading streams of characters, it is recommended to use FileReader class. Consider the following example for reading bytes from a file.

```java
1.       import java.io.FileInputStream;
2.       public class DataStreamExample {
3.          public static void main(String args[]){
4.              try{
5.                FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.                int i=fin.read();
```

```
7.                  System.out.print((char)i);

8.

9.                    fin.close();

10.                }catch(Exception e){System.out.println(e);}

11.              }

12.              }

13.
```

194) What is the purpose of using BufferedInputStream and BufferedOutputStream classes?

Java BufferedOutputStream class is used for buffering an output stream. It internally uses a buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. Whereas, Java BufferedInputStream class is used to read information from the stream. It internally uses the buffer mechanism to make the performance fast.

196) What are FilterStreams?

FilterStream classes are used to add additional functionalities to the other stream classes. FilterStream classes act like an interface which read the data from a stream, filters it, and pass the filtered data to the caller. The FilterStream classes provide extra functionalities like adding line numbers to the destination file, etc.

197) What is an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another. Many Filter classes that allow a user to make a chain using multiple input streams. It generates a combined effect on several filters.

198) In Java, How many ways you can take input from the console?

In Java, there are three ways by using which, we can take input from the console.

- o **Using BufferedReader class:** we can take input from the console by wrapping System.in into an InputStreamReader and passing it into the BufferedReader. It provides an efficient reading as the input gets buffered. Consider the following example.

```
1.      import java.io.BufferedReader;
2.      import java.io.IOException;
3.      import java.io.InputStreamReader;
4.      public class Person
5.      {
6.         public static void main(String[] args) throws IOException
7.         {
8.           System.out.println("Enter the name of the person");
9.            BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
10.           String name = reader.readLine();
11.           System.out.println(name);
```

12.   }
13.   }

o **Using Scanner class:** The Java Scanner class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values. Java Scanner class is widely used to parse text for string and primitive types using a regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces. Consider the following example.

```java
1.      import java.util.*;
2.      public class ScannerClassExample2 {
3.          public static void main(String args[]){
4.              String str = "Hello/This is JavaTpoint/My name is Abhishek.";
5.              //Create scanner with the specified String Object
6.              Scanner scanner = new Scanner(str);
7.              System.out.println("Boolean Result: "+scanner.hasNextBoolean());
8.              //Change the delimiter of this scanner
9.              scanner.useDelimiter("/");
10.             //Printing the tokenized Strings
11.             System.out.println("---Tokenizes String---");
12.         while(scanner.hasNext()){
13.             System.out.println(scanner.next());
14.         }
```

15.   //Display the new delimiter

16.   System.out.println("Delimiter used: " +scanner.delimiter());

17.   scanner.close();

18.   }

19.   }

20.

- **Using Console class:** The Java Console class is used to get input from the console. It provides methods to read texts and passwords. If you read the password using the Console class, it will not be displayed to the user. The java.io.Console class is attached to the system console internally. The Console class is introduced since 1.5. Consider the following example.

```
1.    import java.io.Console;
2.    class ReadStringTest{
3.    public static void main(String args[]){
4.    Console c=System.console();
5.    System.out.println("Enter your name: ");
6.    String n=c.readLine();
7.    System.out.println("Welcome "+n);
8.    }
9.    }
```

Multithreading Interview Questions

1) What is multithreading?

Multithreading is a process of executing multiple threads simultaneously. Multithreading is used to obtain the multitasking. It consumes less memory and gives the fast and efficient performance. Its main advantages are:

- Threads share the same address space.
- The thread is lightweight.
- The cost of communication between the processes is low.

More details.

2) What is the thread?

A thread is a lightweight subprocess. It is a separate path of execution because each thread runs in a different stack frame. A process may contain multiple threads. Threads share the process resources, but still, they execute independently.

More details.

3) Differentiate between process and thread?

There are the following differences between the process and thread.

- A Program in the execution is called the process whereas; A thread is a subset of the process

- Processes are independent whereas threads are the subset of process.
- Process have different address space in memory, while threads contain a shared address space.
- Context switching is faster between the threads as compared to processes.
- Inter-process communication is slower and expensive than inter-thread communication.
- Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.
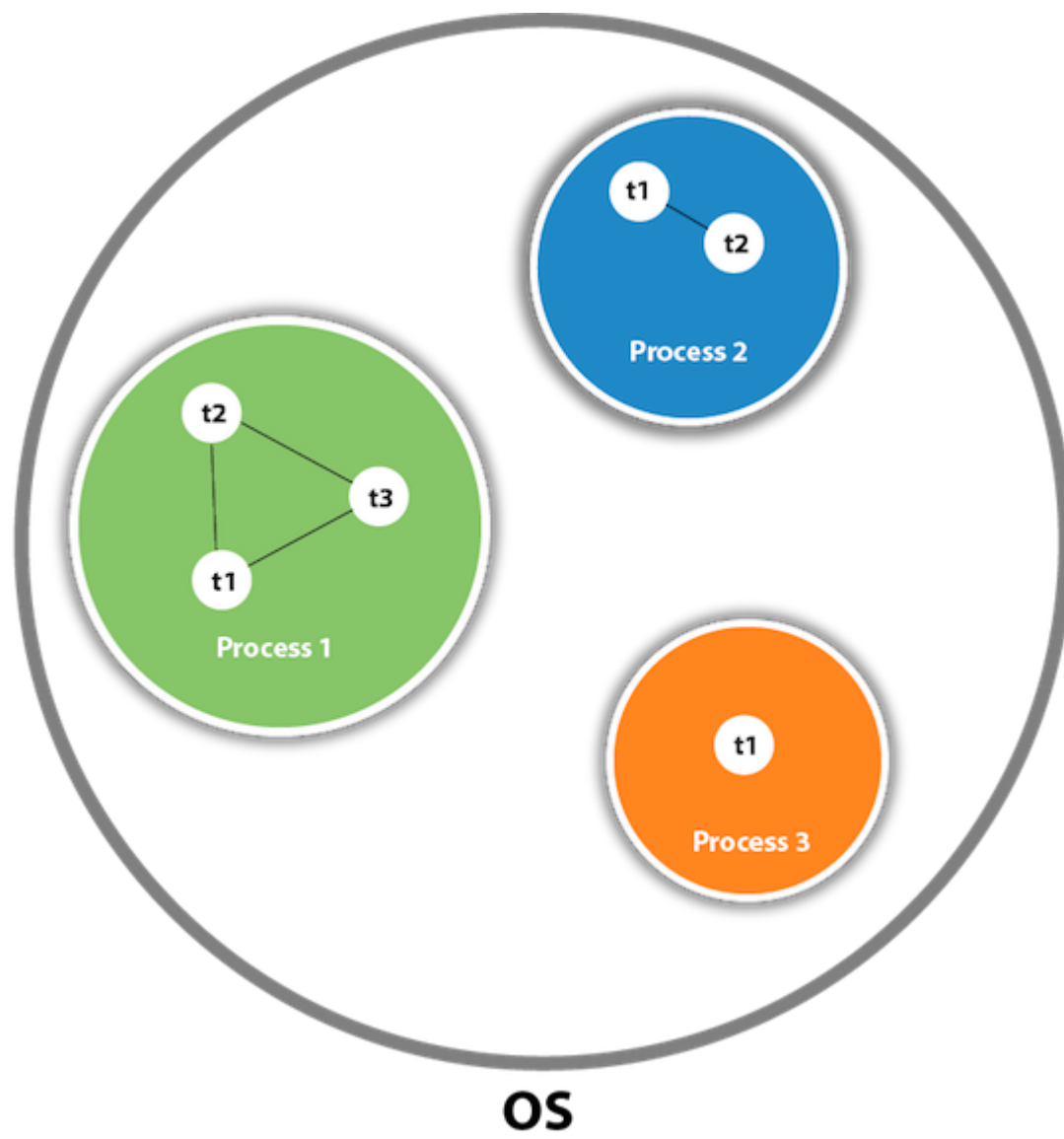
4) What do you understand by inter-thread communication?

- o The process of communication between synchronized threads is termed as inter-thread communication.
- o Inter-thread communication is used to avoid thread polling in Java.
- o The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.
- o It can be obtained by wait(), notify(), and notifyAll() methods.

5) What is the purpose of wait() method in Java?

The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.

public final void wait()

6) Why must wait() method be called from the synchronized block?

We must call the wait method otherwise it will throw **java.lang.IllegalMonitorStateException** exception. Moreover, we need wait() method for inter-thread communication with notify() and notifyAll(). Therefore It must be present in the synchronized block for the proper and correct communication.
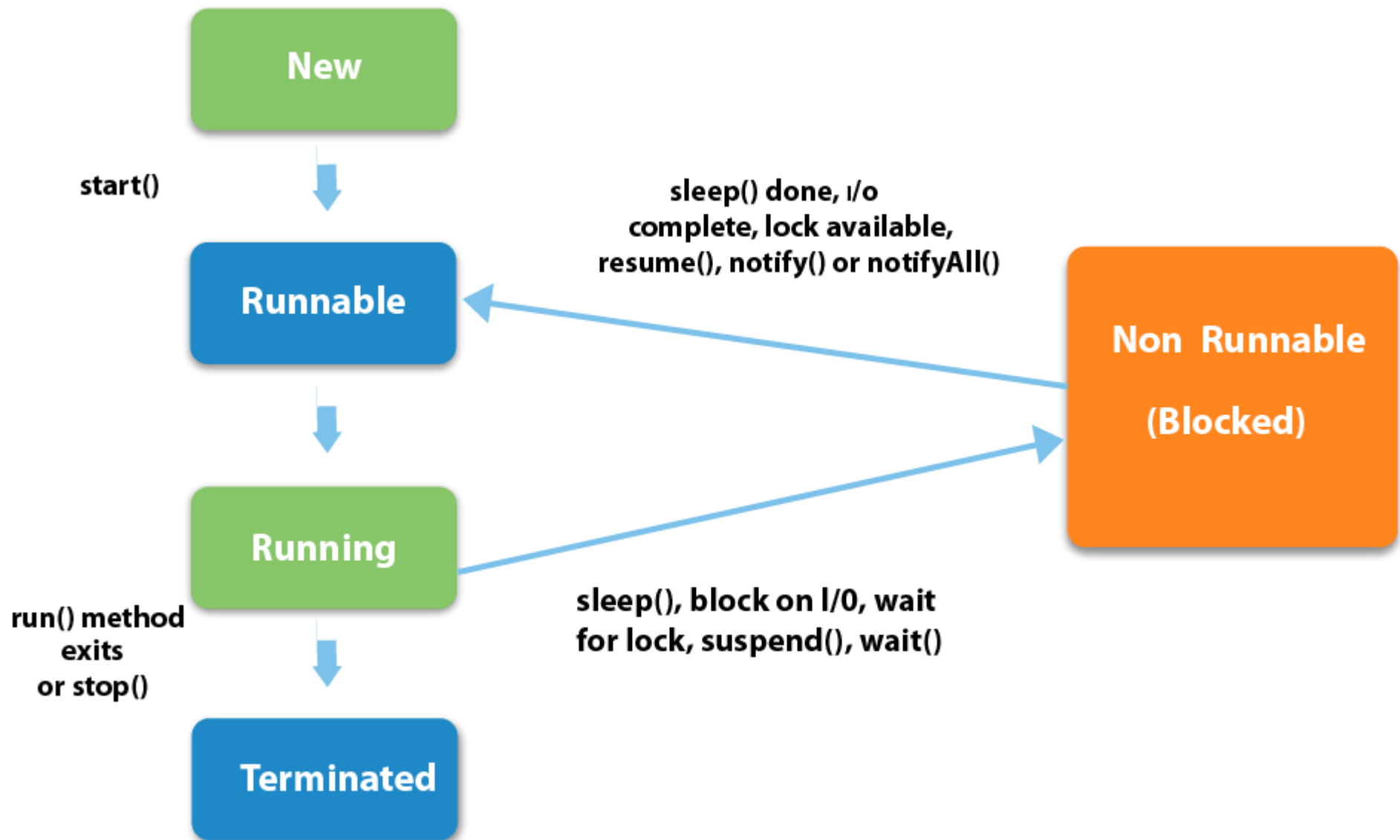
7) What are the advantages of multithreading?

Multithreading programming has the following advantages:

- Multithreading allows an application/program to be always reactive for input, even already running with some background tasks
- Multithreading allows the faster execution of tasks, as threads execute independently.
- Multithreading provides better utilization of cache memory as threads share the common memory resources.
- Multithreading reduces the number of the required server as one server can execute multiple threads at a time.

8) What are the states in the lifecycle of a Thread?

A thread can have one of the following states during its lifetime:

1. **New:** In this state, a Thread class object is created using a new operator, but the thread is not alive. Thread doesn't start until we call the start() method.
2. **Runnable:** In this state, the thread is ready to run after calling the start() method. However, the thread is not yet selected by the thread scheduler.
3. **Running:** In this state, the thread scheduler picks the thread from the ready state, and the thread is running.
4. **Waiting/Blocked:** In this state, a thread is not running but still alive, or it is waiting for the other thread to finish.
5. **Dead/Terminated:** A thread is in terminated or dead state when the run() method exits.

New

start()

Runnable

sleep() done, I/O complete, lock available, resume(), notify() or notifyAll()

Non Runnable (Blocked)

Running

run() method exits or stop()

sleep(), block on I/O, wait for lock, suspend(), wait()

Terminated

9) What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

10) What is context switching?

In Context switching the state of the process (or thread) is stored so that it can be restored and execution can be resumed from the same point later. Context switching enables the multiple processes to share the same CPU.

11) Differentiate between the Thread class and Runnable interface for creating a Thread?

The Thread can be created by using two ways.

- o   By extending the Thread class
- o   By implementing the Thread class

However, the primary differences between both the ways are given below:

- o   By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).

- By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object
- Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

12) What does join() method?

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task. Join method is overloaded in Thread class in the following ways.

- public void join()throws InterruptedException
- public void join(long milliseconds)throws InterruptedException

More details.

13) Describe the purpose and working of sleep() method.

The sleep() method in java is used to block a thread for a particular time, which means it pause the execution of a thread for a specific time. There are two methods of doing so.

Syntax:

- public static void sleep(long milliseconds)throws InterruptedException
- public static void sleep(long milliseconds, int nanos)throws InterruptedException

Working of sleep() method

When we call the sleep() method, it pauses the execution of the current thread for the given time and gives priority to another thread(if available). Moreover, when the waiting time completed then again previous thread changes its state from waiting to runnable and comes in running state, and the whole process works so on till the execution doesn't complete.

14) What is the difference between wait() and sleep() method?

wait()	sleep()
1) The wait() method is defined in Object class.	The sleep() method is defined in Thread class.
2) The wait() method releases the lock.	The sleep() method doesn't release the lock.

15) Is it possible to start a thread twice?

No, we cannot restart the thread, as once a thread started and executed, it goes to the Dead state. Therefore, if we try to start a thread twice, it will give a runtimeException "java.lang.IllegalThreadStateException". Consider the following example.

1.   **public class** Multithread1 **extends** Thread
2.   {
3.   **public void** run()

```
4.              {
5.                try {
6.                    System.out.println("thread is executing now........");
7.                } catch(Exception e) {
8.                }
9.              }
10.             public static void main (String[] args) {
11.                 Multithread1 m1= new Multithread1();
12.                 m1.start();
13.                 m1.start();
14.             }
15.         }
```

Output

```
thread is executing now........
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Thread.java:708)
        at Multithread1.main(Multithread1.java:13)
```
More details.

16) Can we call the run() method instead of start()?

Yes, calling run() method directly is valid, but it will not work as a thread instead it will work as a normal object. There will not be context-switching between the threads. When we call the start() method, it internally calls the run() method, which creates a new stack for a thread while directly calling the run() will not create a new stack.

More details.

17) What about the daemon threads?

The daemon threads are the low priority threads that provide the background support and services to the user threads. Daemon thread gets automatically terminated by the JVM if the program remains with the daemon thread only, and all other user threads are ended/died. There are two methods for daemon thread available in the Thread class:

- **public void setDaemon(boolean status):** It used to mark the thread daemon thread or a user thread.
- **public boolean isDaemon():** It checks the thread is daemon or not.

More details.

18) Can we make the user thread as daemon thread if the thread is started?

No, if you do so, it will throw IllegalThreadStateException. Therefore, we can only create a daemon thread before starting the thread.

1.   **class** Testdaemon1 **extends** Thread{
2.   **public void** run(){
3.   System.out.println("Running thread is daemon...");

```
4.          }
5.          public static void main (String[] args) {
6.              Testdaemon1 td= new Testdaemon1();
7.              td.start();
8.              setDaemon(true);// It will throw the exception: td.
9.            }
10.           }
```

Output

```
Running thread is daemon...

Exception in thread "main" java.lang.IllegalThreadStateException

at java.lang.Thread.setDaemon(Thread.java:1359)

at Testdaemon1.main(Testdaemon1.java:8)
```

20)When should we interrupt a thread?

We should interrupt a thread when we want to break out the sleep or wait state of a thread. We can interrupt a thread by calling the interrupt() throwing the InterruptedException.

More details.

21) What is the synchronization?

Synchronization is the capability to control the access of multiple threads to any shared resource. It is used:

1. To prevent thread interference.
2. To prevent consistency problem.

When the multiple threads try to do the same task, there is a possibility of an erroneous result, hence to remove this issue, Java uses the process of synchronization which allows only one thread to be executed at a time. Synchronization can be achieved in three ways:

- o by the synchronized method
- o by synchronized block
- o by static synchronization

Syntax for synchronized block

1.   **synchronized**(object reference expression)
2.   {
3.   //code block
4.   }
5.
More details.

22) What is the purpose of the Synchronized block?

The Synchronized block can be used to perform synchronization on any specific resource of the method. Only one thread at a time can execute on a particular resource, and all other threads which attempt to enter the synchronized block are blocked.

- o Synchronized block is used to lock an object for any shared resource.
- o The scope of the synchronized block is limited to the block on which, it is applied. Its scope is smaller than a method.

More details.

23)Can Java object be locked down for exclusive use by a given thread?

Yes. You can lock an object by putting it in a "synchronized" block. The locked object is inaccessible to any thread other than the one that explicitly claimed it.

24) What is static synchronization?

If you make any static method as synchronized, the lock will be on the class not on the object. If we use the synchronized keyword before a method so it will lock the object (one thread can access an object at a time) but if we use static synchronized so it will lock a class (one thread can access a class at a time). More details.

25)What is the difference between notify() and notifyAll()?

The notify() is used to unblock one waiting thread whereas notifyAll() method is used to unblock all the threads in waiting state.

26)What is the deadlock?

Deadlock is a situation in which every thread is waiting for a resource which is held by some other waiting thread. In this situation, Neither of the thread executes nor it gets the chance to be executed. Instead, there exists a universal waiting state among all the threads. Deadlock is a very complicated situation which can break our code at runtime.

More details.

## 27) How to detect a deadlock condition? How can it be avoided?

We can detect the deadlock condition by running the code on cmd and collecting the Thread Dump, and if any deadlock is present in the code, then a message will appear on cmd.

Ways to avoid the deadlock condition in Java:

o **Avoid Nested lock:** Nested lock is the common reason for deadlock as deadlock occurs when we provide locks to various threads so we should give one lock to only one thread at some particular time.

o **Avoid unnecessary locks:** we must avoid the locks which are not required.

o **Using thread join:** Thread join helps to wait for a thread until another thread doesn't finish its execution so we can avoid deadlock by maximum use of join method.

1) What is JDBC?

JDBC is a Java API that is used to connect and execute the query to the database. JDBC API uses JDBC drivers to connect to the database. JDBC API can be used to access tabular data stored into any relational database.

More details.
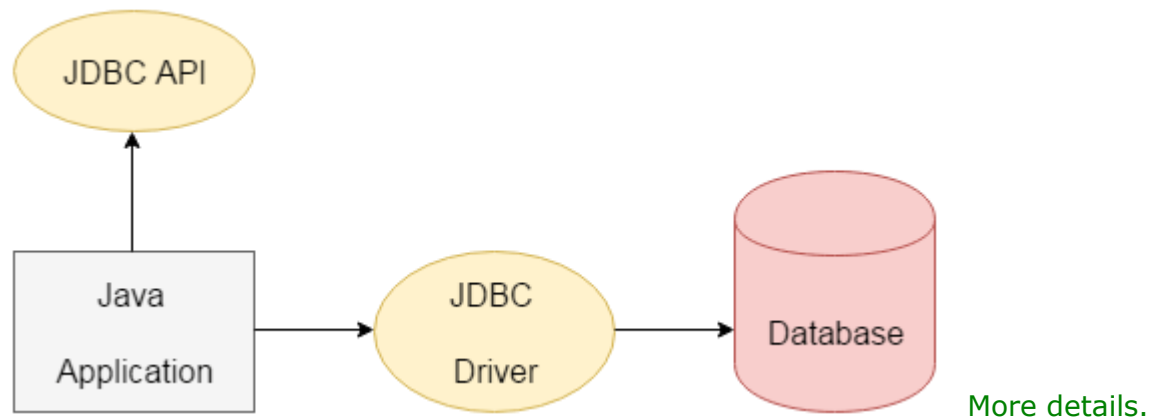
2) What is JDBC Driver?

JDBC Driver is a software component that enables Java application to interact with the database. There are 4 types of JDBC drivers:

1. **JDBC-ODBC bridge driver:** The JDBC-ODBC bridge driver uses the ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of the thin driver. It is easy to use and can be easily connected to any database.
2. **Native-API driver (partially java 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. Its performance is better than JDBC-ODBC bridge driver. However, the native driver must be installed on each client machine.
3. **Network Protocol driver (fully java driver):** The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is entirely written in Java. There is no requirement of the client-side library because of the application server that can perform many tasks like auditing, load balancing, logging, etc.

4. **Thin driver (fully java driver):** The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as the thin driver. It is entirely written in Java language. Its performance is better than all other drivers however these drivers depend upon the database.

More details.

3) What are the steps to connect to the database in java?

The following steps are used in database connectivity.

- **Registering the driver class:**

  The forName() method of the Class class is used to register the driver class. This method is used to load the driver class dynamically. Consider the following example to register OracleDriver class.

  1.   Class.forName("oracle.jdbc.driver.OracleDriver");

- **Creating connection:**

  The getConnection() method of DriverManager class is used to establish the connection with the database. The syntax of the getConnection() method is given below.

  1.   1) **public static** Connection getConnection(String url)**throws** SQLException
  2.   2) **public static** Connection getConnection(String url,String name,String password)

3.   **throws** SQLException

Consider the following example to establish the connection with the Oracle database.

1.   Connection con=DriverManager.getConnection(
2.   "jdbc:oracle:thin:@localhost:1521:xe","system","password");

o **Creating the statement:**

The createStatement() method of Connection interface is used to create the Statement. The object of the Statement is responsible for executing queries with the database.

1.   **public** Statement createStatement()**throws** SQLException

consider the following example to create the statement object

1.   Statement stmt=con.createStatement();

o **Executing the queries:**

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method is given below.

1.   **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute the query

1.   ResultSet rs=stmt.executeQuery("select * from emp");
2.   **while**(rs.next()){
3.   System.out.println(rs.getInt(1)+" "+rs.getString(2));
4.   }

However, to perform the insert and update operations in the database, executeUpdate() method is used which returns the boolean value to indicate the successful completion of the operation.

o **Closing connection:**

By closing connection, object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method is given below.

1.   **public void** close()**throws** SQLException

Consider the following example to close the connection.

1.   con.close();

More details.

4) What are the JDBC API components?

The java.sql package contains following interfaces and classes for JDBC API.

Interfaces:

- **Connection:** The Connection object is created by using getConnection() method of DriverManager class. DriverManager is the factory for connection.

- **Statement:** The Statement object is created by using createStatement() method of Connection class. The Connection interface is the factory for Statement.

- **PreparedStatement:** The PrepareStatement object is created by using prepareStatement() method of Connection class. It is used to execute the parameterized query.

- **ResultSet:** The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points before the first row. The executeQuery() method of Statement interface returns the ResultSet object.

- **ResultSetMetaData:** The object of ResultSetMetaData interface cotains the information about the data (table) such as numer of columns, column name, column type, etc. The getMetaData() method of ResultSet returns the object of ResultSetMetaData.

- **DatabaseMetaData:** DatabaseMetaData interface provides methods to get metadata of a database such as the database product name, database product version, driver name, name of the total number of tables, the name of the total number of views, etc. The getMetaData() method of Connection interface returns the object of DatabaseMetaData.

- **CallableStatement:** CallableStatement interface is used to call the stored procedures and functions. We can have business logic on the database through the use of stored procedures and functions that will make the performance better because these are precompiled. The prepareCall() method of Connection interface returns the instance of CallableStatement.

Classes:

- **DriverManager:** The DriverManager class acts as an interface between the user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. It contains several methods to keep the interaction between the user and drivers.

- **Blob:** Blob stands for the binary large object. It represents a collection of binary data stored as a single entity in the database management system.

- **Clob:** Clob stands for Character large object. It is a data type that is used by various database management systems to store character files. It is similar to Blob except for the difference that BLOB represent binary data such as images, audio and video files, etc. whereas Clob represents character stream data such as character files, etc.

- **SQLException** It is an Exception class which provides information on database access errors.

5) What are the JDBC statements?

In JDBC, Statements are used to send SQL commands to the database and receive data from the database. There are various methods provided by JDBC statements such as execute(), executeUpdate(), executeQuery, etc. which helps you to interact with the database.

There is three type of JDBC statements given in the following table.

Statements	Explanation
Statement	Statement is the factory for resultset. It is used for general purpose access to the database. It executes a static SQL query at runtime.
PreparedStatement	The PreparedStatement is used when we need to provide input parameters to the query at runtime.
CallableStatement	CallableStatement is used when we need to access the database stored procedures. It can also accept runtime parameters.

6) What is the return type of Class.forName() method?

The Class.forName() method returns the object of java.lang.Class object.

7) What are the differences between Statement and PreparedStatement interface?

Statement	PreparedStatement
The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet; i.e., it provides the factory method to get the object of ResultSet.	The PreparedStatement interface is a subinterface of Statement. It is used to execute the parameterized query.
In the case of Statement, the query is compiled each time we run the program.	In the case of PreparedStatement, the query is compiled only once.
The Statement is mainly used in the case when we need to run the static query at runtime.	PreparedStatement is used when we need to provide input parameters to the query at runtime.

More details.

8) How can we set null value in JDBC PreparedStatement?

By using setNull() method of PreparedStatement interface, we can set the null value to an index. The syntax of the method is given below.

1.   **void** setNull(**int** parameterIndex, **int** sqlType) **throws** SQLException
2.

9) What are the benefits of PreparedStatement over Statement?

The benefits of using PreparedStatement over Statement interface is given below.

- o The PreparedStatement performs faster as compare to Statement because the Statement needs to be compiled everytime we run the code whereas the PreparedStatement compiled once and then execute only on runtime.
- o PreparedStatement can execute Parameterized query whereas Statement can only run static queries.
- o The query used in PreparedStatement is appeared to be similar every time. Therefore, the database can reuse the previous access plan whereas, Statement inline the parameters into the String, therefore, the query doesn't appear to be same everytime which prevents cache reusage.

10) What are the differences between execute, executeQuery, and executeUpdate?

execute	executeQuery	executeUpdate
The execute method can be used for any SQL statements(Select and Update both).	The executeQuery method can be used only with the	The executeUpdate method can be used to update/delete/insert

	select statement.	operations in the database.
The execute method returns a boolean type value where true indicates that the ResultSet s returned which can later be extracted and false indicates that the integer or void value is returned.	The executeQuery() method returns a ResultSet object which contains the data retrieved by the select statement.	The executeUpdate() method returns an integer value representing the number of records affected where 0 indicates that query returns nothing.

11) What are the different types of ResultSet?

ResultSet is categorized by the direction of the reading head and sensitivity or insensitivity of the result provided by it. There are three general types of ResultSet.

Type	Description
ResultSet.TYPE_Forward_ONLY	The cursor can move in the forward direction only.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can move in both the direction (forward and backward). The ResultSet is not sensitive to the changes made by the others to the database.
ResultSet.TYPE_SCROLL_SENSITIVE	The cursor can move in both the direction. The ResultSet is sensitive to the changes made by the others to the database.

12) What are the differences between ResultSet and RowSet?

ResultSet	RowSet
ResultSet cannot be serialized as it maintains the connection with the database.	RowSet is disconnected from the database and can be serialized.
ResultSet object is not a JavaBean object	ResultSet Object is a JavaBean object.
ResultSet is returned by the executeQuery() method of Statement Interface.	Rowset Interface extends ResultSet Interface and returned by calling the RowSetProvider.newFactory().createJdbcRowSet() method.
ResultSet object is non-scrollable and non-updatable by default.	RowSet object is scrollable and updatable by default.

13) How can we execute stored procedures using CallableStatement?

Following are the steps to create and execute stored procedures. Here, we are creating a table user420 by using a stored procedure and inserting values into it.

- o **Create the procedure in the database.**

To call the stored procedure, you need to create it in the database. Here, we are assuming that the stored procedure looks like this.

1.   create or replace procedure "INSERTR"
2.   (id IN NUMBER,
3.   name IN VARCHAR2)
4.   is
5.   begin
6.   insert into user420 values(id,name);
7.   end;
8.   /

The table structure is given below:

1.   create table user420(id number(10), name varchar2(200));

o   **Establish a network connection.**

1.   Class.forName("oracle.jdbc.driver.OracleDriver");
2.   Connection con=DriverManager.getConnection(

3.   "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

- ○ **Create the Object of CallableStatement.**

1.   CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");

- ○ **Provide the values and execute the query by using the following syntax.**

1.   stmt.setInt(1,1011);
2.   stmt.setString(2,"Amit");
3.   stmt.execute();

- ○ **Check the database; the values will be found there. However, the complete code will look like the following.**

1.   **import** java.sql.*;
2.   **public class** Proc {
3.   **public static void** main(String[] args) **throws** Exception{
4.
5.   Class.forName("oracle.jdbc.driver.OracleDriver");
6.   Connection con=DriverManager.getConnection(

```
7.          "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

8.

9.          CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");

10.         stmt.setInt(1,1011);

11.         stmt.setString(2,"Amit");

12.         stmt.execute();

13.

14.         System.out.println("success");

15.         }

16.         }
```

14) What is the role of the JDBC DriverManager class?

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

More details.

15) What are the functions of the JDBC Connection interface?

The **Connection interface** maintains a session with the database. It can be used for transaction management. It provides factory methods that return the instance of Statement, PreparedStatement, CallableStatement, and DatabaseMetaData.

16) What does the JDBC ResultSet interface?

The ResultSet object represents a row of a table. It can be used to change the cursor pointer and get the information from the database. By default, ResultSet object can move in the forward direction only and is not updatable. However, we can make this object to move the forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int, int) method.

17) What does the JDBC ResultSetMetaData interface?

The ResultSetMetaData interface returns the information of table such as the total number of columns, column name, column type, etc.

18) What does the JDBC DatabaseMetaData interface?

The DatabaseMetaData interface returns the information of the database such as username, driver name, driver version, number of tables, number of views, etc. Consider the following example.

```
1.       import java.sql.*;
```

```java
class Dbmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
DatabaseMetaData dbmd=con.getMetaData();

System.out.println("Driver Name: "+dbmd.getDriverName());
System.out.println("Driver Version: "+dbmd.getDriverVersion());
System.out.println("UserName: "+dbmd.getUserName());
System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Output

```
Driver Name: Oracle JDBC Driver
```

Driver Version: 10.2.0.1.0XE

Database Product Name: Oracle

Database Product Version: Oracle Database 10g Express Edition Release 10.2.0.1.0 -Production
More details.

19) Which interface is responsible for transaction management in JDBC?

The **Connection interface** provides methods for transaction management such as commit(), rollback() etc.

More details.