

Core Java Interview Questions – Set 1

Reference - <https://www.javatpoint.com/corejava-interview-questions>
<https://www.javatpoint.com/corejava-interview-questions>

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	Java has built-in thread support.

	third-party libraries for thread support.	
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (<code>/** ... */</code>) 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.
- **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.

- Java is the software-based platform whereas other platforms may be the hardware platforms or software-based platforms.
- 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.

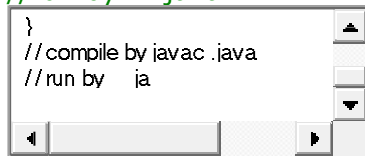
1. **Bootstrap ClassLoader**: This is the first classloader which is the superclass of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like `java.lang` package classes, `java.net` package classes, `java.util` package classes, `java.io` package 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.

- **Public** The classes, methods, or variables which are defined as public, can be accessed by any class or method.
 - **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.
 - **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.

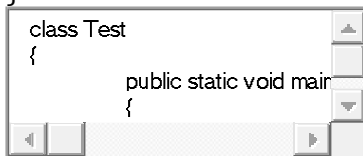
- Packages avoid the name clashes.
 - The Package provides easier access control.
 - We can also have the hidden classes that are not visible outside and used by the package.
 - It is easier to locate the related classes.
-

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**.

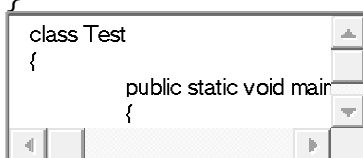
In the second case, the string Javatpoint is concatenated with 10 to be the string **Javatpoint10** which will then be concatenated with 20 to be **Javatpoint1020**.

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
```

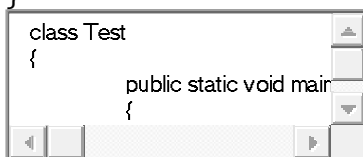
Explanation

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.

- Follows the bottom-up approach in program design.
 - Focus on data with methods to operate upon the object's data
 - Includes the concept like Encapsulation and abstraction which hides the complexities from the user and show only functionality.
 - Implements the real-time approach like inheritance, abstraction, etc.
 - 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.

- 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.
- Object-oriented languages do not have the inbuilt objects whereas Object-based languages have the inbuilt objects, for example, JavaScript has window object.
- 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.

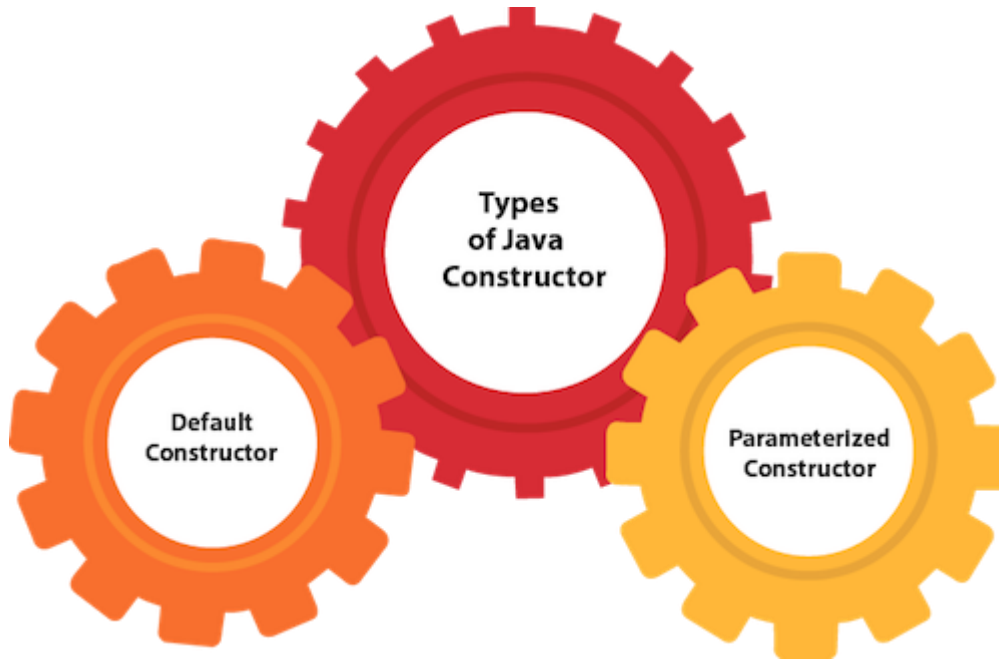
[More Details.](#)

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.

- **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.

- **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. }
```

```
class Student3{  
    int id;  
    String name;  
}
```

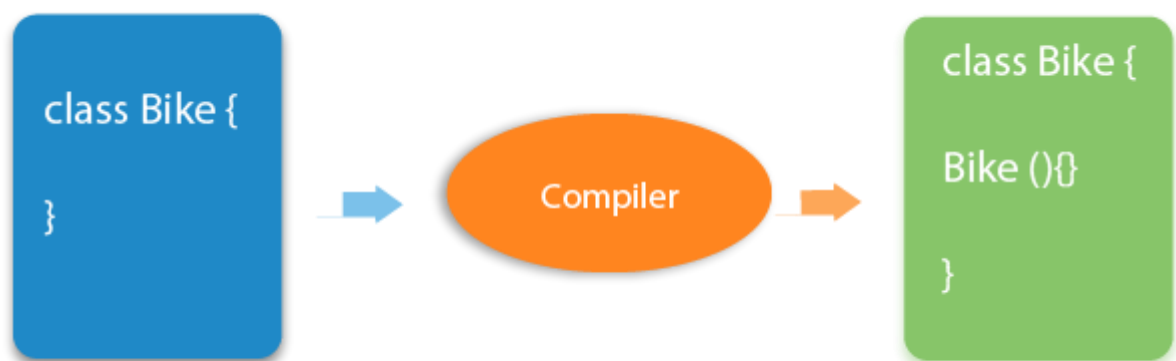
Test it Now

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.



[More Details.](#)

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). [More Details.](#)

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.
```

```

class Test
{
    int i;
    public Test(int k)

```

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:

- By constructor
- By assigning the values of one object into another
- 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. }

```



Test it Now

Output:

```

111 Karan
111 Karan

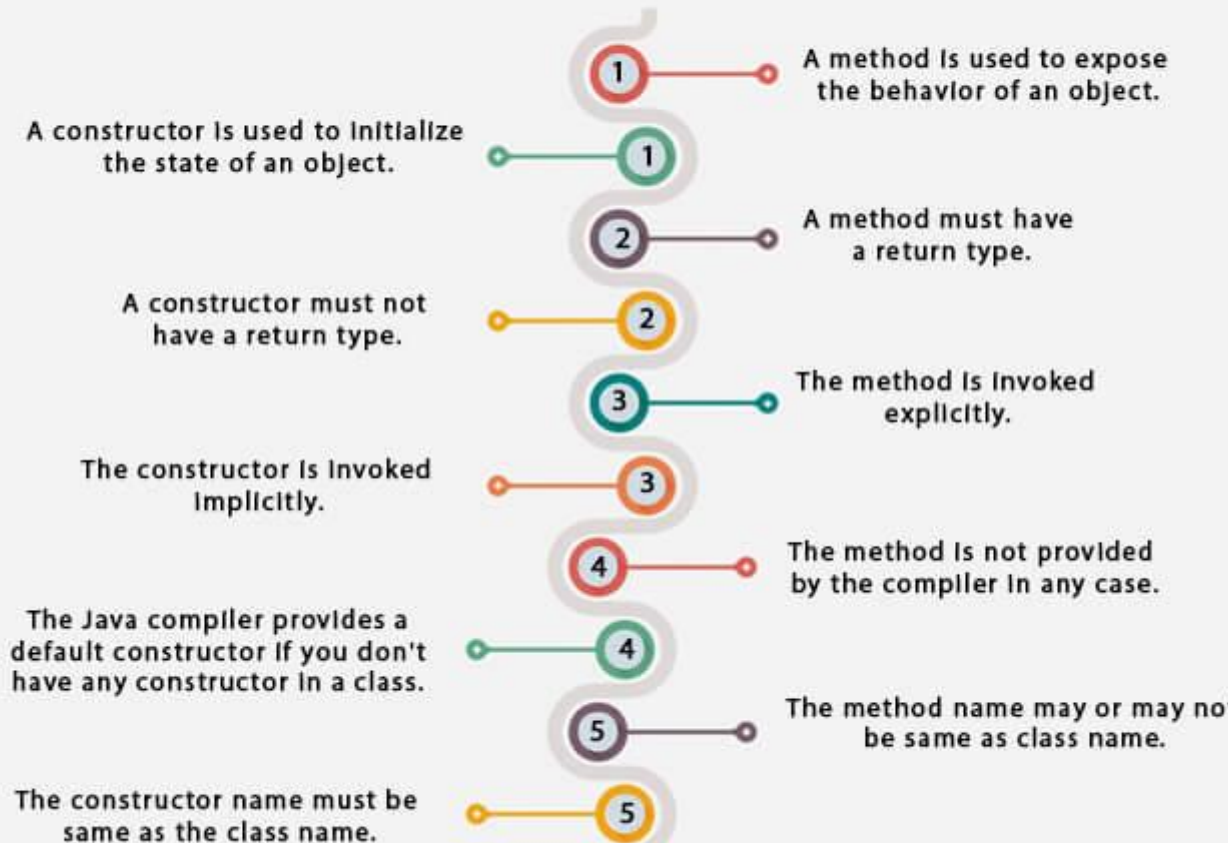
```

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.

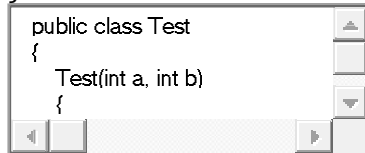
Difference between constructor and method in Java



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

```
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. }
```



```
public class Test
{
    Test(int a, int b)
    {
    }
}
```

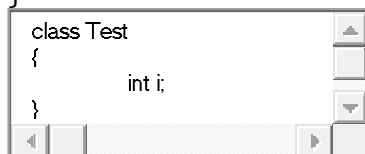
The output of the following program is:

```
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. }
```

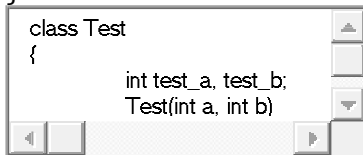


```
class Test
{
    int i;
}
```

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.

Core Java - OOPs Concepts: static keyword Interview Questions

39) What is the static variable?

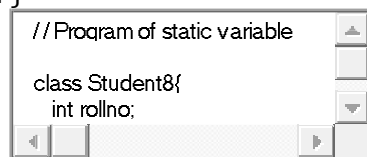
The static variable is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc. Static variable gets memory only once in the class area at the time of class loading. Using a static variable makes your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

```

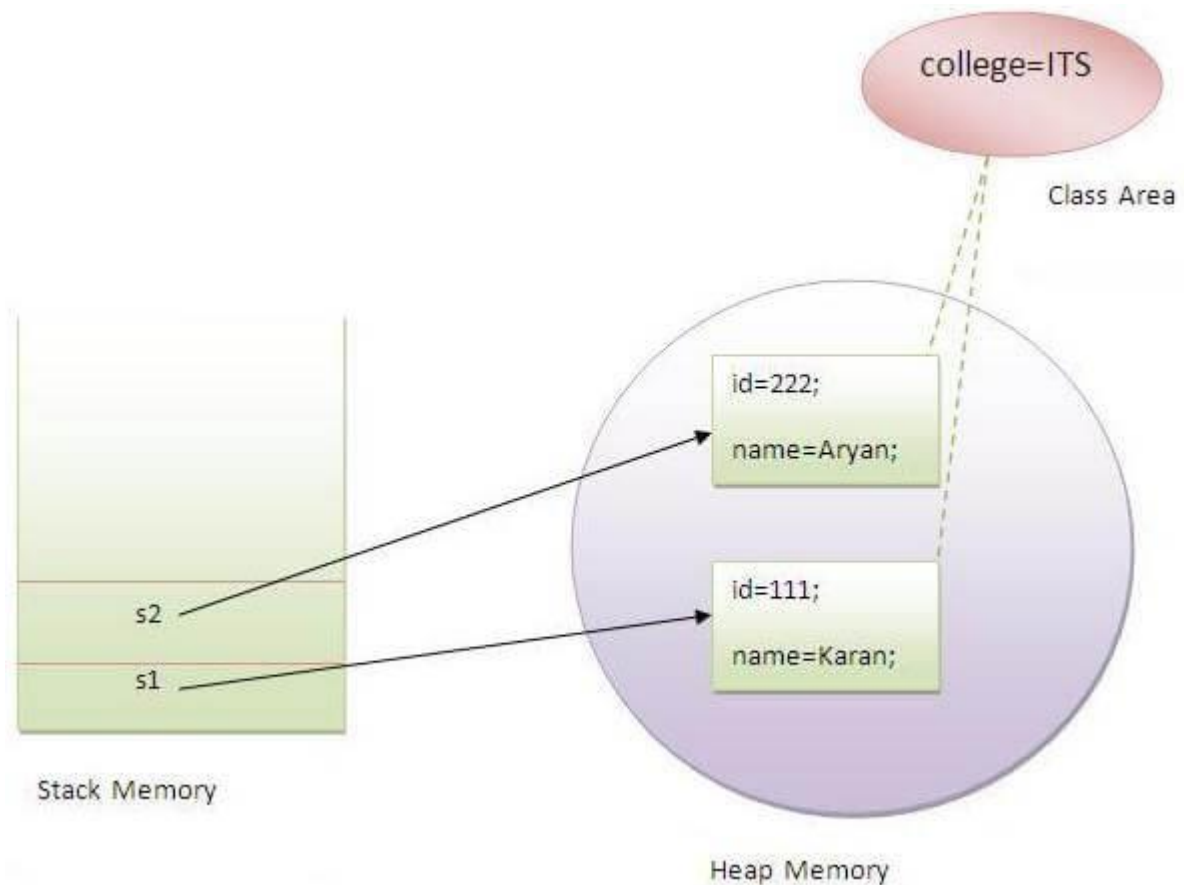
5. String name;
6. static String college = "ITS";
7.
8. Student8(int r,String n){
9.     rollno = r;
10.    name = n;
11. }
12. void display (){System.out.println(rollno+" "+name+" "+college);}
13.
14. public static void main(String args[]){
15. Student8 s1 = new Student8(111,"Karan");
16. Student8 s2 = new Student8(222,"Aryan");
17.
18. s1.display();
19. s2.display();
20. }
21. }

```



Test it Now

Output:111 Karan ITS
222 Aryan ITS



[More Details.](#)

40) What is the static method?

- A static method belongs to the class rather than the object.
- There is no need to create the object to call the static methods.
- 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.

- The static method can not use non-static data member or call the non-static method directly.
 - 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.](#)

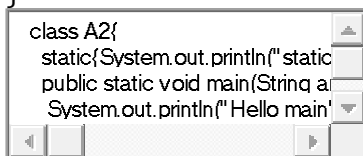
43) Can we override the static methods?

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: <code>public static int cube(int n){ return n*n*n;}</code>	For example: <code>public void msg(){...}.</code>

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.

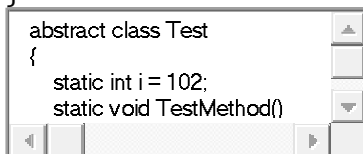
49) Can we make the abstract methods static in Java?

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.

```
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. }
```

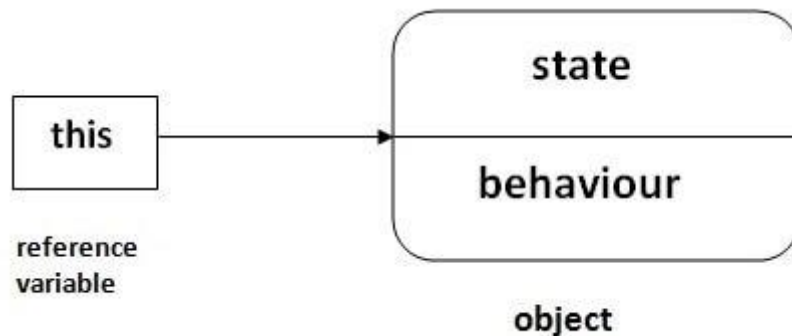


Output

```
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?

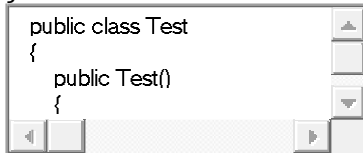
No, this cannot be assigned to any value because it always points to the current class object and this is the final reference in Java. However, if we try to do so, the compiler error will be shown. Consider the following example.

```
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

```

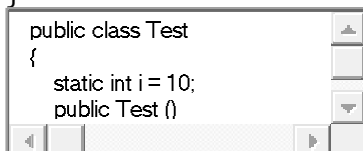
54) Can **this** keyword be used to refer static members?

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.}

```



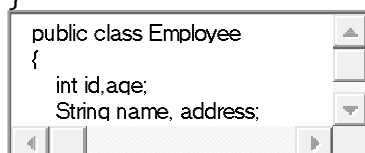
Output

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;
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.

- 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.
 - 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.

- Single-level inheritance
- Multi-level inheritance
- Multiple Inheritance
- Hierarchical Inheritance
- 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.

- 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.
- Runtime polymorphism cannot be achieved without using inheritance.
- We can simulate the inheritance of classes with the real-time objects which makes OOPs more realistic.
- Inheritance provides data hiding. The base class can hide some data from the derived class by making it private.
- 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.

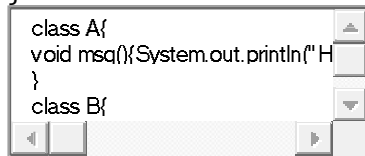
60) Why is multiple inheritance not supported in java?

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. `}`



```
class A{
void msg(){System.out.println("H
"}
class B{
```

Test it Now

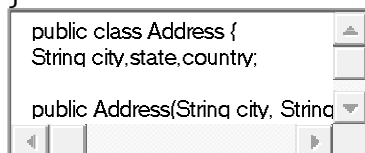
Compile Time Error

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. }



```
public class Address {
String city,state,country;

public Address(String city, String
```

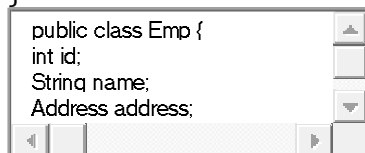
Employee.java

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

```

class Animal{
    Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
    Dog(){System.out.println("dog is created");}
}

```

Test it Now

Output:

```

animal is created
dog is created

```

[More Details.](#)

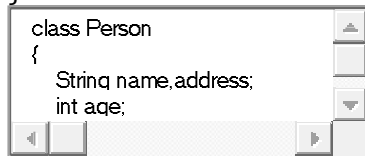
66) How can constructor chaining be done by using the super keyword?

```

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;
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. }
```



```
class Person  
{  
    String name,address;  
    int age;  
}
```

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.

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

- The super keyword always points to the parent class contexts whereas this keyword always points to the current class context.
 - 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.
 - The super and this must be the first statement inside constructor otherwise the compiler will throw an error.
-

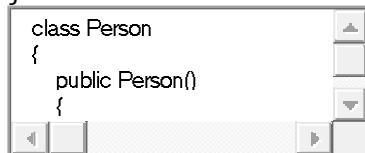
69) What is the output of the following Java program?

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:

```

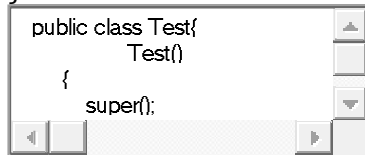
1. public class Test{
2.     Test()
3.     {
4.         super();
5.         this();

```

```

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

```



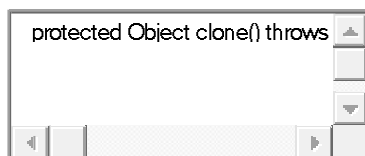
Output:

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.

- Changing the number of arguments

- Changing the return type

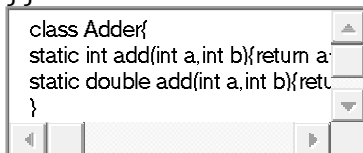
Method overloading increases the readability of the program. Method overloading is performed to figure out the program quickly.

[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?

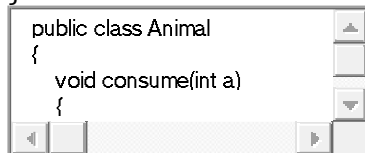
No, We cannot overload the methods by just applying the static keyword to them(number of parameters and types are the same). Consider the following example.

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

```

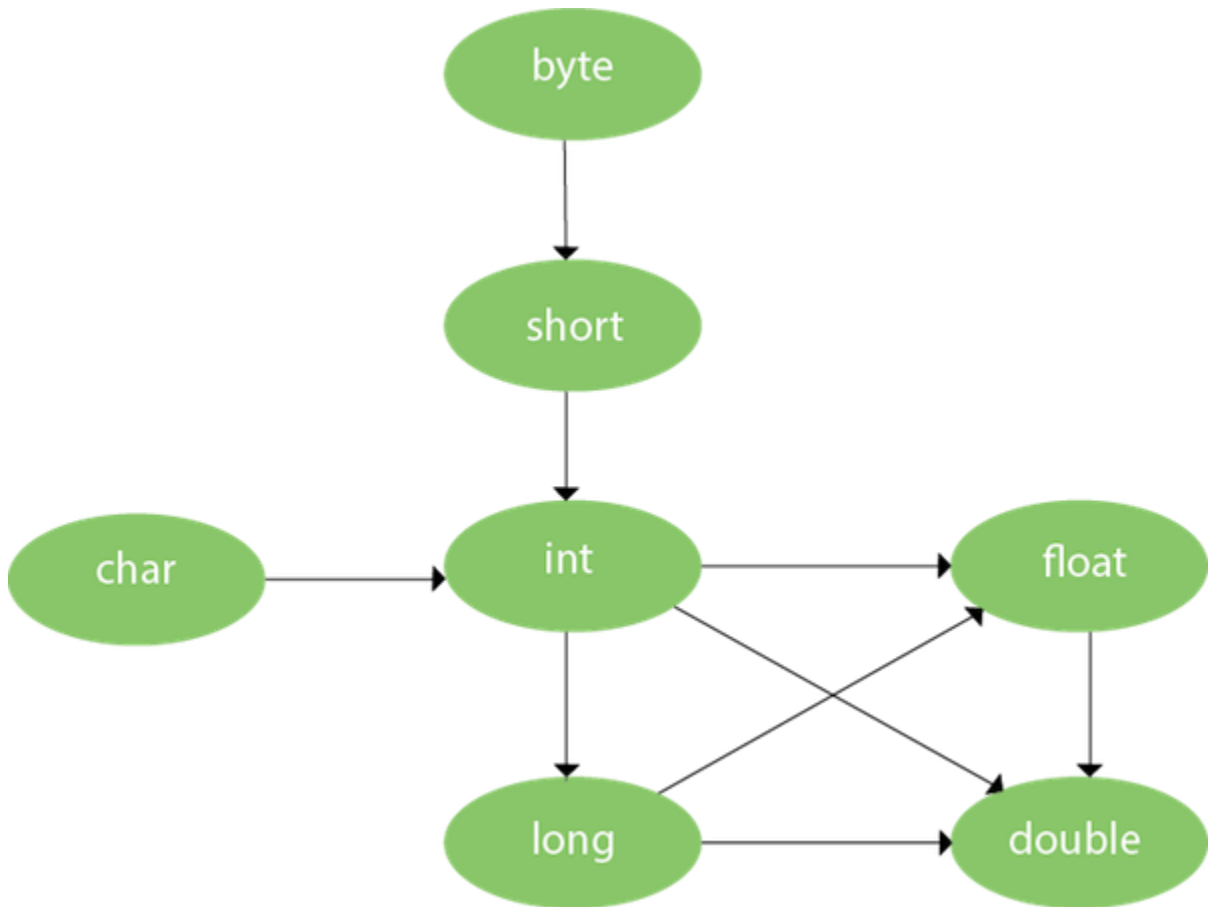
75) Can we overload the main() method?

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. }
```

```
class OverloadingCalculation1{
    void sum(int a,long b){System.out.println("a method invoked");}
    void sum(int a,int b,int c){System.out.println("b method invoked");}
}
```

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. }

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

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.

Core Java - OOPs Concepts: Method Overriding Interview Questions

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

- The method must have the same name as in the parent class.
- The method must have the same signature as in the parent class.
- 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.

- The private can be changed to protected, public, or default.
 - The protected can be changed to public or default.
 - The default can be changed to public.
 - 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.

- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception, but it can declare the unchecked exception.
- 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.}
```

```

class Base
{
    void method(int a)
    {
    }
}

```

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. }

```

class A{
A get(){return this;}
}

```

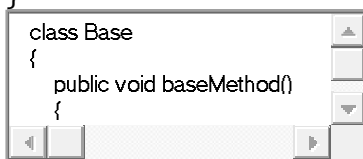
Test it Now

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.



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**

```
class Bike9{
    final int speedlimit=90;//final va
    void run(){
        speedlimit=400;
    }
}
```

Test it Now

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. }

```
class Bike{
    final void run(){System.out.print
}
}
```

Test it Now

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. }

```
final class Bike{}  
  
class Honda1 extends Bike{  
    void run(){System.out.println('');  
}
```

Test it Now

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. }

```
class Student{  
    int id;  
    String name;  
    final String PAN_CARD_NUMBER;  
}
```

[More Details.](#)

94) Can we initialize the final blank variable?

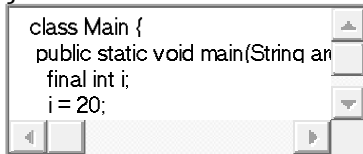
Yes, if it is not static, we can initialize it in the constructor. If it is static blank final variable, it can be initialized only in the static block. [More Details.](#)

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.

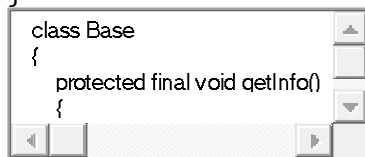
97) What is the output of the following Java program?

1. **class** Base
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 getInfo() 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.