1. What is constructor chaining?

Constructor Chaining is a technique of calling another constructor from one constructor. this() is used to call same class constructor where as super() is used to call super class constructor.

class SuperClass

{

public SuperClass(int i)

{

System.out.println("Super Class Constructor");

}

}

class SubClass extends SuperClass

{

public SubClass()

{

this(10); //Calling same class constructor

}

public SubClass(int i)

{

super(i); //Calling super class constructor

}

}

2. Can we call sub class constructor from super class constructor?

No. There is no way in java to call sub class constructor from a super class constructor.

3. What happens if you keep return type for a constructor?

It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.

class MyClass

{

int MyClass()

{

return 0; //No Compile time error but just a warning

}

}

4. What is No-arg constructor?

Constructor without arguments is called no-arg constructor. Default constructor in java is always a no-arg constructor.

class MyClass

{

public MyClass()

{

//No-arg constructor

}

}

5. What is the use of private constructor?

Private constructors are used to restrict the instantiation of a class. When a class needs to prevent other classes from creating it’s objects then private constructors are suitable for that. Objects to the class which has only private constructors can be created within the class. A very good use of private constructor is in singleton pattern. This ensures only one instance of a class exist at any point of time. Here is an example of singleton pattern using private constructor.

class MyClass

{

private static MyClass object = null;

private MyClass()

{

//private constructor

}

public MyClass getObject()

{

if(object == null)

{

object = new MyClass(); //Creating object using private constructor

}

return object;

}

}

6. Can we use this() and super() in a method?

No, We can’t use this() and super() in a method.

class SuperClass

{

public SuperClass()

{

System.out.println("Super Class Constructor");

}

}

class SubClass extends SuperClass

{

public SubClass()

{

System.out.println("Sub Class Constructor");

}

void method()

{

this(); //Compile time error

super(); //Compile time error

}

}

7. What is a Constructor?

Constructors are used to initialize the object’s state. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.

8. Do we have Copy Constructor in Java?

Like C++, Java also supports copy constructor. But, unlike C++, Java doesn’t create a default copy constructor if you don’t write your own.

To copy the values of one object into another in java, you can use:

Constructor

Assigning the values of one object into another

clone() method of Object class

**9. Can we call sub class constructor from super class constructor?**

No. There is no way in java to call sub class constructor from a super class constructor.

**10. What happens if you keep a return type for a constructor?**

Ideally, Constructor must not have a return type. By definition, if a method has a return type, it’s not a constructor.(JLS8.8 Declaration) It will be treated as a normal method. But compiler gives a warning saying that method has a constructor name.Example:

class GfG

{

int GfG()

{

return 0; // Warning for the return type

}

}

**11. How a no – argument constructor is different from default Constructor?**

If a class contains no constructor declarations, then a default constructor with no formal parameters and no throws clause is implicitly declared.

**12. Do we have destructors in Java?**

No, Because Java is a garbage collected language you cannot predict when (or even if) an object will be destroyed. Hence there is no direct equivalent of a destructor.

If the class being declared is the primordial class Object, then the default constructor has an empty body. Otherwise, the default constructor simply invokes the superclass constructor with no arguments.

A constructor is an integral part of a Java program. It is one of the important topics of core Java. So, in every Java-based interview, there is a possibility that the interviewer may ask few questions from the Java constructor.

In this article, we are going to discuss some commonly asked interview questions on constructors.

1) Define Constructor?

Java constructor is a unique method that initializes the objects, which is called when an instance of the class is created. The memory for the object is allocated when we call the constructor.

Basically, a constructor is a block of code. When we create an object of the class using the new() keyword, at least one constructor is called, and it initializes the objects and allocates memory to them.

If we do not specify any constructor, it will call the default constructor of the class. However, it is not necessary to specify an explicit constructor because the Java compiler provides a default constructor for every Java class.

2) How many types of Constructors are in Java?

There are two types of constructors in Java:

1. **Default Constructor (Non-parameterized Constructor)**
2. **Parameterized Constructor**

The syntax for the default constructor is as follows:

1. <**class** name>() {}

**Example:**

1. Employee()
2. {
3. //some code
4. }

The syntax for the parameterized constructor is as follows:

1. <**class** name>(arg1, arg2) {}

**Example:**

1. Employee(**int** i, String n)
2. {
3. id = i;
4. name = n;
5. }

3) Do we have a copy constructor in Java?

Unlike C++, there is no explicit copy constructor in Java. However, we can achieve the functionality of a copy constructor in Java by copying the values from one object to another, just like the copy constructor.

The following are some methods to copy the values from one object to another:

* By constructor
* By assigning the values of one object to another
* By clone() method of Object class

4) Write a Java Program to Copy the values from one object to another Object.

Below Java program copies the values from one object to another object:

**ConstructorDemo.java:**

1. **class** ConstructorDemo{
2. **int** id;
3. String name;
4. //constructor to initialize integer and string
5. ConstructorDemo(**int** i,String n){
6. id = i;
7. name = n;
8. }
9. //constructor to initialize another object
10. ConstructorDemo(ConstructorDemo c){
11. id = c.id;
12. name =c.name;
13. }
14. **void** display(){System.out.println(id+" "+name);}
15. **public** **static** **void** main(String args[]){
16. ConstructorDemo c1 = **new** ConstructorDemo(100,"Joy");
17. ConstructorDemo c2 = **new** ConstructorDemo(c1);
18. c1.display();
19. c2.display();
20. }
21. }

**Output:**

100 Joy

100 Joy

In the above example, we have created two instances of the ConstructorDemo and passed the first object value into the second constructor. This way, we can use a copy constructor in Java.

5) Is there any method to call a sub-class constructor from a superclass constructor?

The subclass constructor has its own private data members, so Java does not provide any way to access the sub-class constructor from a super class constructor. However, we can call a superclass constructor from a sub-class constructor by using the super keyword.

6) Can we have a constructor in the Interface?

No, we cannot have constructors in the Java interface.

7) Explain Constructor Chaining?

Constructor Chaining is a way to call one constructor from another constructor with respect to the current object. It can be achieved in the following two ways:

**From base class:** We can use the super keyword to call a constructor from the base class.

**Within the same class:** We can call a constructor within the same class by using **this()** keyword.

Below is an example of constructor chaining:

1. **class** TestSuper
2. {
3. **public** TestSuper(**int** i)
4. {
5. System.out.println("Super Class Constructor");
6. }
7. }
9. **class** TestSub **extends** TestSuper
10. {
11. **public** TestSub()
12. {
13. **this**(10);      //Calling same class constructor
14. }
16. **public** TestSub(**int** i)
17. {
18. **super**(i);      //Calling super class constructor
19. }
20. }

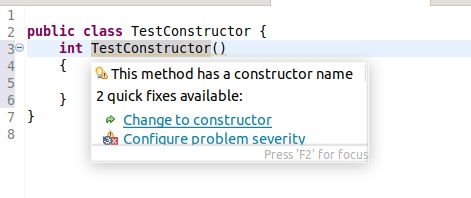
8) What happens if we provide a return type to a constructor?

If we provide a return type to a constructor, it will function as a general method. But, the compiler will display a warning message, "**This method has a Constructor name**".

**Consider the below example:**

1. **public** **class** TestConstructor {
2. **int** TestConstructor()
3. {
4. **return** 0;
5. }

The above program will be compiled gracefully, but it displays below warning message:



9) What is a private constructor?

Like methods, we can have the private constructors in Java. To make or create a constructor as private, use the **private** keyword while declaring it. It can only be accessed within that class.

The following are some usage scenarios when we need a private constructor:

* Internal Constructor chaining
* Singleton class design pattern

Below is an example of the private constructor:

**PrivateConstructor.java:**

1. **class** SingletonDemo {
2. **private** SingletonDemo() {
3. System. out.println("In a private constructor");
4. }
5. **public** **static** SingletonDemo getObject() {
6. // we can call this constructor
7. **if** (ref == **null**)
8. ref = **new** SingletonDemo();
9. **return** ref;
10. }
11. **private** **static** SingletonDemo ref;
12. }
13. **public** **class** PrivateConstructor {
14. **public** **static** **void** main(String args[]) {
15. SingletonDemo sObj = SingletonDemo.getObject();
16. }
17. }

**Output:**

In a private constructor

10) Why constructors in Java cannot be static?

The constructors cannot be static in Java. When we declare a method as static, it means the method belongs to the class and not to a specific object. But the constructor is always invoked to the reference of objects. So, there is no sense in making a constructor static.

11) Can we make a constructor final?

No, we cannot make a constructor final. If we made a constructor final, it would throw a compile-time error "**modifier final not allowed**".

12) Can we make a constructor abstract?

a body, which really makes no sense. It is automatically called at the time of object creation. So, it cannot be a block without a body.

13) what will happen when a constructor is declared as protected?

Generally, when we declare a method as protected, other classes can access that method in a different package by using inheritance only. But, when we declare a constructor protected, it behaves slightly differently than a method. The protected constructor can only be accessed by using a super keyword according to Java language standards.

14) Why constructor name is similar to the class name?

When we create an object of a class using a new keyword, it should have information about that particular class. That is why the constructor's name must be similar to the class name.

15) Why return type is not allowed for the constructor?

The return type is not allowed in the constructor because if we provide a return type in the constructor, it will act as the normal method. So, to differentiate between constructor and method block, the return type is not allowed in constructors.

**1. What is a**[**Constructor in Java**](https://javainterviewpoint.com/constructor-in-java/)**?**

**Constructor** is just like a method in [**Java**](https://javainterviewpoint.com/category/core-java/) that is used to initialize the state of an object and will be invoked during the time of object creation.

**2. What are the Rules for defining a constructor?**

1. Constructor **name** should be the same as the class name
2. It **cannot** contain any **return type**
3. It **can** have all **Access Modifiers** are allowed (private , public, protected, default)
4. It **Cannot** have any **Non Access Modifiers** (final ,static, abstract, synchronized)
5. **No return** statement is allowed
6. It **can** take any number of **parameters**
7. Constructor can **throw exception**, we can have**throws clause**

**3. What is the**[**use of Private Constructors in Java**](https://javainterviewpoint.com/what-is-the-use-of-a-private-constructors-in-java/)**?**

When we use **private** for a constructor then object for the class can only be created **internally** within the class, **no outside class** can create object for this class. Using this we can **restrict** the caller from creating objects.

class PrivateConstructorExample

{

/\*\*

\* Private Constructor for preventing object creation

from outside class

\*\*/

private PrivateConstructorExample(){ }

public void disp()

{

System.out.println("disp() method called");

}

}

public class Sample

{

public static void main(String args[])

{

//Creating the object for the Private Constructor class

PrivateConstructorExample pc = new PrivateConstructorExample();

pc.disp();

}

}

When we run the above code we will be getting the below exception.

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The constructor PrivateConstructorExample() is not visible

at Sample.main(Sample.java:19)

**4. Can we have a**[**Constructor in an Interface**](https://javainterviewpoint.com/java-constructor-in-an-interface/)**?**

**No**, We cannot have a Constructor defined in an [**Interface**](https://javainterviewpoint.com/interface-java/).

**5. What is**[**Constructor Chaining in Java**](https://javainterviewpoint.com/java-constructor-chaining-with-example/)**?**

**Constructor Chaining** is nothing but calling one Constructor from another. [**this keyword**](https://javainterviewpoint.com/java-keyword/) is used to call the **current** class constructor and [**super keyword**](https://javainterviewpoint.com/java-super-keyword/) is used to call the **parent** class constructor.

class Parent

{

public Parent()

{

System.out.println("Parent class no-args constructor called");

}

public Parent(String name)

{

System.out.println("Parent class Parameterized constructor called by "+name);

}

}

public class Child extends Parent

{

public Child()

{

this("JIP");

System.out.println("Child class no-args constructor called");

}

public Child(String name)

{

super("JIP");

System.out.println("Child class Parameterized constructor called by "+name);

}

public static void main(String args[])

{

Child c = new Child();

}

}

**Output :**

Parent class Parameterized constructor called by JIP

Child class Parameterized constructor called by JIP

Child class no-args constructor called

**6. Can we have this and super in the same constructor?**

**No,** we **cannot** have have **this** and **super** in a same constructor as any one only can be in the first line of the constructor.

class Parent

{

public Parent()

{

System.out.println("Parent class no-args constructor");

}

}

public class Child extends Parent

{

public Child()

{

this("JIP");

super();

System.out.println("Child class no-args constructor");

}

public Child(String name)

{

System.out.println("Child class Parameterized constructor"+name);

}

public static void main(String args[])

{

Child c = new Child();

}

}

**Output :**

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Constructor call must be the first statement in a constructor

at Child.(Child.java:13)

at Child.main(Child.java:23)

**7. Is it possible to call a sub class constructor from super class constructor?**

**No**. You cannot call a sub class constructor from a super class constructor.

**8. What is a No-arg constructor?**

Constructor **without arguments** is called no-arg constructor. In Java Default constructor is a no-arg constructor.

class Demo

{

public Demo()

{

//No-arg constructor

}

}

**9. Can we have a class with no Constructor in it ? What will happen during object creation ?**

**Yes**, we can have a class with no constructor, When the compiler encounters a class with no constructor then it will automatically create a default constructor for you.

**10. Can we have both Default Constructor and Parameterized Constructor in the same class?**

**Yes**, we have both Default Constructor and Parameterized Constructor in the same class.

**11. Can a Constructor return any value ?**

A Constructor cannot return any explicit value but implicitly it will be returning the instance of the class.

**12. Will compiler create the Default Constructor when we already have a Constructor defined in the class ?**

**No,** the compiler will not create the Default Constructor when we already have a Constructor defined.

**13. Can an**[**abstract class in Java**](https://javainterviewpoint.com/abstract-class-java/)**have a constructor?**

**Yes,** an abstract class can have a constructor. The below code work perfectly fine.

abstract class Demo1 {

String value;

public Demo1( String value ) {

this.value = value;

}

public String getValue()

{

return value;

}

}

public class Test extends Demo1 {

public Test() {

super("CoreJava");

}

}

**14. What happens when a Constructor is defined as “protected” ?**

In general **protected** method can be accessed by other class in a different package only through [**Inheritance**](https://javainterviewpoint.com/inheritance-in-java/). But when you assign protected access to a constructor it behaves a bit different. It can be accessed only by a call of **super() (according to**[**JLS**](https://docs.oracle.com/javase/specs/jls/se7/html/jls-6.html#jls-6.6.2)**)** and not directly by any other means.

package com.javainterviewpoint;

public class Parent

{

protected Parent()

{

System.out.println("Parent Constructor called");

}

public void parentDisp()

{

System.out.println("Parent Disp called");

}

}

package com.javainterviewpoint1;

import com.javainterviewpoint.Parent;

public class Child extends Parent

{

public Child()

{

/\*\*

\* Using super() Parent Class protected constructor can be called

\*/

super();

System.out.println("Child Constructor called");

}

public void childDisp()

{

System.out.println("Child Disp called");

}

public static void main(String args[])

{

/\*\*

\* Even though we have extended Parent class in Child class,

\* below way of calling Parent class Constructor is not allowed

\*

\* The constructor Parent() is not visible - error will be thrown

\*/

Parent p = new Parent() // Error will be thrown

}

}

**15. Why constructors cannot be final in Java?**

When you set a method as final, then” The method cannot be overridden by any class”, but **Constructor** by JLS ( [**Java Language Specification**](https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.8.3) ) definition can’t be overridden. A constructor is not inherited, so there is no need for declaring it as **final**.

**16. Why constructors cannot be abstract in Java?**

When you set a method as abstract, then “The method doesn’t or cannot have body”. A constructor will be automatically called when object is created. It cannot lack a body moreover an abstract constructor could never be implemented.

**17. Why constructors cannot be static in Java?**

When you set a method as static, it means “The Method belong to class and not to any particular object” but a constructor is always invoked with respect to an object, so it makes no sense for a constructor to be **static**.

What is the importance of default constructor?

The main reason for having a constructor is to initialize the variables. We know, in C we have to initialize a variable. Otherwise it will take some garbage value. But in Java we do not need to initialize any variable .The default constructor automatically initializes any variable with zero by default. This is the main importance of default constructor.

What is the importance of copy constructor?

Copy constructor is parameterized constructor and used when we want to copy the values of one object to another.

What will happen to the default version of constructor that is provided by the JVM if you have an explicit constructor in your class?

When we define an explicit constructor, JVM does not provide the default constructor. If we define a no-argument constructor then we have have to create an object which will invoke the no-argument constructor. Similarly if we define a parameterized constructor, we have to create a separate object and pass the parameters which will invoke the parameterized constructor.

Give an example of constructor overloading.

class A{

int a;

A(){

System.out.println(“AAA”);

}

A(int a1){

a=a1

System.out.println(a);

}

}

class B{

public static void main(String args[]){

A ob1=new A();

A ob2= new a(2);

}

}

When the constructor is called?

We define a constructor in the general class. As the scope of the general class is different from the main class, so to execute a constructor from the main class we have to create on object of the general class into the main class. Whenever we define an object the constructor is already invoked. There is no need to execute a constructor. Constructor is automatically executed when we create an object.

What is the importance of default constructor?

The main reason for having a constructor is to initialize the variables. We know, in C we have to initialize a variable. Otherwise it will take some garbage value. But in Java we do not need to initialize any variable .The default constructor automatically initializes any variable with zero by default. This is the main importance of default constructor.

What is copy constructor?

A copy constructor is a constructor that creates a new object using an existing object of the same class . It initializes each variable of newly created object with corresponding variables of existing object passed as argument. This constructor takes a single argument whose type is that of the class containing the constructor.

class Rectangle{

int l;

int b;

Rectangle(int l1, int b2){

l=l1;

b=b1;

}

Rectangle(Rectangle obj)##COPY CONSTRUCTOR

{

System.out.println(“Copy constructor invoked”);

l=obj.l;

b=obj.b;

}

int area(){

return(l\*b);

}

}

class CopyConstructor

{

public static void main(String[] args)

{

Rectangle R1 = new Rectangle(5,6);

Rectangle R2= new Rectangle(R1);

System.out.println(“Area of First Rectangle : ”+ R1.area());

System .out.println(“Area of First Second Rectangle :” + R2.area());

}

}

What is the importance of copy constructor?

Copy constructor is parameterized constructor and used when we want to copy the values of one object to another.