Top 17 Core Java Interview Questions on Constructors

1. What is a Constructor in Java?

Constructor is just like a method in **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?

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.

The major scenarios where we use private constructor:

- Internal Constructor chaining
- Singleton class design pattern

```
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?

No, We cannot have a Constructor defined in an **Interface**.

5. What is Constructor Chaining in Java?

Constructor Chaining is nothing but calling one Constructor from another. **this keyword** is used to call the **current** class constructor and **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 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**. 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)** 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**) 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**.

18. 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

19. 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

    }

}
```

20.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.
pr
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.

21. When do we need Constructor Overloading?

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading. Different constructors can do different work by implementing different line of codes and are called based on the type and no of parameters passed.
According to the situation, a constructor is called with specific number of parameters among overloaded constructors.

22. 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.

Quiz-

https://www.geeksforgeeks.org/java-gq/constructors-2-gq/

https://programmingtrick.com/quiz-java_constructors