

Constructors in Java

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

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

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

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

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

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

Note: We can use [access modifiers](#) while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

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

Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

1. `<class_name>(){}`

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

1. `//Java Program to create and call a default constructor`
2. `class Bike1{`
3. `//creating a default constructor`
4. `Bike1(){System.out.println("Bike is created");}`
5. `//main method`
6. `public static void main(String args[]){`
7. `//calling a default constructor`
8. `Bike1 b=new Bike1();`
9. `}`
10. `}`

Test it Now

Output:

```
Bike is created
```

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.

Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

```
1. //Let us see another example of default constructor
2. //which displays the default values
3. class Student3{
4.     int id;
5.     String name;
6.     //method to display the value of id and name
7.     void display(){System.out.println(id+" "+name);}
8.
9.     public static void main(String args[]){
10.        //creating objects
11.        Student3 s1=new Student3();
12.        Student3 s2=new Student3();
13.        //displaying values of the object
14.        s1.display();
15.        s2.display();
16.    }
17. }
```

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.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
1. //Java Program to demonstrate the use of the parameterized constructor.
2. class Student4{
3.     int id;
4.     String name;
5.     //creating a parameterized constructor
6.     Student4(int i,String n){
7.         id = i;
8.         name = n;
9.     }
10.    //method to display the values
11.    void display(){System.out.println(id+ " "+name);}
12.
13.    public static void main(String args[]){
14.        //creating objects and passing values
15.        Student4 s1 = new Student4(111,"Karan");
16.        Student4 s2 = new Student4(222,"Aryan");
17.        //calling method to display the values of object
18.        s1.display();
19.        s2.display();
20.    }
21.}
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java

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

Constructor [overloading in Java](#)

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

Example of Constructor Overloading

```
1. //Java program to overload constructors
2. class Student5{
3.     int id;
4.     String name;
5.     int age;
6.     //creating two arg constructor
7.     Student5(int i,String n){
8.         id = i;
9.         name = n;
10.    }
11.    //creating three arg constructor
12.    Student5(int i,String n,int a){
13.        id = i;
14.        name = n;
15.        age=a;
16.    }
17.    void display(){System.out.println(id+" "+name+" "+age);}
18.
19.    public static void main(String args[]){
20.        Student5 s1 = new Student5(111,"Karan");
21.        Student5 s2 = new Student5(222,"Aryan",25);
22.        s1.display();
23.        s2.display();
24.    }
25.}
```

Output:

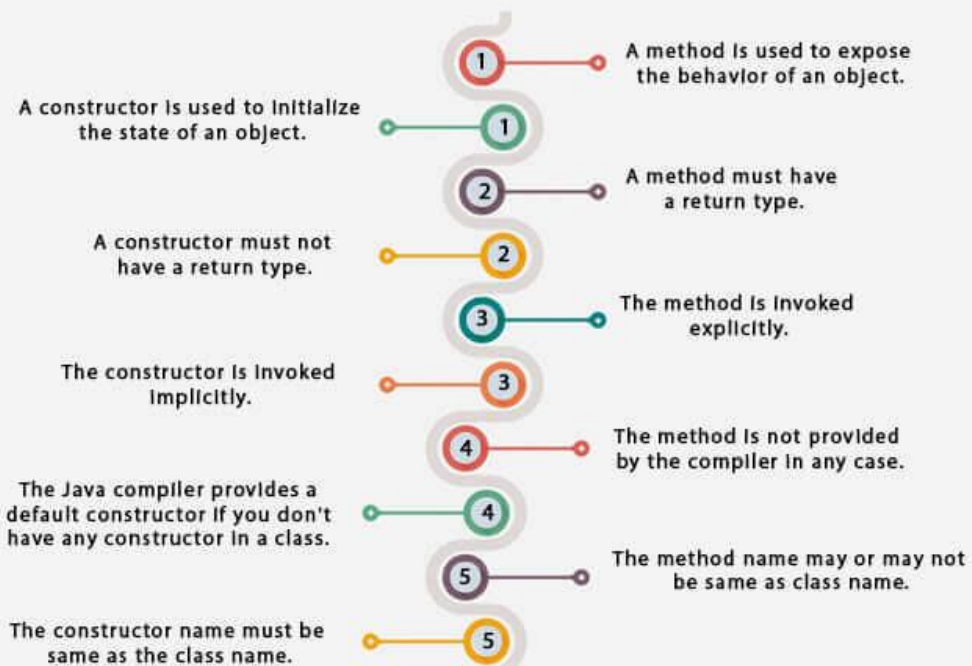
```
111 Karan 0
222 Aryan 25
```

Difference between constructor and method in Java

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 the class name.

Difference between constructor and method in Java



Java Copy Constructor

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.

//Java program to initialize the values from one object to another object.

```
1. class Student6{
2.     int id;
3.     String name;
4.     //constructor to initialize integer and string
5.     Student6(int i,String n){
6.         id = i;
7.         name = n;
8.     }
9.     //constructor to initialize another object
10.    Student6(Student6 s){
11.        id = s.id;
12.        name =s.name;
13.    }
14.    void display(){System.out.println(id+ " "+name);}
15.
16.    public static void main(String args[]){
17.        Student6 s1 = new Student6(111,"Karan");
18.        Student6 s2 = new Student6(s1);
19.        s1.display();
20.        s2.display();
21.    }
22.}
```


Output:

```
111 Karan
111 Karan
```

Copying values without constructor

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

```
1. class Student7{
2.     int id;
3.     String name;
4.     Student7(int i,String n){
5.         id = i;
6.         name = n;
7.     }
8.     Student7(){
9.         void display(){System.out.println(id+ " "+name);}
10.
11.     public static void main(String args[]){
12.         Student7 s1 = new Student7(111,"Karan");
13.         Student7 s2 = new Student7();
14.         s2.id=s1.id;
15.         s2.name=s1.name;
16.         s1.display();
17.         s2.display();
18.     }
19. }
```

Output:

```
111 Karan
111 Karan
```
