Constructors in Java

Difficulty Level : Easy Last Updated : 09 Feb, 2022

Java constructors or constructors in Java is a terminology been used to construct something in our programs. A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

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 the 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.

> ***Note:*** *It is not necessary to write a constructor for a class. It is because java compil er creates a default constructor if your class doesn't have any.*

How Constructors are Different From Methods in Java?

- Constructors must have the same name as the class within which it is defined while it is not necessary for the method in Java.
- Constructors do not return any type while method(s) have the return type or **void** if does not return any value.
- Constructors are called only once at the time of Object creation while method(s) can be called any number of times.

Now let us come up with the syntax for the constructor being invoked at the time of object or instance creation.

```
class Geek
{
  .......

  // A Constructor
  new Geek() {}
```

```
    .......
}

// We can create an object of the above class
// using the below statement. This statement
// calls above constructor.
Geek obj = new Geek();
```

Need of Constructor

Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in the computer's memory), then can a box be there with no value defined for its dimensions. The answer is no.
So constructors are used to assigning values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

When is a Constructor called?

Each time an object is created using a **new()** keyword, at least one constructor (it could be the default constructor) is invoked to assign initial values to the **data members** of the same class.

The rules for writing constructors are as follows:

- Constructor(s) of a class must have the same name as the class name in which it resides.
- A constructor in Java can not be abstract, final, static, or Synchronized.
- Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

So by far, we have learned constructors are used to initializing the object's state.
Like methods, a constructor also contains a collection of statements(i.e. instructions) that are executed at the time of Object creation.

Types of Constructors in Java

Now is the correct time to discuss types of the constructor, so primarily there are two types of constructors in java:

- No-argument constructor
- Parameterized Constructor

1. No-argument constructor

A constructor that has no parameter is known as the default constructor. If we don't define a constructor in a class, then the compiler creates a **default constructor(with no arguments)** for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.

> ***Note:*** *Default constructor provides the default values to the object like 0, null, etc. d epending on the type.*

Example:

```
// Java Program to illustrate calling a
// no-argument constructor

import java.io.*;

class Geek {
    int num;
    String name;

    // this would be invoked while an object
    // of that class is created.
    Geek() { System.out.println("Constructor called"); }
}

class GFG {
    public static void main(String[] args)
    {
        // this would invoke default constructor.
        Geek geek1 = new Geek();

        // Default constructor provides the default
        // values to the object like 0, null
        System.out.println(geek1.name);
```

```
        System.out.println(geek1.num);
    }
}
```

Output

```
Constructor called
null
0
```

2. Parameterized Constructor

A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with our own values, then use a parameterized constructor.

Example:

```
// Java Program to Illustrate Working of
// Parameterized Constructor

// Importing required inputoutput class
import java.io.*;

// Class 1
class Geek {
    // data members of the class.
    String name;
    int id;

    // Constructor would initialize data members
    // With the values of passed arguments while
    // Object of that class created
    Geek(String name, int id)
    {
        this.name = name;
        this.id = id;
    }
}

// Class 2
class GFG {
    // main driver method
    public static void main(String[] args)
```

```
    {
        // This would invoke the parameterized constructor.
        Geek geek1 = new Geek("adam", 1);
        System.out.println("GeekName :" + geek1.name
                           + " and GeekId :" + geek1.id);
    }
}
```

Output

```
GeekName :adam and GeekId :1
```

> ***Remember: Does constructor return any value?***
>
> *There are no "return value" statements in the constructor, but the constructor return s the current class instance. We can write 'return' inside a constructor.*

Now the most important topic that comes into play is the strong incorporation of OOPS with constructors known as constructor overloading. JustLike methods, we can overload constructors for creating objects in different ways. Compiler differentiates constructors on the basis of numbers of parameters, types of the parameters, and order of the parameters.

Example:

```
// Java Program to illustrate constructor overloading
// using same task (addition operation ) for different
// types of arguments.

import java.io.*;

class Geek
{
    // constructor with one argument
    Geek(String name)
    {
        System.out.println("Constructor with one " +
                    "argument - String : " + name);
```

```
    }

    // constructor with two arguments
    Geek(String name, int age)
    {

        System.out.println("Constructor with two arguments : " +
                " String and Integer : " + name + " "+ age);

    }

    // Constructor with one argument but with different
    // type than previous..
    Geek(long id)
    {
        System.out.println("Constructor with one argument : " +
                                            "Long : " + id);
    }
}

class GFG
{
    public static void main(String[] args)
    {
        // Creating the objects of the class named 'Geek'
        // by passing different arguments

        // Invoke the constructor with one argument of
        // type 'String'.
        Geek geek2 = new Geek("Shikhar");

        // Invoke the constructor with two arguments
        Geek geek3 = new Geek("Dharmesh", 26);

        // Invoke the constructor with one argument of
        // type 'Long'.
        Geek geek4 = new Geek(325614567);
    }
}
```

Output

```
Constructor with one argument - String : Shikhar
Constructor with two arguments :  String and Integer : Dharmesh 26
Constructor with one argument : Long : 325614567
```

In order to know to deep down into constructors there are two concepts been widely used as listed below:

- Constructor Chaining
- Copy constructor

This article is contributed by **Nitsdheerendra**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.