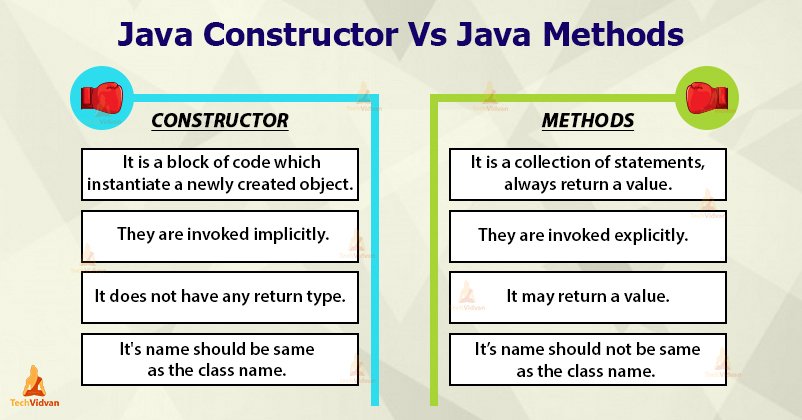
Java Constructor – An Exclusive Guide on Constructors

In this Java tutorial, we are going to discuss everything that you must know about a Constructor in Java. Constructor in Java is a block of code that creates an object. We can also call it an **Object Builder**.

They are similar to methods in Java but they differ from methods in the fact that they do not have a return type like methods.

In this article, we will learn what a constructor is, the need for constructors, its types, and the rules for writing constructors in Java. We will also cover some other topics like Constructor Overloading and Constructor Chaining.

We will also see how the methods are different from the constructors in Java.

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/03/java-constructor-vs-java-method.jpg)

Constructor in Java

*“A****Constructor****is a member function which has the same name as its class and is used to initialize the object of that class type with the legal initial value.”*

A constructor is a member function of a class that is called for initializing objects when we create an object of that class. It is a special type of method that instantiates a newly created object and just after the memory allocation of this object takes place, the constructor is called.

The name of the constructor is the same name as that of the class and its primary job is to initialize the object with a legal initial value for the class. It is not necessary for Java coders to write a constructor for a class.

***Note:****When we create an object of a class, at least one constructor is called. If we do not write any constructor in the class then the default constructor is called.*

Need for Java Constructor

We can use the constructors when we want to assign values to the class variables at the time of object creation. To understand the importance of constructors, let’s take an example. Suppose there is a Table.

If we create a class named **Apple**, then it will have some class variables like shape, color, and taste. But, when we create an object of the class Apple, now **Apple** will reside in the computer’s memory. Can we define an **Apple** with no value defined for its properties? We can definitely not do this.

The constructors allow us to define the values while creating the objects. We can create a constructor either explicitly through programming and if we do not define it explicitly, the Java compiler itself defines the default constructor.

Rules for Writing Constructors in Java

Following are some rules for writing constructors in Java:

* The name of the constructor must be the same as the name of its class.
* A constructor must have no return type. It can not have not even void as its return type.
* We can use the access modifiers with a constructor to control its access so that other classes can call the constructor.
* We can not declare a constructor as final, abstract, abstract and synchronized.

The syntax for writing a Constructor

A constructor has the same name as the class and we can write a constructor as follows:

public **class** MyClass

{

//This is the Constructor

MyClass()

{

//Constructor body

}

..

}

Note that the constructor name matches the class name and it is without a return type.

How does constructor work in Java?

Let’s take an example to understand the working of a constructor. Suppose we have a class named MyClass. When we initialize or create the object of MyClass it looks like this:

MyClass obj = **new** MyClass();

In the above line, the **new** keyword creates the object of class MyClass and invokes or calls the constructor to initialize this newly created object.

Types of Constructor in Java

There are two types of constructors in Java, which are:

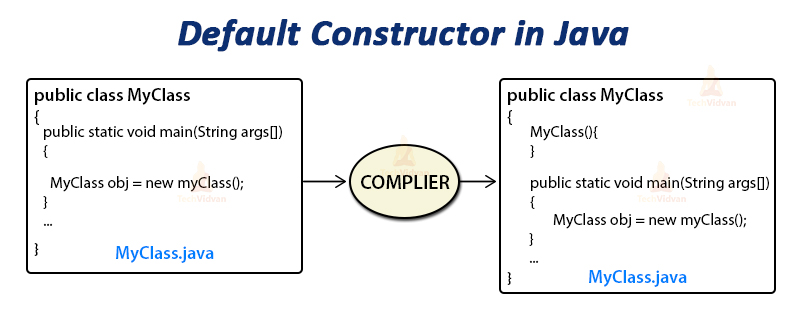
* Default Constructor
* Parameterized Constructor

Let’s discuss each of them with examples:

1. Default Constructor

A Default Constructor is a constructor with no parameter. The Java compiler automatically creates a default constructor if we do not write any constructor in our program.

This default constructor is not present in your source code or the java file as the compiler automatically puts it into the Java code during the compilation process and therefore we can’t find it in our java file, rather it exists in the bytecode or .class file. The following figure shows this process:

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/03/default-constructor-in-java.jpg)

If we do not provide an user-defined constructor in a class, the compiler initializes member variables to its default values such as:

* numeric data types set to 0
* char data types set to a null character (‘\0’)
* reference variables set to null

**Code to understand Default Constructors:**

package com.techvidvan.constructors;

**class** TechVidvan

{

int number;

String name;

TechVidvan()

{

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

}

}

public **class** DefaultConstructor

{

public static **void** main(String[] args)

{

TechVidvan object = **new** TechVidvan();

System.out.println(object.name);

System.out.println(object.number);

}

}

**Output:**

Default Constructor called  
null  
0

**Note:** If you implement any constructor then the Java compiler will no longer provide a default constructor.

2. Parameterized Constructor

A Parameterized constructor is a constructor with a specific number of parameters. We can use parameterized constructor mainly to initialize the members of the class with different values or objects.

**Code to understand Parameterized Constructors:**

package com.techvidvan.constructors;

**class** TechVidvan

{

String name;

int id;

//Creating a parameterized constructor

TechVidvan(String name, int id)

{

**this**.name = name;

**this**.id = id;

}

}

public **class** ParamaterizedConstructor

{

public static **void** main (String[] args)

{

TechVidvan object = **new** TechVidvan("Raj", 16);

System.out.println("Name: " + object.name );

System.out.println("id: " + object.id);

TechVidvan object1 = **new** TechVidvan1("Shivani", 24);

System.out.println("Name: " + object1.name );

System.out.println("id: " + object1.id);

}

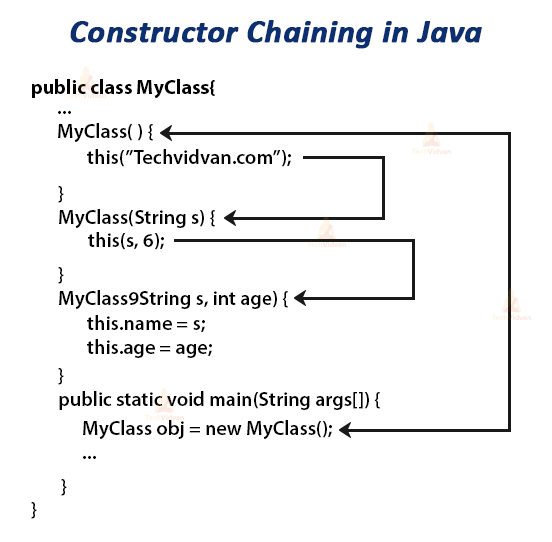
}

**Output:**

Name: Raj  
id: 16  
Name: Shivani  
id: 24

Constructor Chaining in Java

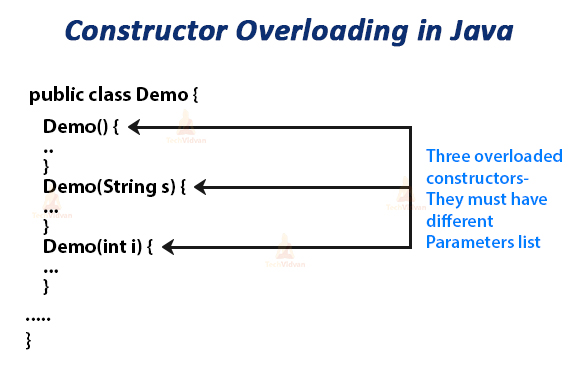
Constructor Chaining in Java is a process in which a constructor calls another constructor of the same class with the current/present object. The concept of constructor chaining helps to pass the parameters through different constructors, but with the same object.

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/03/constructor-chaining-in-java-1.jpg)

Constructor Overloading in Java- Multiple Constructors for a Java Class

Overloading generally means *“to have multiple instances of the same thing”.* Constructor Overloading in Java is a process of having more than one constructor with different parameters list.

It allows the constructor to behave differently and perform a different task with respect to its parameters. Constructor overloading is the same as [***method overloading in Java.***](https://techvidvan.com/tutorials/method-overloading-and-overriding/)

[](https://techvidvan.com/tutorials/wp-content/uploads/sites/2/2020/03/constructor-overloading-in-java.jpg)

**Code to understand Constructor Overloading in Java:**

package com.techvidvan.constructors;

**class** TechVidvan

{

TechVidvan(String name)

{

System.out.println("Constructor with one parameter: String: ");

System.out.println("Name: " +name);

}

TechVidvan(String name, int age)

{

System.out.println("Constructor with two parameters: String and Integer: ");

System.out.println("Name: " +name);

System.out.println("Age: " +age);

}

TechVidvan(long id)

{

System.out.println("Constructor with one parameter: Long: ");

System.out.println("id: " +id);

}

}

public **class** ConstructorOverloading

{

public static **void** main(String[] args)

{

TechVidvan ObjectName = **new** TechVidvan("Sameer");

TechVidvan ObjectName1 = **new** TechVidvan("Neeraj", 25);

TechVidvan ObjectName2 = **new** TechVidvan(235784567);

}

}

**Output:**

Constructor with one parameter: String:  
Name: Sameer  
Constructor with two parameters: String and Integer:  
Name: Neeraj  
Age: 25  
Constructor with one parameter: Long:  
id: 235784567

Constructor and Inheritance: Super() Keyword

The compiler implicitly invokes the constructor of the parent class whenever we invoke or call a constructor of its child class. To do this, the Java compiler inserts a super() keyword at the beginning of the child class constructor.

In the below code, we called the child class constructor but firstly the constructor of parent class runs and then the constructor of child class id executed because the compiler puts the super keyword at the beginning of child class constructor,

therefore, the control first moves to the parent class constructor and then to the child class constructor.

**class** Parent

{

Parent()

{

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

}

}

**class** Child **extends** Parent

{

Child()

{

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

}

public static **void** main(String args[])

{

**new** Child();

}

}

**Output:**

Parent Class Constructor  
Child Class Constructor

Difference Between Constructor and Method in Java

The following points explain the difference between constructor and method in Java.

* A constructor is a block of code that instantiates a newly created object, while a method is a set of statements that always return value depending upon its execution.
* The constructor’s name should be the same as the class name. On the other hand, the name of the method should not be the same as the class name.
* Constructors are called implicitly, while we call the methods explicitly.
* A constructor should not have any return type, not even void, but methods must have a return type.
* The compiler automatically creates the constructor if there is no constructor in the class. But, in the case of the method, there is no default method provided by the compiler.
* We can override a method but we can’t override a constructor.

Important Points

* Constructors are called implicitly when we instantiate objects with the new operator.
* The two rules for creating a constructor are:

1. A name of a Java constructor name must exactly match with the class name.
2. A Java constructor must not have a return type.

* If there is no constructor in a class then the Java compiler automatically creates a default constructor during the compilation.
* We can’t declare constructors as abstract, synchronized, static or final.
* We can overload a constructor but we can’t override a constructor.
* Every class has a constructor whether it is a concrete class or an abstract class.
* A constructor can use any access specifier.
* ***Interfaces*** can not have constructors