In Java, **constructors** play an important role in object creation. A constructor is a special block of code that is called when an object is created. Its main job is to initialize the object, to set up its internal state, or to assign default values to its attributes. This process happens automatically when we use the "new" keyword to create an object.

**Characteristics of Constructors:**

* **Same Name as the Class:** A constructor has the same name as the class in which it is defined.
* **No Return Type:** Constructors do not have any return type, not even void. The main purpose of a constructor is to initialize the object, not to return a value.
* **Automatically Called on Object Creation:**When an object of a class is created, the constructor is called automatically to initialize the object’s attributes.
* **Used to Set Initial Values for Object Attributes:** Constructors are primarily used to set the initial state or values of an object’s attributes when it is created.

Now, let us look at a simple example to understand how a constructor works in Java.

**Example:** This program demonstrates how**a constructor is automatically called when an object is created in Java.**

// Java Program to demonstrate

// Constructor usage

import java.io.\*;

​

// Driver Class

class Test {

// Constructor

Test()

{

super();

System.out.println("Constructor Called");

}

​

// main function

public static void main(String[] args)

{

Test Test =new Test();

}

}

**Output**

Constructor Called

**Note:**It is not necessary to write a constructor for a class. It is because the Java compiler creates a default constructor (constructor with no arguments) if your class doesn't have any.

**Constructor vs Method in Java**

The below table demonstrates the key difference between Java Constructor and Java Methods.

| **Features** | **Constructor** | **Method** |
| --- | --- | --- |
| **Name** | Constructors must have the same name as the class name | Methods can have any valid name |
| **Return Type** | Constructors do not return any type | Methods have the return type or void if does not return any value. |
| **Invocation** | Constructors are called automatically with new keyword | Methods are called explicitly |
| **Purpose** | Constructors are used to initialize objects | Methods are used to perform operations |

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

*class Test*

*{   
......*

*// A Constructor*

*Test() {*

*}  
.......*

*}*

*// We can create an object of the above class  
// using the below statement. This statement  
// calls above constructor.*

*Test obj = new Test();*

The first line of a constructor is a call to **super()** or **this()**, (a call to a constructor of a super-class or an overloaded constructor), if you don't type in the call to super in your constructor the compiler will provide you with a non-argument call to super at the first line of your code, the super constructor must be called to create an object:

**Note:** If you think your class is not a subclass it actually is, every class in Java is the subclass of a class **object**even if you don't say extends object in your class definition.

**Why Do We Need Constructors in Java**

Constructors play a very important role, it ensures that an **object is properly initialized before use.**

**What happens when we don't use constructors:**

Without constructors:

* Objects might have undefined or default values.
* Extra initialization methods would be required.
* Risk of improper object state

*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 N****o****.   
So, constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).*

**When Java Constructor is 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. Rules for writing constructors are as follows:

* The constructor 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, we have learned constructors are used to initialize the object's state. Like [methods](https://www.geeksforgeeks.org/java/methods-in-java/) , 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 the types of the constructor, so primarily there are three types of constructors in Java are mentioned below:

* Default Constructor
* Parameterized Constructor
* Copy Constructor

**1. Default Constructor in Java**

**A constructor that has no parameters is known as default constructor.** A default constructor is invisible. And if we write a constructor with no arguments, the compiler does not create a default constructor. Once you define a constructor (with or without parameters), the compiler no longer provides the default constructor. Defining a parameterized constructor does not automatically create a no-argument constructor, we must explicitly define if needed. The default constructor can be implicit or explicit.

* **Implicit Default Constructor:**If no constructor is defined in a class, the Java compiler automatically provides a default constructor. This constructor doesn’t take any parameters and initializes the object with default values, such as 0 for numbers, null for objects.
* **Explicit Default Constructor:**If we define a constructor that takes no parameters, it's called an explicit default constructor. This constructor replaces the one the compiler would normally create automatically.Once you define any constructor (with or without parameters), the compiler no longer provides the default constructor for you.

**Example:**This program demonstrates**the use of a default constructor, which is automatically called when an object is created.**

// Java Program to demonstrate

// Default Constructor

import java.io.\*;

​

// Driver class

class Tests{

​

// Default Constructor

Tests() {

System.out.println("Default constructor");

}

​

// Driver function

public static void main(String[] args)

{

Tests hello = new Tests();

}

}

**Output**

Default constructor

**Note:**Default constructor provides the default values to the object like 0, null, false etc. depending on the type.

**2. Parameterized Constructor in Java**

**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:**This program demonstrates the **use of a parameterized constructor to initialize an object's attributes with specific values.**

// Java Program for Parameterized Constructor

import java.io.\*;

​

class Tests {

// data members of the class

String name;

int id;

Tests(String name, int id) {

this.name = name;

this.id = id;

}

}

​

class GFG

{

public static void main(String[] args)

{

// This would invoke the parameterized constructor

Tests Test1 = new Tests("Sweta", 68);

System.out.println("TestName: " + Test1.name

+ " and TestId: " + Test1.id);

}

}

**Output**

TestName: Sweta and TestId: 68

***Remember: Does constructor return any value?***

*There are no "return value" statements in the constructor, but the constructor returns the current class instance. We can write 'return' inside a constructor.*

**3. Copy Constructor in Java**

Unlike other constructors [copy constructor](https://www.geeksforgeeks.org/java/copy-constructor-in-java/) is passed with another object which copies the data available from the passed object to the newly created object.

***Note:*** *Java does not provide a built-in copy constructor like C++. We can create our own by writing a constructor that takes an object of the same class as a parameter and copies its fields.*

**Example:**This example, demonstrates**how a copy constructor can be used to create a new object by copying the values of another object's attributes.**

// Java Program for Copy Constructor

import java.io.\*;

​

class Tests {

// data members of the class

String name;

int id;

​

// Parameterized Constructor

Tests(String name, int id)

{

this.name = name;

this.id = id;

}

​

// Copy Constructor

Tests(Tests obj2)

{

this.name = obj2.name;

this.id = obj2.id;

}

}

​

class GFG {

public static void main(String[] args)

{

// This would invoke the parameterized constructor

System.out.println("First Object");

Tests Test1 = new Tests("Sweta", 68);

System.out.println("TestName: " + Test1.name

+ " and TestId: " + Test1.id);

​

System.out.println();

​

// This would invoke the copy constructor

Tests Test2 = new Tests(Test1);

System.out.println(

"Copy Constructor used Second Object");

System.out.println("TestName: " + Test2.name

+ " and TestId: " + Test2.id);

}

}

**Output**

First Object

TestName: Sweta and TestId: 68

Copy Constructor used Second Object

TestName: Sweta and TestId: 68

**Constructor Overloading**

This is a key concept in [OOPs](https://www.geeksforgeeks.org/java/object-oriented-programming-oops-concept-in-java/) related to constructors is [constructor overloading](https://www.geeksforgeeks.org/java/constructor-overloading-java/). This allows us to create multiple constructors in the same class with different parameter lists.

**Example:**This example, demonstrates constructor overloading, **where multiple constructors perform the same task (initializing an object) with different types or numbers of arguments.**

// Java Program to illustrate constructor overloading

// using same task (addition operation ) for different

// types of arguments

import java.io.\*;

​

class Tests {

// constructor with one argument

Tests(String name)

{

System.out.println("Constructor with one "

+ "argument - String: " + name);

}

​

// constructor with two arguments

Tests(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

Tests(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 'Test'

// by passing different arguments

​

// Invoke the constructor with one argument of

// type 'String'.

Tests Test2 = new Tests("Sweta");

​

// Invoke the constructor with two arguments

Tests Test3 = new Tests("Amiya", 28);

​

// Invoke the constructor with one argument of

// type 'Long'.

Tests Test4 = new Tests(325614567);

}

}

**Output**

Constructor with one argument - String: Sweta

Constructor with two arguments: String and Integer: Amiya 28

Constructor with one argument: Long: 325614567

**Common Mistakes to Avoid**

Some common mistakes to avoid when working with constructors in Java are listed below:

* **Forgetting super() in Child Classes:**Always call the parent constructor (super()) if the parent class has no default constructor, or it will lead to compilation errors.
* **Excessive Work in Constructors:** Keep constructors simple and focused on initialization, avoiding heavy logic that slows down object creation.
* **Not Handling Null Checks:** Always validate parameters to avoid NullPointerException when constructing objects.