Constructor in java:

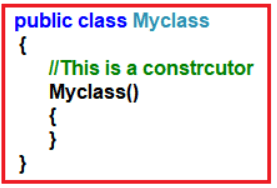
**What are Constructors in Java?**

Java allows the object to initialize itself when it is created. This automatic initialization is known as Constructors. The constructor is a block of code that initializes the newly created object.

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is called automatically immediately after the object is created before the new operator completes it. Constructors have no return type, not even void.

**Defining a Constructor in Java:**

This is a simple Java Constructor Declaration example:



**How does Constructor work in Java?**

Let’s say we have a class called MyClass in the above declaration. When we create the object of MyClass like this:

**Myclass myclassobj = new Myclass();**

The **new**keyword here creates the object of class MyClass and invokes the constructor to initialize this newly created object. Each time an object is created using the **new** keyword, at least one constructor (the default constructor) is invoked to assign initial values to the **data members**of the same class.

**Why do we need a Constructor?**

**Constructors** initialize the new object; that is, they set the startup property values for the object. They might also do other things necessary to make the object usable. You can distinguish constructors from other methods of a class because **constructors** always have the same name as the class.

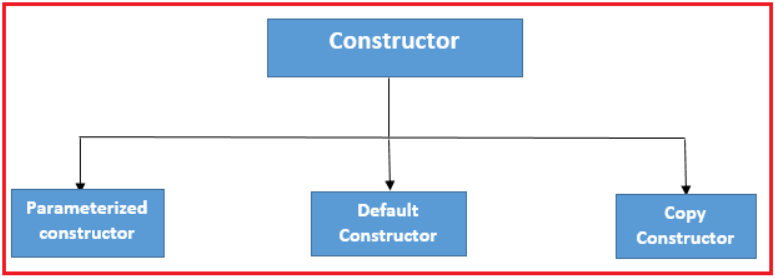
**Rules for Constructors:**

* The constructor’s name must be the same as that of the class name in which it resides.
* Constructors must not have a return type. If you keep the return type for the constructor, it will be treated as a method.
* Every class should have at least one constructor. If you don’t write a constructor for your class, the compiler will give a default constructor.
* A constructor in Java cannot be abstract, final, static, and Synchronized.
* Access modifiers can be used in constructor declaration to control its access, i.e., which other class can call the constructor.

**Types of Constructors in Java:**

Basically, there are three types of constructors in java:

1. Parameterized Constructors
2. Default Constructors
3. Copy Constructors



**Parameterized Constructor in Java:**

Constructors with parameters that can be used to initialize the internal state (fields) of the newly created object are known as Parameterized Constructors.  If we want to initialize class fields with our own values, then use a parameterized constructor.

**Example to Understand Parameterized Constructor in Java:**

**package** *Demo*;

**import** *java.io.*\*;

**class** Student

**{**

// 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.

Student **(String** name, **int** id**)**

**{**

this.name = name;

this.id = id;

**}**

**}**

**public** **class** ParameterizedConstructor

**{**

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

**{**

//This would invoke the parameterized constructor.

Student student1 = new Student **(**"Ashok", 101**)**;

System.out.println **(**"Student Name: " + student1.name +" and Student Id: " + student1.id**)**;

**}**

**}**

**Output: Student Name: Ashok and Student Id: 101**

**Default Constructor in Java:**

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. Therefore, it is also known as a no-args constructor. Once the class is compiled, it will always at least have a no-argument constructor. If you define a constructor for your class, then the Java compiler will not insert the default no-argument constructor into your class.

**Sample Program for Default Constructor:**

**package** *Demo*;

**public** **class** DefaultConstructor

**{**

**public** DefaultConstructor **()**

**{**

System.out.println **(**"This is a no-argument constructor"**)**;

**}**

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

**{**

new DefaultConstructor**()**;

**}**

**}**

**Output: This is a no-argument constructor**

**Copy Constructor in Java:**

A copy constructor is used for copying one object’s values to another.

**Example to Understand Copy Constructor in Java:**

**package** *Demo*;

**public** **class** CopyConstructor

**{**

**String** web;

CopyConstructor **(String** w**)**

**{**

web = w;

**}**

/\* This is the Copy Constructor, it

\* copies the values of one object

\* to the another object (the object

\* that invokes this constructor)

\*/

CopyConstructor **(**CopyConstructor cc**)**

**{**

web = cc.web;

**}**

**void** disp **()**

**{**

System.out.println **(**"Constructor: " + web**)**;

**}**

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

**{**

CopyConstructor obj1 =new CopyConstructor **(**"Example of Copy Constructor in Java"**)**;

/\* Passing the object as an argument to the constructor

\* This will invoke the copy constructor

\*/

CopyConstructor obj2 = new CopyConstructor **(**obj1**)**;

obj1.disp **()**;

obj2.disp **()**;

**}**

**}**

**Output:**

**Example of Copy Constructor in Java**  
**Example of Copy Constructor in Java**

**Calling a Constructor from another Constructor in Java using this Keyword**

In Java, calling a constructor from inside another constructor is possible. When you call a constructor from inside another constructor, you need to use **this**keyword to refer to the constructor.

this keyword can be very useful in the handling of Variable Hiding. We cannot create two instances/local variables with the same name. However, it is legal to create one instance variable & one local variable or method parameter with the same name. In this scenario, the local variable will hide the instance variable this is called Variable Hiding.

**Example to understand this keyword for Variable Hiding:**

**package** *Demo*;

**public** **class** VariabeHiding

**{**

**int** variable = 5;

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

**{**

VariabeHiding obj = new VariabeHiding **()**;

obj.method **(**20**)**;

obj.method **()**;

**}**

**void** method **(int** variable**)**

**{**

variable = 10;

System.out.println **(**"Value of Instance variable: " + this.variable**)**;

System.out.println **(**"Value of Local variable: " + variable**)**;

**}**

**void** method **()**

**{**

**int** variable = 40;

System.out.println **(**"Value of Instance variable: " + this.variable**)**;

System.out.println **(**"Value of Local variable: " + variable**)**;

**}**

**}**

**Output:**

**Value of Local variable: 10**  
**Value of Instance variable: 5**  
**Value of Local variable: 40**

**Calling Constructors in Superclasses by using Super Keyword in Java:**

The **super** keyword refers to superclass objects. It is used to call superclass methods and to access the superclass constructor. The keyword “super” came into the picture with the concept of Inheritance. Inheritance in detail, we will cover in my tutorial of Inheritance in Java.

**Note:** The super keyword can be called both parametric and non-parametric constructors.

**Example to Understand Super Keyword in Java:**

**package** *Demo*;

**class** College

**{**

// Superclass (parent)

**public** **void** collegeRecord **()**

**{**

System.out.println **(**"The college is good with a good placement record"**)**;

**}**

**}**

**class** Students **extends** College

**{**

// Subclass (child)

**public** **void** collegeRecord **()**

**{**

// Call the superclass method

super.collegeRecord **()**;

System.out.println**(**"The students of this college are performing good in academics."**)**;

**}**

**}**

**public** **class** SuperDemo

**{**

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

**{**

College myStudent = new College **()**;

myStudent.collegeRecord **()**;

**}**

**}**

**Output: The college is good with a good placement record**

**Constructor Overloading in Java**

More than one constructor with a different signature in a class is called constructor overloading. The constructor’s signature includes the number, type, and sequence of arguments.

If two constructors in the same class have the same signature, it represents ambiguity. In this case, the Java compiler will generate an error message because the compiler cannot differentiate which form to use.

Hence, an overloaded constructor must have different signatures. The Java compiler decides which constructor has to be called depending on the number of arguments passing with the object.

**Example to Understand Constructor Overloading in Java:**

**package** *Demo*;

**class** Student1

**{**

**int** Roll;

**String** Name;

**double** Marks;

Student1**(int** R,**String** N,**double** M**)** // Constructor 1

**{**

Roll = R;

Name = N;

Marks = M;

**}**

Student1**(String** N,**double** M,**int** R**)** // Constructor 2

**{**

Roll = R;

Name = N;

Marks = M;

**}**

**void** Display**()**

**{**

System.out.print**(**"\n\t" + Roll+"\t" + Name+"\t" + Marks**)**;

**}**

**}**

**class** ConstructorOverloading

**{**

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

**{**

Student1 S1 = new Student1**(**1,"Kumar",78.53**)**; // Statement 2

Student1 S2 = new Student1**(**"Sumit",89.42,2**)**; // Statement 1

System.out.print**(**"\n\tRoll\tName\tMarks\n"**)**;

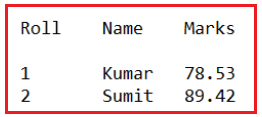
S1.Display**()**;

S2.Display**()**;

**}**

**}**

**Output:**



**Some interview questions related to Constructors in Java with Examples:**

**Can we define a method with the same class name?**

Yes, it is allowed to define a method with the same class name. NO compile-time error and NO runtime error are raised but it is not recommended as per coding standards.

**If we place the return type in the constructor prototype, will it lead to CE?**

No, because the compiler and JVM consider it as a method.

**How do the compiler and JVM differentiate constructor and method definitions if both have the same class name?**

By using return type. If there is a return type, then it is considered a method or a constructor.

**How do the compiler and JVM differentiate constructor and method invocations if both have the same class name?**

By using the new keyword. If the new keyword is used in calling, then the constructor is executed; else, the method is executed.

**Why return type is not allowed in the constructor?**

As there is a possibility to define a method with the same class name, the return type does not allow the constructor to differentiate the constructor block from the method block.

**Why should the constructor name should be the same as the class name?**

Every class object is created using the same new keyword, so it must have information about the class to which it must create an object. For this reason, the constructor name should be the same as the class name.

**Can we declare the constructor as private?**

Yes. We can declare the constructor as private. All four accessibility modifiers are allowed to the constructor. We should declare the constructor private for not allowing users to create an object outside our class. Basically, we will declare the constructor as private to implement a singleton design pattern.

**Is constructor definition mandatory in a class?**

No, it is optional. If we don’t define a constructor, then the compiler will define the constructor.

**How many types of constructors are there in Java?**

Java supports three types of constructors.

**Default constructor**: The compiler-given constructor is called the default constructor. It does not have parameters and logic except a super() call.

**No-argument/non-parameterized constructor**: The developer-given constructor without a parameter is called a no-argument or non-parameterized constructor.

**Parameterized constructor**: The developer-given constructor with parameters is called a parameterized constructor.

**Rule**: super() call must also be placed as the first statement in the developer-given constructor. If the developer does not place it, the compiler places this statement.

**Why is the compiler the given constructor called the default constructor?**

Because it obtains all its default properties from its class

1. Its accessibility modifier is the same as its class accessibility modifier.
2. Its name is the same as its class name.
3. It does not have parameters and logic

**What is the default accessibility modifier of a default constructor?**

* It is assigned from its class. So, it may be default or public.
* If a class is created with a default accessibility modifier, then the constructor is also created with a default accessibility modifier.
* If the class is created with a public accessibility modifier, then the constructor is also created with a public accessibility modifier.

**Why does the compiler define the default constructor without logic and parameters?**

* Because the compiler does not know the logic required for our class, the object’s initialization logic is different from one class to another class.
* But it places the super() method call in all constructors because it is generic logic required for every class for calling the superclass constructor in order to initialize superclass non-static variables when the subclass object is created.
* Since there is no logic in the default constructor compiler defines the default constructor without parameters.

**When does the compiler provide the default constructor?**

Only if there is no explicit constructor defined by the developer.

**When must the developer provide the constructor explicitly?**

If we want to execute some logic at the time of object creation, that logic may be object initialization logic or some other useful logic, then the developer must provide the constructor explicitly.

**If the class has an explicit constructor, does it have a default constructor?**

No, the compiler places the default constructor only if there is no explicit developer-given constructor.

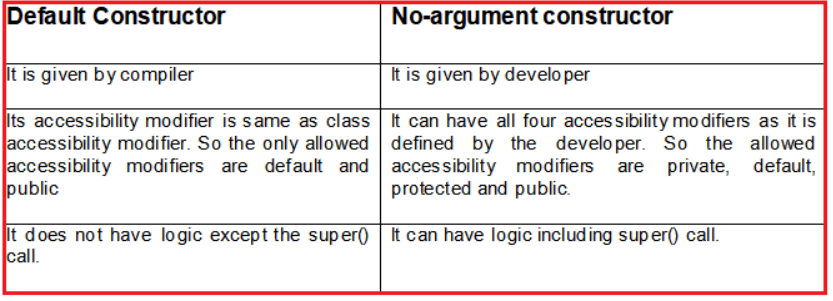
**How can we create an object of a class?**

We must create an object of a class by using the new keyword and available constructor.

**Can we consider both default and no-argument constructors to be the same?**

No, both are different. They seem to be the same but not really the same.

**What are the differences between a no-argument and a default constructor?**



**Note:**The common point between these two constructors is both constructors don’t have parameters.

**Can we define a class with a public accessibility modifier and a constructor with other accessibility modifiers?**

Yes, we can define it. The compiler does not change the developer-given constructor’s accessibility modifier to a class accessibility modifier.

**When should we define a parameterized constructor in a class?**

To initialize objects dynamically with the user-given values, we should define the parameterized constructor.

**How many constructors can be defined in a class?**

In a class, we can define multiple constructors, but every constructor must have a different parameter type and parameter order. So, in a class, we can define one no-argument constructor plus the ‘n’ number of parameterized constructors.

**What is constructor overloading?**

Defining multiple constructors with different parameter types/order/ list is called constructor overloading.

**In an object creation, apart from the invoked constructor, are other constructors also executed?**

No, only the invoked constructor is executed. Other constructors will not be executed.

**Then, how can we execute logic when an object is created using any of the constructors?**

**Solution 1**: Write that logic in all constructors.

**Problem**: It is not recommended because we lose code re-usability and centralized code change. Since code is redundant, we must perform code changes in every constructor, which leads to a lot of maintenance costs because, after code change, every constructor’s logic should be checked again.

**Solution 2**: As per modularity, write that logic in a non-static method and call that method in all constructors.

**Problem**: It is also not recommended because there is a chance of missing calling that method in one of the constructors, and also, that method can be called after object creation.

**Solution3**: The best solution for this requirement is a Non-Static Block (NSB)

**Difference Between Methods and Constructors in Java**

|  |  |
| --- | --- |
| **Constructors** | **Methods** |
| It is a special type of method used to initialize objects of their class. | Methods are a set of instructions that are invoked at any point in a program to return a result. |
| The purpose of a constructor is to create an instance of a class. | The purpose of a method is to execute Java code. |
| They are called implicitly immediately after object creation. | They are called directly without even creating an instance of that class. |
| They are used to initialize objects that don’t exist. | They perform operations on already-created objects. |
| The name of the constructor must be the same as the name of the class. | They can have arbitrary names in Java. |
| They are not inherited by subclasses. | They are inherited by subclasses. |