University of Engineering & Management, Kolkata

Department of CSE (IoT)

Subject Name: Advanced Programming

Subject Code: PCCCS405

Batch-2023-2027_2nd Year_4th Semester

Constructors

Java Constructors

- Java constructors or constructors in Java is a terminology 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.

What are 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 the constructor, memory for the object is allocated in the memory. It is a special type of method that is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called.

Example of Java Constructor

Below is the implementation of Java Constructors:

```
// Java Program to demonstrate
// Constructor
import java.io.*;


// Driver Class
```

```
class Geeks {
                                                                    7


                                                                    8
    // Constructor
                                                                    9
    Geeks()
                                                                    10
    {
                                                                    11
        super();
                                                                    12
        System.out.println("Constructor Called");
                                                                    13
    }
                                                                    14


                                                                    15
    // main function
                                                                    16
    public static void main(String[] args)
                                                                    17
    {
                                                                    18
        Geeks geek = new Geeks();
                                                                    19
    }
                                                                    20
}
```

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.*

How Java Constructors are Different From Java Methods?

- Constructors must have the same name as the class within which it is defined 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
    Geek() {
    }
    .......
}
// We can create an object of the above class
// using the below statement. This statement
// calls above constructor.
Geek obj = new Geek();
```

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: 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.

Need of Constructors in Java

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(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 initialize 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 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. It is taken out. It is being overloaded and called a parameterized constructor. The default constructor changed into the parameterized constructor. But Parameterized constructor can't change the default constructor. 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:

```
1
// Java Program to demonstrate
2
// Default Constructor
3
import java.io.*;
4


5
// Driver class
6
class GFG {
7
```

```
    // Default Constructor
    GFG() { System.out.println("Default constructor"); }


    // Driver function
    public static void main(String[] args)
    {
        GFG hello = new GFG();
    }
}
```

Output

```
Default constructor
```

Note: *Default constructor provides the default values to the object like 0, null, 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:

```
// Java Program for Parameterized Constructor

import java.io.*;



class Geek {

    // data members of the class.
```

```
    String name;
    int id;

    Geek(String name, int id) {
        this.name = name;
        this.id = id;
    }
}

class GFG
{
    public static void main(String[] args)
    {
        // This would invoke the parameterized constructor.
        Geek geek1 = new Geek("Avinash", 68);
        System.out.println("GeekName :" + geek1.name
                           + " and GeekId :" + geek1.id);
    }
}
```

Output

```
GeekName :Avinash and GeekId :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.
Now the most important topic that comes into play is the strong incorporation of OOPS with constructors known as constructor overloading. Just like methods, we can overload constructors for creating objects in different ways. The compiler differentiates constructors on the basis of the number of parameters, types of 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
```

3. Copy Constructor in Java

Unlike other constructors copy constructor is passed with another object which copies the data available from the passed object to the newly created object.

Note: *In Java,there is no such inbuilt copy constructor available like in other programming languages such as C++, instead we can create our own copy constructor by passing the object of the same class to the other instance(object) of the class.*

Example:

```
// Java Program for Copy Constructor

import java.io.*;



class Geek {

    // data members of the class.

    String name;

    int id;



    // Parameterized Constructor

    Geek(String name, int id)

    {

        this.name = name;

        this.id = id;

    }



    // Copy Constructor

    Geek(Geek 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");
        Geek geek1 = new Geek("Avinash", 68);
        System.out.println("GeekName :" + geek1.name
                           + " and GeekId :" + geek1.id);

        System.out.println();

        // This would invoke the copy constructor.
        Geek geek2 = new Geek(geek1);
        System.out.println(
            "Copy Constructor used Second Object");
```

```
        System.out.println("GeekName :" + geek2.name
                                                                    39
                    + " and GeekId :" + geek2.id);
                                                                    40
    }
                                                                    41
}
```

Output

```
First Object
GeekName :Avinash and GeekId :68


Copy Constructor used Second Object
GeekName :Avinash and GeekId :68
```

Constructor overloading in Java

In Java, we can overload constructors like methods. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Consider the following Java program, in which we have used different constructors in the class.

Example

```
1.  public class Student {
2.  //instance variables of the class
3.  int id;
4.  String name;
5.
6.  Student(){
7.  System.out.println("this a default constructor");
8.  }
9.
10. Student(int i, String n){
11. id = i;
12. name = n;
13. }
14.
15. public static void main(String[] args) {
16. //object creation
17. Student s = new Student();
```

```
18. System.out.println("\nDefault Constructor values: \n");
19. System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);
20.
21. System.out.println("\nParameterized Constructor values: \n");
22. Student student = new Student(10, "David");
23. System.out.println("Student Id : "+student.id + "\nStudent Name : "+student.name);
24. }
25. }
```

Output:

this a default constructor

Default Constructor values:

Student Id : 0
Student Name : null

Parameterized Constructor values:

Student Id : 10
Student Name : David

In the above example, the Student class constructor is overloaded with two different constructors, I.e., default and parameterized.

Here, we need to understand the purpose of constructor overloading. Sometimes, we need to use multiple constructors to initialize the different values of the class.

We must also notice that the java compiler invokes a default constructor when we do not use any constructor in the class. However, the default constructor is not invoked if we have used any constructor in the class, whether it is default or parameterized. In this case, the java compiler throws an exception saying the constructor is undefined.

Consider the following example, which contains the error since the Colleges object can't be created using the default constructor now since it doesn't contain one.

```
1. public class Colleges {
2. String collegeId;
3. Colleges(String collegeId){
4. this.collegeId = "IIT " + collegeId;
5. }
6. public static void main(String[] args) {
7. // TODO Auto-generated method stub
8. Colleges clg = new Colleges(); //this can't create colleges constructor now.
9. }
10.
11.
12. }
```

Use of this () in constructor overloading

However, we can use this keyword inside the constructor, which can be used to invoke the other constructor of the same class.

Consider the following example to understand the use of this keyword in constructor overloading.

```
1.  public class Student {
2.  //instance variables of the class
3.  int id,passoutYear;
4.  String name,contactNo,collegeName;
5.
6.  Student(String contactNo, String collegeName, int passoutYear){
7.  this.contactNo = contactNo;
8.  this.collegeName = collegeName;
9.  this.passoutYear = passoutYear;
10. }
11.
12. Student(int id, String name){
13. this("9899234455", "IIT Kanpur", 2018);
14. this.id = id;
15. this.name = name;
16. }
17.
18. public static void main(String[] args) {
19. //object creation
20. Student s = new Student(101, "John");
21. System.out.println("Printing Student Information: \n");
22. System.out.println("Name: "+s.name+"\nId: "+s.id+"\nContact No.: "+s.contactNo+"\nCollege
    Name: "+s.contactNo+"\nPassing Year: "+s.passoutYear);
23. }
24. }
```

Output:

Printing Student Information:

Name: John
Id: 101
Contact No.: 9899234455
College Name: 9899234455
Passing Year: 2018