18-default vs parameterized constructor

In Java, constructors are special methods used to initialize objects. When an object of a class is created, the constructor is automatically called. Constructors can either set default values or accept parameters to set custom values for object attributes.

<mark>**Types of Constructors in Java**</mark>

Constructors in Java can be broadly categorized into two types:

1. **Default Constructor:**

   o A constructor that has no parameters.

   o It is automatically provided by the Java compiler if no constructor is explicitly defined.

   o **Syntax:**

<class_name>() {}

   o If a class has no constructors, the compiler automatically creates a default constructor.

2. **Parameterized Constructor:**

   o A constructor that accepts parameters to initialize an object with specific values.

   o Allows for greater flexibility in object creation.

   o **Example:**

```
public Human(int age, String name) {

    this.age = age;

    this.name = name;

}
```

<mark>**Key Differences Between Default and Parameterized Constructors**</mark>

● **Default Constructor:**

   o Has no parameters.

   o Provides default values to object attributes.

   o Automatically generated by the compiler if not explicitly defined.

● **Parameterized Constructor:**

   o Accepts arguments to initialize object attributes with specific values.

o   Must be explicitly defined by the programmer.

Example: Default Constructor

```java
1   class Human {
2       private int age;
3       private String name;
4
5       public Human() {  // Default Constructor
6           System.out.println("Inside Default Constructor");
7           age = 12;
8           name = "John";
9       }
10
11      public int getAge() {
12          return age;
13      }
14
15      public String getName() {
16          return name;
17      }
18  }
19
20  public class Demo {
21      public static void main(String[] args) {
22          Human obj = new Human();  // Default Constructor is called
23          System.out.println("Name: " + obj.getName() + "\nAge: " + obj.getAge());
24      }
25  }
```

Output:

```
Output     Generated Files


Inside Default Constructor
Name: John
Age: 12
```

In this example, every time an object of the Human class is created using the default constructor, the same default values (age = 12, name = "John") are assigned to the object's attributes.

Example: Parameterized Constructor

```java
class Human {
    private int age;
    private String name;

    public Human() {  // Default Constructor
        System.out.println("Inside Default Constructor");
        age = 12;
        name = "John";
    }

    public Human(int a, String n) {  // Parameterized Constructor
        System.out.println("Inside Parameterized Constructor");
        age = a;
        name = n;
    }

    public int getAge() {
        return age;
    }

    public String getName() {
        return name;
    }
}

public class Demo {
    public static void main(String[] args) {
        Human obj = new Human();  // Default Constructor is called
        Human obj1 = new Human(18, "June");  // Parameterized Constructor is called
        System.out.println("Name: " + obj.getName() + "\nAge: " + obj.getAge());
        System.out.println("Name: " + obj1.getName() + "\nAge: " + obj1.getAge());
    }
}
```

Output:

```
Output    Generated Files

Inside Default Constructor
Inside Parameterized Constructor
Name: John
Age: 12
Name: June
Age: 18
```

Here, the default constructor assigns values for obj, while the parameterized constructor allows different values to be assigned to obj1.

Additional FAQs on Java Constructors

1. **What happens if I define a constructor with parameters and no default constructor?**

   o If you define a parameterized constructor and omit the default constructor, the compiler will not provide a default constructor. Therefore, attempting to create an object without arguments will result in a compile-time error.

2. **Can a constructor be private?**

   o Yes, a constructor can be private. This is often used in the Singleton design pattern to restrict object creation and ensure only one instance of a class exists.

3. **Can constructors be overloaded in Java?**

   o Yes, constructors can be overloaded by defining multiple constructors with different parameter lists within the same class.

4. **Why do we need a parameterized constructor?**

   o A parameterized constructor is useful when you want to create objects with different initial values, providing more control over object creation.