Different Ways to Create the Instances of Wrapper Classes in Java

Last Updated : 17 Mar, 2021

Wrapper Class a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Methods:

We can use two ways to construct the instance of the Wrapper Classes

1. Using the constructor of the wrapper class
2. Using the valueOf() method provided by the Wrapper classes
3. Using concept of AutoBoxing

Let us discuss both ways individually in detail

Method 1: Using the constructor of the wrapper class

Syntax:

```
ClassName object = new ClassName(argument);
```

Illustration:

```
Integer number = new Integer(5);
```

Method 2: Using the valueOf() method provided by the Wrapper classes

Syntax:

```
ClassName object = ClassName.valueOf(argument);
```

Illustration:

```
Integer number = Integer.valueOf(5);
```

Now the question arises what is the difference between the two methods in the creation of instances of the Wrapper classes and which method is better for constructing instances. Let us implement both of the methods to get fair play among them.

Implementation:

Example

```
// Importing input output classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main (String[] args) {

        // Creating and initializing two integer numbers
        // Value is passed as an argument to which it is initialized

        // Custom entries
        // Number 1 where N = 5
        Integer num1 = new Integer(5);
        // Number 2 where N = 5
        Integer num2 = new Integer(5);

        // Creating objects of Integer class
        // using valueOf() method

        // Again, creating and initializing two integer numbers
        // Value is passed as an argument to which it is initialized
        Integer num3 = Integer.valueOf(5);
        Integer num4 = Integer.valueOf(5);


        // Now by far, all the objects contain the same value
        // N = 5

        // Boolean will return true if numbers are equal
        // else eturning false

        // Comparing two numbers
        boolean value1 = (num1 == num2);
        boolean value2 = (num3 == num4);

        // Print and display the bool results
        System.out.println(value1);
        System.out.println(value2);
    }
}
```

Output:

```
false
true
```

Output explanation:

Note that, the instances of the classes point to the memory locations assigned in the heap and themselves do not hold the value. While we are creating objects with the help of the constructor of the wrapper class, each time a new memory is allocated in the heap and the objects point to the different memory locations. Hence, in the above example, In this case, both num1 and num2 are pointing to different memory locations, thus on the comparison, they return false.

Do note is not so in the case of the valueOf() method as the valueOf() method checks if any memory is allocated to the same value for that class in the heap. If it finds the value, then it provides the location of the previously allotted memory to the new instance and both start pointing to the same memory location in the heap. Hence, on the comparison, it returns true.

Since the wrapper class object's values are immutable just like String and thus can not be changed once allotted, it does not affect how many instances are pointing to the same memory location. Hence, in the above example, the memory was allotted to value 5 and the num3 was pointing to that memory location, but when we created one more instance num4 with the same value, it also started pointing to the same memory location as pointed by num3.

Currently, the method using a constructor to create an instance is deprecated, and therefore it is always best to use the valueOf() method. So let us move ahead a bit discussing the new concept of autoboxing.

Method 3: Using the concept of AutoBoxing

AutoBoxing is to reduce the efforts to write the valueOf() method each time we are creating instances, AutoBoxing is implemented. The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as AutoBoxing.

We were creating wrapper classes until now by using *valueOf()* method, but it seems quite lengthy when we can use AutoBoxing. In AutoBoxing, our work is done by the compiler, i.e. Java compiler in the background would perform the valueOf() operation and create the instance of it.

Instances created using autoboxing follow the process of valueOf() in the background and hence in this also, multiple instances with the same value point to the same memory location.

Illustration: In the above example, it can also be written as *Integer.valueOf(15)*) and put the reference of it in the object (i.e. a number).

```
Integer number = 15;
```

Syntax:

```
ClassName object = value;
// of primitive data type associated with the wrapper class.
```

Example:

```
// Importing input output classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main (String[] args) {

      // Creating Instances using AutoBoxing
      Integer num1 = 5;
      Integer num2 = 5;

      boolean bool = (num1 == num2);
      System.out.println(bool);
    }
}
```

Output

```
true
```

Output explanation:

Both the num1 and num2 are pointing to the same memory location in the heap as we discussed in the *valueOf()* method.