Object Class in Java

Difficulty Level : Medium Last Updated : 03 Jan, 2022

Object class is present in **java.lang** package. Every class in Java is directly or indirectly derived from the **Object** class. If a class does not extend any other class then it is a direct child class of **Object** and if extends another class then it is indirectly derived. Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of inheritance hierarchy in any Java Program.

Using Object Class Methods

There are methods in the **Object** class:

1. toString(): The toString() provides a String representation of an object and is used to convert an object to String. The default toString() method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, it is defined as:

```
// Default behavior of toString() is to print class name, then
// @, then unsigned hexadecimal representation of the hash code
// of the object

public String toString()
{
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

It is always recommended to override the **toString()** method to get our own String representation of Object. For more on override of toString() method refer – Overriding toString() in Java

> ***Note:*** *Whenever we try to print any Object reference, then internally toString() method is ca lled.*

```
Student s = new Student();
```

```
// Below two statements are equivalent
System.out.println(s);
System.out.println(s.toString());
```

2. hashCode(): For every object, JVM generates a unique number which is hashcode. It returns distinct integers for distinct objects. A common misconception about this method is that the hashCode() method returns the address of the object, which is not correct. It converts the internal address of the object to an integer by using an algorithm. The hashCode() method is **native** because in Java it is impossible to find the address of an object, so it uses native languages like C/C++ to find the address of the object.

Use of hashCode() method: It returns a hash value that is used to search objects in a collection. JVM(Java Virtual Machine) uses the hashcode method while saving objects into hashing-related data structures like HashSet, HashMap, Hashtable, etc. The main advantage of saving objects based on hash code is that searching becomes easy.

> ***Note:*** *Override of* ***hashCode()*** *method needs to be done such that for every object we gener ate a unique number. For example, for a Student class, we can return the roll no. of a stude nt from the hashCode() method as it is unique.*

```
// Java program to demonstrate working of
// hashCode() and toString()

public class Student {
    static int last_roll = 100;
    int roll_no;

    // Constructor
    Student()
    {
        roll_no = last_roll;
        last_roll++;
    }

    // Overriding hashCode()
    @Override public int hashCode() { return roll_no; }

    // Driver code
    public static void main(String args[])
    {
        Student s = new Student();

        // Below two statements are equivalent
        System.out.println(s);
        System.out.println(s.toString());
    }
}
```

Output :

```
Student@64
Student@64
```

Note that $4*16^0 + 6*16^1 = 100$

3. equals(Object obj): It compares the given object to "this" object (the object on which the method is called). It gives a generic way to compare objects for equality. It is recommended to override the **equals(Object obj)** method to get our own equality condition on Objects. For more on override of equals(Object obj) method refer – Overriding equals method in Java

> ***Note:*** *It is generally necessary to override the* ***hashCode()*** *method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states th at equal objects must have equal hash codes.*

4. getClass(): It returns the class object of "this" object and is used to get the actual runtime class of the object. It can also be used to get metadata of this class. The returned Class object is the object that is locked by static synchronized methods of the represented class. As it is final so we don't override it.

```
// Java program to demonstrate working of getClass()

public class Test {
    public static void main(String[] args)
    {
        Object obj = new String("GeeksForGeeks");
        Class c = obj.getClass();
        System.out.println("Class of Object obj is : "
                           + c.getName());
    }
}
```

Output:

```
Class of Object obj is : java.lang.String
```

> ***Note:*** *After loading a .class file, JVM will create an object of the type java.lang.Class in the Heap area. We can use this class object to get Class level information. It is widely used in Reflection*

5. finalize() method: This method is called just before an object is garbage collected. It is called the Garbage Collector on an object when the garbage collector determines that there are no more references to the object. We should override finalize() method to dispose of system resources, perform clean-up activities and minimize memory leaks. For example, before destroying Servlet objects web container, always called finalize method to perform clean-up activities of the session.

> ***Note:*** *The finalize method is called just* ***once*** *on an object even though that object is eligibl e for garbage collection multiple times.*

```
// Java program to demonstrate working of finalize()

public class Test {
    public static void main(String[] args)
    {
        Test t = new Test();
        System.out.println(t.hashCode());
        t = null;

        // calling garbage collector
        System.gc();

        System.out.println("end");
    }

    @Override protected void finalize()
    {
        System.out.println("finalize method called");
    }
}
```

Output:

```
366712642
finalize method called
end
```

6. clone(): It returns a new object that is exactly the same as this object. For clone() method refer Clone().

The remaining three methods **wait()**, **notify() notifyAll()** are related to Concurrency. Refer to Inter-thread Communication in Java for details.