Object Class in Java

Last Updated : 20 Dec, 2024

Object class in Java 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 the **Java Object class** and if it extends another class then it is indirectly derived. The Object class provides several methods such as **toString(),equals()**, **hashCode(),** and many others. Hence, the Object class acts as a root of the inheritance hierarchy in any Java Program.

Example: Here, we will use the toString() and **hashCode()** methods ***to provide a custom string representation for a class***.

```
// Java Code to demonstrate Object class
class Person {
    String n;  //name

    // Constructor
    public Person(String n) {
        this.n = n;
    }

    // Override toString() for a
    // custom string representation
    @Override
    public String toString() {
        return "Person{name:'" + n + "'}";
    }

    public static void main(String[] args) {

```

```
        Person p = new Person("Geek");

        // Custom string representation
        System.out.println(p.toString());

        // Default hash code value
        System.out.println(p.hashCode());
    }
}
```

Output

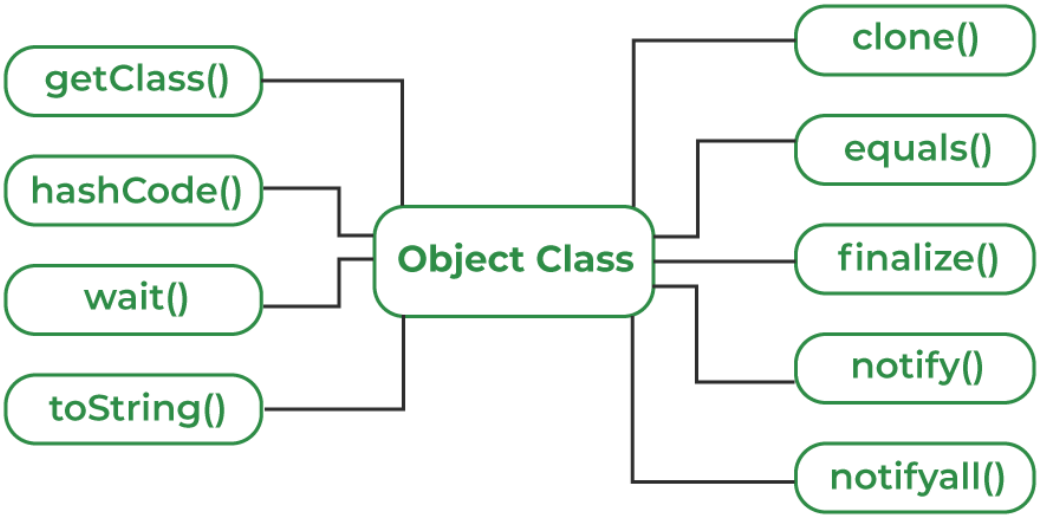
```
Person{name:'Geek'}
321001045
```

Explanation: In the above example, we override the **toString()** method to provide a **custom string representation of the Person class** and use the **hashCode()** method to display the default hash code value of the object.

Object Class Methods

The Object class provides multiple methods which are as follows:

- toString() method
- hashCode() method
- equals(Object obj) method
- finalize() method
- getClass() method
- clone() method
- wait(), notify() notifyAll() (Concurrency methods)



1. toString() Method

The **toString()** provides a String representation of an object and is used to convert an object to a 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.

Note: Whenever we try to print any Object reference, then internally toString() method is called.

Example:

```
public class Student {

    public String toString() {

        return "Student object";


    }


}
```

Explanation: The toString() method is overridden to return a custom string representation of the Student object.

2. hashCode() Method

For every object, JVM generates a unique number which is a 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 generate a unique number. For example, for a Student class, we can return the roll no. of a student from the hashCode() method as it is unique.*

Example:

```
public class Student {

    int roll;

    @Override

    public int hashCode() {

        return roll;


    }
```

```
}
```

Explanation: The **hashCode()** method is overridden to return a custom hash value based on the roll of the Student object.

3. equals(Object obj) Method

The **equals()** method compares the given object with the current object. It is recommended to override this method to define custom equality conditions.

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 that equal objects must have equal hash codes.

Example:

```
public class Student {

    int roll;



    @Override

    public boolean equals(Object o) {

        if (o instanceof Student) {

            return this.roll == ((Student) o).roll;


        }

        return false;
```

```
    }

}
```

Explanation: The **equals()** method is overridden to compare **roll** between two Student objects.

4. getClass() method

The **getClass()** method 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.

Example:

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

Output

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

Explanation: The **getClass()** method is used to print the runtime class of the “o” object.

> ***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

The finalize() 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 the 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 eligible for garbage collection multiple times.*

Example:

```
// Demonstrate working of finalize()
public class Geeks {
    public static void main(String[] args) {

        Geeks t = new Geeks();
        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");
    }
```

```
20  }
```

Output

```
1510467688
end
finalize method called
```

Explanation: The **finalize()** method is called just before the object is garbage collected.

6. clone() method

The clone() method creates and returns a new object that is a copy of the current object.

Example:

```
public class Book implements Cloneable {

    private String t;  //title


    public Book(String t) {

        this.t = t;


    }

    @Override

    public Object clone() throws CloneNotSupportedException {

        return super.clone();


    }
```

```
}
```

Explanation: The **clone()** method is overridden to return a cloned copy of the **Book object**.

7. Concurrency Methods: wait(), notify(), and notifyAll()

These methods are related to thread Communication in Java. They are used to make threads wait or notify others in concurrent programming.

Example of using all the Object Class Methods in Java

```
import java.io.*;

public class Book implements Cloneable {
    private String t;   // title
    private String a;   // author
    private int y;      // year

    public Book(String t, String a, int y)
    {
        this.t = t;
        this.a = a;
        this.y = y;
    }

    // Override the toString method
    @Override public String toString()
    {
        return t + " by " + a + " (" + y + ")";
    }

    // Override the equals method
    @Override public boolean equals(Object o)
    {
        if (o == null || !(o instanceof Book)) {
            return false;
        }
```

```
27            Book other = (Book)o;
28            return this.t.equals(other.getTitle())
29                && this.a.equals(other.getAuthor())
30                && this.y == other.getYear();
31        }
32
33        // Override the hashCode method
34        @Override public int hashCode()
35        {
36            int res = 17;
37            res = 31 * res + t.hashCode();
38            res = 31 * res + a.hashCode();
39            res = 31 * res + y;
40            return res;
41        }
42
43        // Override the clone method
44        @Override public Book clone()
45        {
46            try {
47                return (Book)super.clone();
48            }
49            catch (CloneNotSupportedException e) {
50                throw new AssertionError();
51            }
52        }
53
54        // Override the finalize method
55        @Override protected void finalize() throws Throwab
56        {
57            System.out.println("Finalizing " + this);
58        }
59
60        public String getTitle() { return t; }
61
62        public String getAuthor() { return a; }
63
64        public int getYear() { return y; }
65        public static void main(String[] args)
66        {
67            // Create a Book object and print its details
68            Book b1 = new Book(
69                "The Hitchhiker's Guide to the Galaxy",
70                "Douglas Adams", 1979);
```

```
        System.out.println(b1);

        // Create a clone of the Book object and print its
        // details
        Book b2 = b1.clone();
        System.out.println(b2);

        // Check if the two objects are equal
        System.out.println("b1 equals b2: "
                           + b1.equals(b2));

        // Get the hash code of the two objects
        System.out.println("b1 hash code: "
                           + b1.hashCode());
        System.out.println("b2 hash code: "
                           + b2.hashCode());

        // Set book1 to null to trigger garbage collection
        // and finalize method
        b1 = null;
        System.gc();
    }
}
```

Output

```
The Hitchhiker's Guide to the Galaxy by Douglas Adams (1979)
The Hitchhiker's Guide to the Galaxy by Douglas Adams (1979)
b1 equals b2: true
b1 hash code: 1840214527
b2 hash code: 1840214527
```

Explanation: The above example demonstrates the use of **toString()**, **equals()**, **hashCode()**, and **clone()** methods in the **Book class**.