29-Object Class equals toString hashcode

In Java, the Object class is the root class of the class hierarchy. Every class in Java is directly or indirectly derived from the Object class, meaning that it serves as the parent class for all other classes. If a class does not explicitly extend another class, it implicitly extends the Object class, making all methods of the Object class available to every Java class.

The Object class provides several key methods that are essential for basic object manipulation, including **toString()**, **equals()**, and **hashCode()**. Understanding these methods is crucial for working with objects in Java, especially when dealing with collections, object comparison, and debugging.

The toString() Method

The toString() method in Java is used to obtain a string representation of an object. By default, when you print an object, the toString() method is implicitly called, returning a string that includes the class name followed by the "@" symbol and the object's hashcode in hexadecimal form.
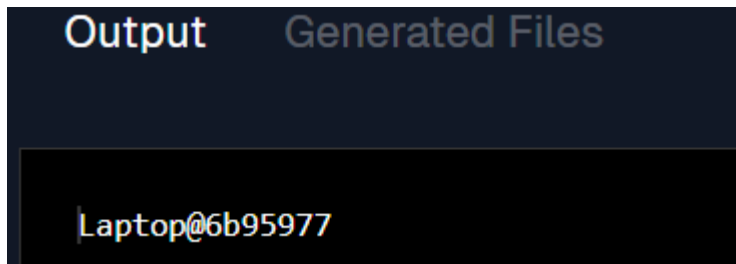
Syntax of toString() Method:

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

Default Behavior of toString() Method: By default, the toString() method returns the name of the class, followed by "@" and the hexadecimal representation of the object's hash code. This behavior is useful when you need a basic string representation of an object, but in most cases, it's more practical to override the toString() method to provide a more meaningful output.

Example:

```java
class Laptop {
    String model;
    int price;
}

public class Demo {
    public static void main(String[] args) {
        Laptop obj = new Laptop();
        obj.model = "Lenovo Yoga";
        obj.price = 1000;
        System.out.println(obj);
    }
}
```
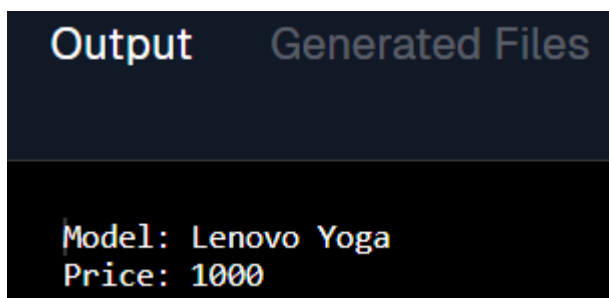
Output:

In the above example, when we try to print the object obj, it returns a string that represents the class name and hash code. This occurs because the toString() method of the Object class is invoked by default.

Overriding the toString() Method: To make the output more meaningful, you can override the toString() method in your class.

Example:

```java
1  class Laptop {
2        String model;
3        int price;
4
5        public String toString() {
6            return "Model: " + model + "\nPrice: " + price;
7        }
8  }
9
10  public class Demo {
11      public static void main(String[] args) {
12          Laptop obj = new Laptop();
13          obj.model = "Lenovo Yoga";
14          obj.price = 1000;
15          System.out.println(obj);
16      }
17  }
```

Output:

Here, the toString() method is overridden to provide a more meaningful representation of the Laptop object.

The hashCode() method returns a hash code value (an integer) for the object. This hash code is primarily used in hashing-based collections and algorithms such as HashMap, HashSet, and Hashtable.

Syntax of hashCode() Method:

```
1  public int hashCode() {
2       // Returns the hash code value for the object.
3  }
4
```

When you override the equals() method in a class, it is also a good practice to override the hashCode() method to ensure that objects that are considered equal have the same hash code.

The equals() Method

The equals() method is used to compare two objects for equality. The default implementation of equals() provided by the Object class checks whether two references point to the same object in memory.

Syntax of equals() Method:

```
1  public boolean equals(Object obj) {
2       // Compares this object with the specified object for equality.
3  }
4
```

Example:

```
1  class laptop{
2  String model;
3  int price;
4
5  public String toString(){
6    return "model : "+model+ "price :"+ price;
7  }
8  public boolean equals(laptop that)
9  {
10      return  (this.model.equals(that.model) && this.price == (that.price));
11  }
12  }
13  public class Demo{
14  public static void main(String[] args){
15    laptop obj=new laptop();
16  laptop obj1=new laptop();
17
18  obj.model="Lenovo yoga";
19  obj.price=1000;
20  obj1.model="Lenovo yoga";
21  obj1.price=1000;
22
23  System.out.println(obj == obj1);
24  System.out.println(obj.equals(obj1));
25  }
26  }
```

Output:

In this example, the equals() method is overridden to compare the content of two Laptop objects rather than their memory references.

- **Improved Debugging:** Overriding toString() allows you to get a more meaningful output when printing objects, making it easier to debug.

- **Accurate Object Comparison:** Overriding equals() ensures that object comparisons are based on actual content rather than memory references.

- **Proper Hashing:** Overriding hashCode() ensures that objects that are considered equal have the same hash code, which is critical for the correct functioning of hash-based collections.

The Object class in Java provides fundamental methods such as toString(), equals(), and hashCode(), which are crucial for object manipulation. By understanding and correctly overriding these methods, you can enhance the functionality and readability of your Java programs.

1. **Can we override the toString() method in every class?**

   o Yes, you can override the toString() method in any class to provide a custom string representation of your objects.

2. **Why is it important to override the hashCode() method when equals() is overridden?**

   o It's important because objects that are considered equal by the equals() method must have the same hash code to maintain the contract between equals() and hashCode().

3. **What happens if we do not override the equals() method?**

   o If you do not override the equals() method, the default implementation from the Object class will be used, which compares memory references rather than content.

4. **Is it mandatory to override toString() in every class?**

   o No, it's not mandatory, but it's often useful for debugging and logging purposes.

5. **Can two different objects have the same hash code?**

   o Yes, different objects can have the same hash code due to hash code collisions, but good hash code implementations minimize the chances of collisions.