# What is the contract between equals() and hashCode() methods in Java?n

[Java](https://www.tutorialspoint.com/articles/category/Java)[Object Oriented Programming](https://www.tutorialspoint.com/articles/category/Object-Oriented-Programming)[Programming](https://www.tutorialspoint.com/articles/category/Programming)

Every Java object has two very important methods **equals()** and **hashCode()**and these methods are designed to be overridden according to their specific **general contract**. An **Object** class is the parent class of every class, the default implementation of these two methods is already present in each class. However, we can override these methods based on the requirement.

## **hashCode() Method**

public int hashCode()

This method returns an **integer** value, which is referred to as the hash code value of an object. Every Object, at the time of creation assigned with a unique 32-bit, signed int value. This value is the hash code value of that object.

## **General contract associated with hashCode() method**

* The**hashCode()** method should return the same integer value for the same object for each calling of this method unless the value stored in the object is modified.
* If two objects are equal(according to **equals()** method) then the **hashCode()** method should return the same integer value for both the objects.
* But, it is not necessary that the**hashCode()** method will return the distinct result for the objects that are not equal (according to **equals()** method).

## **equals() Method**

public boolean equals(Object obj)

The **equals()** method of Object class checks the equality of the objects and accordingly it **returns** **true** or **false**. The default implementation, as provided by Object class, checks the equality of the objects on the basis if both references refer to the same object. It does not check the value or state of the objects. But we can override this method to provide own implementation to compare the state or value of the objects.

## **General contract associated with equals() method**

For any non-null reference variables a, b and c

* **a.equals(a)** should always return **true**.
* **a.equals(b)** should return **true** if and only if **b.equals(a)** returns true.
* If **a.equals(b)** returns **true** and **b.equals(c)** returns **true** then **a.equals(c)** should return **true**.
* Multiple calling of **a.equals(b)** should consistently return **true** or consistently return **false** If the value of the object is not modified for either object.
* **a.equals(null)** should return **false**.

So it is necessary to override the **hashCode()**method of Object class if we are overriding the **equals()** method.

## **Example**

[Live Demo](http://tpcg.io/eLBk5x)

public class MainClass {

   public static void main(String[] args) {

      MainClass mainClass = new MainClass();

      TestClass obj1 = new TestClass(1);

      TestClass obj2 = new TestClass(1);

      mainClass.test1(obj1, obj2);

      TestClass obj3 = new TestClass(1);

      TestClass obj4 = new TestClass(2);

      mainClass.test2(obj3, obj4);

   }

   public void test1(TestClass obj1, TestClass obj2) {

      if (obj1.equals(obj2)) {

         System.out.println("Object One and Object Two are equal");

      }

      else {

         System.out.println("Object One and Object Two are not equal");

      }

   }

   public void test2(TestClass obj3, TestClass obj4) {

      if (obj3.equals(obj4)) {

         System.out.println("Object Three and Object Four are equal");

      }

      else {

         System.out.println("Object Three and Object Four are not equal");

      }

   }

}

class TestClass {

   private int val;

   TestClass(int val) {

      this.val = val;

   }

   public int getValue() {

      return val;

   }

   @Override

   public boolean equals(Object o) {

      // null check

      if (o == null) {

         return false;

      }

      // this instance check

      if (this == o) {

         return true;

      }

      // instanceof Check and actual value check

      if ((o instanceof TestClass) && (((TestClass) o).getValue() == this.val)) {

         return true;

      }

       else {

         return false;

      }

   }

   @Override

   public int hashCode() {

      int result = 0;

      result = (int) (val / 11);

      return result;

   }

}

# Java program to count the occurrence of each character in a string using Hashmap

**import** java.io.\*;

**import** java.util.\*;

**class** OccurrenceOfCharInString {

**static** **void** characterCount(String inputString)

    {

        // Creating a HashMap containing char

        // as a key and occurrences as  a value

        HashMap<Character, Integer> charCountMap

            = **new** HashMap<Character, Integer>();

        // Converting given string to char array

**char**[] strArray = inputString.toCharArray();

        // checking each char of strArray

**for** (**char** c : strArray) {

**if** (charCountMap.containsKey(c)) {

                // If char is present in charCountMap,

                // incrementing it's count by 1

                charCountMap.put(c, charCountMap.get(c) + 1);

            }

**else** {

                // If char is not present in charCountMap,

                // putting this char to charCountMap with 1 as it's value

                charCountMap.put(c, 1);

            }

        }

        // Printing the charCountMap

**for** (Map.Entry entry : charCountMap.entrySet()) {

            System.out.println(entry.getKey() + " " + entry.getValue());

        }

    }

    // Driver Code

**public** **static** **void** main(String[] args)

    {

        String str = "Ajit";

        characterCount(str);

    }

}

Java program to count the occurrence of each character in a string using Hashmap   
  (Example:- String val="Brillio"; --->> <B,1>,<r,1>,<i,2>,<l,2>,<o,1>  
- transient keyword in java  
- Differnce ways of creating threads in Java. Difference between Runnable vs Thread in Java  
- Synchronization in Java(Purpose of synchronized block in java. How its useful in multithreading?)  
- Singleton patten usage in Mutlithreading env.  
- Marker Interface  
- what is encapsulation?   
- Questions related to Run Time Polymorphism.  
- possible to override static block in java ? and Why?  
- Any agreement between hash code and equals method when you are using with HashMap or HashTable?  
- Dependency Injection and how to achieve? its Type.  
- What is IoC Container?  
- Http Request flow in Spring Boot?  
- what is @RequestController, @ComponentScan, @Service?

SQL Interview Question:-

- Stored procedure in sql.  
- Differnce between between having and group by in Sql.  
- Queries related to the Group By and Having.  
- What is a Subquery? What are its types with examples?