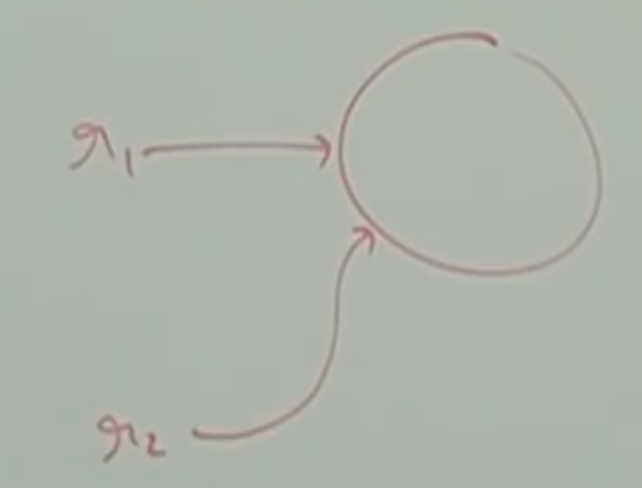
**java.lang package – Part-12**

* **Relation between == operator and .equals() method:**

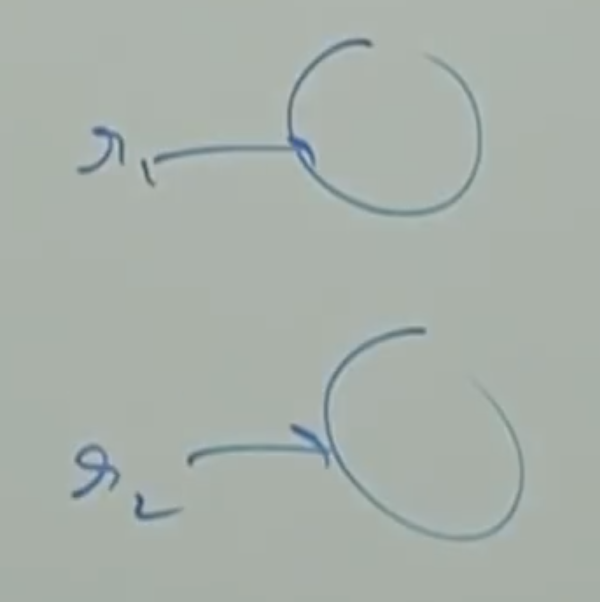
1. If two objects are equals by == operator then these objects are always equal by .equals method.

That is, if r1 == r2 is true, then r1.equals(r2) is always true



1. If two objects are not equal by == operator then we can’t conclude anything about .equals() method. It may returns true or false.

That is, if r1 == r2 is false the r1.equals(r2) may returns true or false.



1. If two objects are equals by .equals() method, then we can’t conclude anything about == operator. It may returns true or false.

That is r1.equals(r2) is true, then we can’t conclude anything about r1 == r2. It may returns true or false.

1. If two objects are not equal by .equals() method, then these objects are always not equal by == operator.

That is, if r1.equals(r2) is false, then r1 == r2 is always false.

* **Differences between == operator and .equals() method:**

To use == operator, compulsory there should be some relation between argument types. (either child to parent or parent to child or same type). Otherwise we will get compile time error saying

Incomparable types: java.lang.String and java.lang.StringBuffer.

If there is no relation between argument types then .equals() method won’t rise any compile time or runtime errors. Simply it returns false.

Example:

class Test{

public static void main(String[] args){

String s1 = new String(“durga”);

String s2 = new String(“durga”);

StringBuffer sb1 = new StringBuffer(“durga”);

StringBuffer sb2 = new StringBuffer(“durga”);

System.out.println(s1==s2); // false

System.out.println(s1.equals(s2)); // true

System.out.println(sb1==sb2); //false

System.out.println(sb1.equals(sb2)); //false

System.out.println(s1==sb1); //CE

System.out.println(s1.eqauls(sb1));// false

}

}

CE: incomparable types: java.lang.String and java.lang.StringBuffer

|  |  |  |
| --- | --- | --- |
| S.No | == Operator | .equals() |
| 1 | It is an operator in Java, applicable for both primitives and object types. | It is a method applicable only for object types but not for primitives |
| 2 | In the case of object references == operator meant for reference comparison (address comparison) | By default .equals() method present in object also meant for reference comparison. |
| 3 | We can’t override == operator for content comparison. | We can override .equals() method for content comparison. |
| 4 | To use == operator compulsory there should some relation between argument types (either child to parent or parent to child or same type).  Otherwise we will get compile time error saying:  Incomparable types:  java.lang.String and java.lang.StringBuffer | If there is no relation between argument types then .equals() method won’t rise any compile or runtime errors and simply returns false. |

Answer in one line:

In general we can use == operator for reference comparison and .equals() method for content comparison.

Note:

For any object reference r,

r == null

r.equals(null) always returns false.

Example:

Thread t = new Thread();

System.out.println(t == null);

System.out.println(t.equals(null);

Note:

Hashing related data structures follow the following fundamental rule.

Two equivalent objects should be placed in same bucket but all objects present in the same bucket need not be equal.

* **Contract between equals() method and hashCode() method:**

If two objects are equal by .equals() method then their hashcode must be equal. That is, two equivalent objects should have same hashcode.

That is, if r1.equals(r2) is true, then r1.hashCode() == r2.hashCode() is always true.

Object class .equals() method and hashCode() method follows above contract. Hence, whenever we are overriding .equals() method compulsory we should override hashCode() method to satisfy above contract. (That is two equivalent objects should have same hashCode).

1. If two objects are not equal by .equals() method, then there is no restriction on hashcodes. May be equal or may not be equal.
2. If hashCodes of two objects are equal then we can’t conclude anything about .equals() method. It may returns true or false.
3. If hascodes of two objects are not equal then these objects are always not equal by .equals() method.

Note\*\*\*

To satisfy contract between equals() and hashCode() methods, whenever we are overriding .equals() method compulsory we have to override hashCode() method.

Otherwise we won’t get any compile time or runtime errors but it is not a good programming practice.

In String class, .equals() method is overridden for content comparison and hence hashCode() method is also overridden. To generate hashCode based on content.

Example:

String s1 = new String(“durga”);

String s2 = new String(“durga”);

System.out.println(s1.equals(s2)); // true

System.out.println(s1.hashCode()); // 95950491

System.out.println(s2.hashCode()); // 95950491

In StringBuffer .equals() method is not overridden for content comparison and hence hashCode() method is also not overridden.

Example:

StringBuffer sb1 = new StringBuffer(“durga”);

StringBuffer sb2 = new StringBuffer(“durga”);

System.out.println(sb1.equals(sb2)); // false

System.out.println(sb1.hashCode()); // 19621457

System.out.println(sb2.hashCode()); // 4872 882

* **Consider the following person class:**

class Person{

public boolean equals(Object obj){

if(obj instanceof Person){

Person p = (Person) obj;

if(name.equals(p.name) && age == p.age){

return true;

} else{

return false;

}

}

return false;

}

}

Which of the following hashCode() methods are appropriate for Person class?

1. public int hashCode(){

return 100;

}

1. public int hashCode(){

return age+ssno;

}

1. public int hashCode(){

return name.hashCode()+age;

}

1. No restrictions

Answer: 3

Note:

Based on which parameters we override .equals() method, it is highly recommended to use same parameters while overriding hashCode() method also.

Note: In all collection classes, in all wrapper classes and in String class equals() method is overridden for content comparison. Hence, it is highly recommended to override equals() in our class also for content comparison.