**java.lang**

**The Object class**

Object is the root or super class of the class hierarchy. Every pre – defined and user defined classes are sub classes from object.

**Why object is super class for all Java objects?**

Because of below two reasons Object class is developed as super class for every class:

1. **Reusability:** Every object has 11 common behaviors. These behaviors must be implemented by every class developer. So to reduce burden on developer, SUN has developed a class called Object by implementing all these 11 behaviors with 11 methods. If this logic is not satisfying to sub class requirement, then sub class should override it.
2. **To achieve run –** **time polymorphism**: So that we can write single method to receive and send any type of class object as argument and return type.

**What are common functionalities of every class object?**

Every object contains 11 common operations:

* 1. **Comparing two objects**

public boolean equals(Object obj)

* 1. **Retrieving hashCode**

public native int hashCode()

* 1. **Printing object information in String format for printing purpose**

public String toString()

* 1. **Retrieving the run – time class object reference**

public final native Class getClass()

* 1. **Cloning Object**

protected native Object clone() throws CloneNotSupportedException

* 1. **Garbage Collection**

Protected void finalize() throws Throwable

* 1. **Releasing object lock and sending thread to waiting state**

public final void wait() throws InterruptedException

public final native void wait(long millis) throws InterruptedException

public final void wait(long millis, int nanos) throws InterruptedException

* 1. **Notify about object lock availability to waiting threads**

public final native void notify()

public final native void notifyAll()

All above methods are implemented with generic logic that is common for all subclass objects. So that developer can avoid implementing these operation in every class, also SUN can ensure that all these operations are overridden by developers with the same method prototype. This feature provides run – time polymorphism.

**What are the methods we can override in subclass from Object class?**

We can override below five methods as they are not final methods.

1. equals()
2. hashCode()
3. toString()
4. clone()
5. finalize()

**When should we override equals and hashCode methods in subclass?**

To use subclass as a key to add entry to Map objects and also to add its objects as element to Set collection, that class **must override** equals() and hashCode() methods.

If these two methods are not overridden no CE or RE, but those objects are not found for retrieving or removing because these methods are executed from java.lang Object class.

**When should we override toString() method?**

To print object data we must override, toString() method by returning object’s non – static variable value (state).

**What is difference between == operator and equals() method?**

|  |  |
| --- | --- |
| == | equals() |
| It always compares object with references. | It compares objects either with state or reference based on its implementation.   * In object class it compares objects with reference. * In subclass it compares objects with state. |
| We cannot compare incompatible objects, compiler throws CE. | We can compare incompatible objects, it returns false. |
| We can also use it for comparing primitive values and two null’s directly. | We cannot use it to compare primitive values and two null’s directly, it leads to CE. |

**Hashcode**

Hashcode is an identity of an object. It is used to differentiate one object from another object. Every object has its own unique address. As per javadocs defined “hashcode is typically implemented by converting the internal address of the object into an integer number”.

**When hashcode of an object is used?**

It is used by hashtable data structure to store, remove, retrieve, and search the object is Set and Map collection object.

**How can we retrieve hashcode of an object?**

By calling hashCode() method, it is defined in java.lang.Object class.

It is a native method and its default implementation is returning the reference of the object in integer form. The prototype of this method is:

**public native hashCode()**

**Can we generate custom hashCode for our object?**

Yes, we can also generate custome hashcode by using its state.

**So, in how many ways we can generate hashcode of an object?**

In two ways,

1. By using its reference  
   It is the default implementation of JVM and that hashcode number is returned through hashcode method of java.lang.Object.
2. By using its state  
   It is the overriding implementation developed by subclass developer by overriding hashCode() method in sibclass.

**If we change state of the object, is its hashCode changed?**

It depends on hashCode() method implementation:

* If hashCode is generated by using object reference, then its hashcode is not changed when object state is changed.
* If hashCode() method is overridden in subclass for generating hashCode by using object’s state then object hashcode is changed when its state is changed.

**Can two objects have same hashcode?**

Yes, there is possibility. It depends on hashcode() method implementation.

* JVM always generates different hashcode for objects, because objects always have different references.
* But developer overriding hashcode() method may give same hashcode for multiple objects because objects of same class can have same state.

**If equals() method is overridden then hashCode() method must be overridden with below contract:**

* If equals() method returns true by comparing two objects, then hashCode of both objects must be same.
* If it returns false, the hashcode of both objects may or may not be same.

**Program to demonstrate the above fact**

class Example

{}

class Student

{

int rollNo;

String name;

Student(int rollNo, String name)

{

this.rollNo = rollNo;

this.name = name;

}

public boolean equals(Object obj)

{

if (this == obj)

{

return true;

}

else

{

if (obj instanceof Student)

{

Student s = (Student)obj;

return this.rollNo == s.rollNo &&

this.name.equals(s.name);

}

else

return false;

}

}

}

class HashCodeDemo1

{

public static void main(String[] args)

{

Example e1 = new Example();

Example e2 = new Example();

System.out.println(e1.hashCode());

System.out.println(e2.hashCode());

/\* Checking contract between equals() and hashCode() method

If equals() mtethod returns true, then hashCode() must also returns true \*/

System.out.println(e1 == e2); // false

System.out.println(e1.equals(e2)); // false

System.out.println(e1.hashCode() == e2.hashCode()); // false

Example e3 = e2;

System.out.println(e2 == e3); // true

System.out.println(e2.equals(e3)); // true

System.out.println(e2.hashCode() == e3.hashCode()); // true

/\*In the below case the contract will failed because equals()

in Student class compares state of the object \*/

Student s1 = new Student(1,"Ashu");

Student s2 = new Student(1,"Ashu");

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

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

}

}

Output

31168322

17225372

false

false

false

true

true

true

true

false