*Object Class*

* As object class acts as a root (or) parent (or) super for all java classes, by default its methods are available to every java class.
* If our class doesn't extends any other class then it is the direct child class of object If our class extends any other class then it is the indirect child class of Object
* For writing any java program the most commonly required classes and interfaces are encapsulated in the separate package which is nothing but java.lang package.
* It is not required to import java.lang package in our program because it is available by default to every java program.

The following is the list of all methods present in java.lang Object class :

1. toString();

2. hashCode();

3. equals();

4. clone();

5. getClass();

6. finalize();

7. wait();

8. wait();

9. wait();

10. notify();

11. notifyAll();

There are different methods of object class are as follows.

## *Public final Class getClass()*

This class is used to get the metadata of class ff.i.e. returns runtime class definition of an object

**public class** Example {

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

Example example =**new** Example (); System.***out***.println(example.getClass().getName()); System.***out***.println(example.getClass().getSimpleName());

}

}

## *Public int hashCode()*

For every object unique number is generated by JVM called as hashcode. It is based on address of the object but it doesn't mean hashCode represents address of the object. Jvm will be using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

## Note-

1. If two objects are equal, their hashcode will be same.
2. If two object hashcode are same, you cannot guaranty that objects are equal.
3. Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number as hashcode for every object

Example

**public class** Test {

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

Test test1 = **new** Test ();

Test test2 = **new** Test ();

System.***out***.println(test1.hashCode()); System.***out***.println(test2.hashCode());

}

}

## *Public Boolean equals (Object obj)-*

It compares the given object to this object.

If our class doesn't contain. equals () method then object class. equals () method will be executed which is always meant for reference comparison [address comparison]. i.e., if two references pointing to the same object then only. equals () method returns true.

Example-

**public class** Employee {

**int** empId;

String empName;

**public static void** main(String[] args) { Employee emp1 = **new** Employee(); emp1.empId = 1;

emp1.empName = "Ashok";

Employee emp2 = **new** Employee(); emp2.empId = 2;

emp2.empName = "Sachin";

System.***out***.println(emp1.equals(emp2));

}

}

Output- false

## *protected Object clone () throws CloneNotSupportedException-*

It creates and returns the exact copy (clone) of this object.

The main objective of cloning is to maintain backup purposes. (i.e., if something goes wrong we can recover the situation by using backup copy.)

Example-

**public class** Example **implements** Cloneable {

**int** x;

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

CloneNotSupportedException {

Example example1 = **new** Example(); example1.x = 50;

System.***out***.println("First Object data is>>" + example1.x);

Object example2 = example1.clone();

System.***out***.println("Second Object data is>>" + example2);

}

}

## *public String toString()*

It returns the string representation of this object.

Whenever we are try to print any object reference internally toString() method will be executed.

**public class** Example {

**int** x;

**public** String toString() {

**return** "Example [x=" + x + "]";

}

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

Example example1 = **new E**xample();

example1.x = 50;

System.***out***.println("First Object data is>>" + example1);

}

}

## *public final void notify()*

It wakes up single thread, waiting on this object's monitor.

## *public final void notifyAll()*

It wakes up all the threads, waiting on this object's monitor.

## *public final void wait(long timeout)throws InterruptedException()*

It causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).

## *public final void wait(long timeout,int nanos)throws InterruptedException*-

It causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).

## *public final void wait()throws InterruptedException*

It causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

## *protected void finalize()throws Throwable*

It is invoked by the garbage collector before object will be destroyed to perform clean up activity.