**String –** String Object are immutable.

**StringBuffer –** StringBuffer Object are mutable.

**== Operator –** it is always meant for reference compression.

**.equals operator –** It is meant for content compression. But In StringBuffer it is meant for reference compression and even in Object class.

**Note:**

* SCP concept is only available for String not for String buffer because String is used a lot in Java rather than String Buffer.
* String objects are immutable but StringBuffer are mutable because Strings are stored in with one object and many references whereas String buffer object stored with each object with unique references. So if string object is modified then all other references are modified.
* All Wrapper class objects are immutable.

**Java.lang package**

**java.lang.Object**

the most common methods required for every class are defined in separate class is nothing but object class. Every class in java is child class of object either directly or indirectly. So that object class method is default available to every java class.

Object class contains 12 methods… but one method which is registerNatives is internally required for object class but not available for child class. Hence we don’t consider it.

* Public String toString()
* Public native int hashCode()
* Public Boolean equals(Object o)
* Protected native Object clone() throws CloneNotSupportedException
* Protected void finalize() throws Throwable
* Public final class getClass()
* Public final void wait() throws InterruptedException
* Public final native void wait(long ms) throws InterruptedException
* Public final void wait(long ms, int ns) throws InterruptedException
* Public native final void notify()
* Public native final void notifyAll()

**toString –**

we can use toString method to get string representation of object. Whenever we are trying to print object reference internally toString method will be called.   
Student s = new Student();  
s.o.p(s); == s.o.p(s.toString());

**public String toString()**

**{**

**return getClass().getName()+"@"+Integer.toHexString(hashCode());**

**}**

**hashCode –**

For every object unique number generated by JVM is nothing but hashCode. Hashcode won’t represent address of object. JVM will use Hashcode by saving object into hashing related data structures like hashTable , hashMap, hashSet, etc.

The main advantage of saving object based on hashCode is search operation will become easily. Most powerful search algorithm up to today is hashing.

If you are giving chance to object class hashcode method. It will generate hashcode based on address of object. It doesn’t mean hashcode represent address of object. Based on our requirement we can overwrite hashcode method in our class to generate our own hashcode.