**Common methods in Object class**

1. Equals() => comparing objects
2. HashCode() => getting object reference
3. toString() => to print object information
4. clone() => to copy the object
5. finalize()=> to close all the open resources/objects
6. wait(),wait(time)=> to keep treads on waiting state by realising the locks
7. notify(), notifyAll()=> to notify all waiting threads about object lock availability
8. getClass()

**Methods we can override is**

1. Equals()
2. HashCode()
3. toString()
4. clone()
5. finalize()

**Equals() Method**

* Present in Object class
* By default it compare two objects reference wise as internally its using “==” operator. For comparing content wise or state wise, we should override it in subclass.

**HashCode() method**

1. Numeric representation of object reference
2. Hashcode is used by **hashtable to store, retrieve, remove and search elements in Set and Map collection.**

**Ways for Hashcode generation**

1. By using object reference: it will be JVM implemented hashCode() of Object class.
2. By using state: by overriding hashCode() in subclass. Here we use state in hashCode()

JVM implemented hashcode we can get by

1. Calling System.identityHashCode()
2. By calling super.hashCode() in override hashCode() of subclass.

* If we change object state, then two object may have
  + **Same hashCode**: if we override hashCode() in subclass by using state of object.
  + **Different hashCode:** if we are using hashCode() of Object class.
* Two objects can have
  + **Same hashcode:** if we are overriding in subclass
  + **Different hashcode:** JVM always generates different hashCode for object

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

* If equals() method is overridden, the hashCode() must be override with below condition
  + equals() return true by comparing 2 objects then hash code of both objects must be same because in equals , it uses == internally, so it compares objects.
  + If equals return false, then hash code of both object may or may not be same.

**Clone()**

Cloning is creating duplicate object from the current object state.

Use Object class clone() to clone it.

For cloning an object, that class should implement Cloneable Interface

After cloning objects, original obj and duplicate obj both are having different hash code and reference as they are newly created and they have same data

Modifying one object will not affect other one.

**Rules for calling clone method**

1. Clone() should be called on class object only inside that class because it is protected. If we call in other classes or subclass, it shows CE. To call from other class, we must override clone() with public modifier.
2. Cast the output of clone method into current object as clone() return class of Object type
3. Clone() methods calling method should handle ***CloneNotSupportedException*** either by try/catch or by throws statement.
4. The object on which we are calling clone(), that should implement Cloneable interface otherwise we will get ***CNSE***

How can we call clone method from user classes of Cloneable class

**public** **class** Test {

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

Employee t1 = **new** Employee();

Employee t2 = (Employee) t1.clone();

System.***out***.println(t2.city);

}

}

**class** Employee **implements** Cloneable {

**int** rollNo = 12;

String city = "Morshi";

@Override

**public** Object clone() **throws** CloneNotSupportedException {

// **TODO** Auto-generated method stub

**return** **super**.clone();

}

}

**Cloning with HAS-A relationship**

If classes are in composition, so after cloning, it will not clone internal objects

**public** **class** Test {

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

Employee t1 = **new** Employee();

Employee t2 = (Employee) t1.clone();

System.***out***.println(t2.b.dd);

System.***out***.println(t1.b==t2.b); //true

}

}

**class** B {**int** dd = 90; String surname = "Wadatkar";}

**class** Employee **implements** Cloneable {

**int** rollNo = 12;

String city = "Morshi";

B b=**new** B();

@Override

**public** Object clone() **throws** CloneNotSupportedException {

// **TODO** Auto-generated method stub

**return** **super**.clone();

}

}

**Cloning with IS-A relation ship**

**Type of cloning:**

1. **Shallow cloning (default cloning)**
2. **Deep cloning**
3. **Shallow cloning (default cloning)**

* Object will be copied without contained objects.
* So here, if composition relation is there then internal cloned object will point to same internal object.

1. **Deep cloning**

Object will be copied with all its internal objects also.

Copies all objects from top to bottom recursively.