**Java core concepts - Clone**

**Clone:**

* Using Clonable interface we can achieve this.
* If the class has **only primitive data type members** then a completely new copy of the object will be created and the reference to the **new object copy will be returned**.
* If the class contains **members of any class type** then only the object references to those members are copied and hence the **member references in both the original object as well as the cloned object refer to the same object**. (Same reference will be there)
* To avoid clone we can throw ClonableException

**Type:**

**Shallow cloning** - Default java implementation (it won’t clone object in the class only primitive types)

**Deep cloning** – Forcedly clone the objects in the class

* No need to separately copy primitives.
* All the member classes in original class should support cloning and in clone method of original class in context should call super.clone() on all member classes.
* If any member class does not support cloning then in clone method, one must create a new instance of that member class and copy all its attributes one by one to new member class object. This new member class object will be set in cloned object.

**Best practices**

When you don’t know whether you can call the clone() method of a particular class as you are not sure if it is implemented in that class, you can check with checking if the class is instance of “Cloneable” interface as below.

if(obj1 instanceof Cloneable){

obj2 = obj1.clone();

}

//Dont do this. Cloneabe dont have any methods

obj2 = (Cloneable)obj1.clone();

No constructor is called on the object being cloned. As a result, it is your responsibility, to make sure all the members have been properly set. Also, if you are keeping track of number of objects in system by counting the invocation of constructors, you got a new additional place to increment the counter.

I hope that this post has been a refresher for you. I will be posting more about cloning in java. So, keep visiting.