Java Clone Implementation

When implementing Cloneable interface, the compiler understands that the class can be cloned and modifies the behavior of the Clone method in Object Class which is

Protected native Object clone() in Super Class.

Clone method creates an object of class without calling the constructor of the class.

Following are true for the clone method –

x.clone() != x

x.clone.getClass() == x.getClass()

x.clone.equals(x) will be true

Problems with Clone method are :-

1 ) No constructor is called while creating a object when cloning the object.

2 ) Therefore, **if you override the clone method in a nonfinal class, you should return an object obtained by invoking super.clone.** If all of a class's superclasses obey this rule, then invoking super.clone will eventually invoke Object's clone method, creating an instance of the right class. This mechanism is vaguely similar to automatic constructor chaining, except that it isn't enforced.

**In practice, a class that implements Cloneable is expected to provide a properly**

**functioning public clone method.**

It is not, in general, possible to do so unless all of the class's superclasses provide a well-behaved clone implementation, whether public or protected.

This object will be, from the standpoint of each superclass, a fully functional clone of the original object The fields declared in your class (if any) will have values identical to those of the object being cloned. If every field contains a primitive value or a reference to an immutable object, the returned object may be exactly what you need, in which case no further processing is necessary.

If its clone method merely returns super.clone(), the resulting Stack instance will have the correct value in its size field, but its elements field will refer to the same array as the original Stack instance.

Modifying the original will destroy the invariants in the clone and vice versa.

This situation could never occur as a result of calling the sole constructor in the Stack class. **In effect, the clone method functions as another constructor; you must ensure that it does no harm to the original object and that it properly establishes invariants on the clone.**

**public class HashTable implements Cloneable { private Entry[] buckets = ...;**

**}**

**}**

**private static class Entry { Object key;**

**Object value; Entry next;**

**Entry(Object key, Object value, Entry next) {**

* **}**
* **this.key this.value this.next**
* **= key; = value; = next;**
* **copy the linked list headed by this Entry**
* **// Recursively**
* **Entry deepCopy() { return new Entry(key, value,**
* **}**
* **next == null ? null : next.deepCopy());**
* **public Object clone() throws CloneNotSupportedException { HashTable result = (HashTable) super.clone(); result.buckets = new Entry[buckets.length]; for (int i = 0; i < buckets.length; i++)**
* **if (buckets[i] != null) result.buckets[i] = (Entry)**
* **buckets[i].deepCopy(); return result;**

**} ... // Remainder omitted**

While this technique is cute and works fine if the buckets aren't too long, it is not a good way to clone a linked list because it consumes one stack frame for each element in the list. If the list is long, this could easily cause a stack overflow.

To prevent this from happening, you can replace the recursion in deepCopy with iteration:

* Entry deepCopy() { Entry result =
* new Entry(key, value, next);
* result; p.next != null; p = p.next) Entry(p.next.key, p.next.value, p.next.next);
* }
* for (Entry p = p.next = new

return result;

A final approach to cloning complex objects is to call super.clone, set all of the fields in the resulting object to their virgin state, and then call higher-level methods to regenerate the state of the object. In the case of our Hashtable example, the buckets field would be initialized to a new bucket array, and the put(key, value) method (not shown) would be invoked for each key-value mapping in the hash table being cloned. This approach typically yields a simple, reasonably elegant clone method that doesn't run quite as fast as one that directly manipulates the innards of the object and its clone.