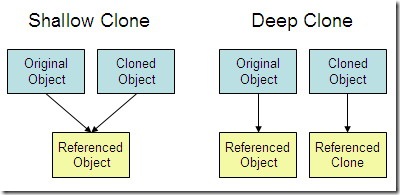
Shallow Copy VS Deep Copy in Java – 2022

**Shallow Copy Vs Deep Copy**



**What is Shallow Copy?**

Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object. If any of the fields of the object are references to other objects, just the reference addresses are copied i.e., only the memory address is copied.

Here is an example of Shallow Copy implementation

**public class** Student **implements** Cloneable {  
 *//Contained object* **private** Subject **subj**;  
 **private** String **name**;  
  
 **public** Student(String s, String sub) {  
 **name** = s;  
 **subj** = **new** Subject(sub);  
 }  
  
 **get() /set() methods**

**public** Object clone() {  
 *//shallow copy* **try** {  
 **return super**.clone();  
 } **catch** (CloneNotSupportedException e) {  
 **return null**;  
 }  
 }  
  
 @Override  
 **public** String toString() {  
 **return "Student{"** +  
 **"name='"** + **name** + **'\''** +  
 **", subj="** + **subj** +  
 **'}'**;  
 }  
}

**class** Subject {  
 **private** String **name**;  
  
 **public** Subject(String s) {  
 **name** = s;  
 }  
  
 **get() /set() methods**

@Override

**public** String toString() {  
 **return "Subject{"** +  
 **"name='"** + **name** + **'\''** +  
 **'}'**;  
 }  
}

**public class** ShallowCopyTest {  
 **public static void** show(Student as, Student cs) {  
 System.***out***.println(**"--------- Inside show() ---------"**);  
 System.***out***.println(**"Original Object : "** + as);  
 System.***out***.println(**"Cloned Object : "** + cs);  
 System.***out***.println(**"---- After Method Modification -----"**);  
  
 as.getSubj().setName(**"Triology"**);  
 as.setName(**"Trio"**);  
 cs.setName(**"Music"**);  
 cs.getSubj().setName(**"Musicology"**);  
  
 System.***out***.println(**"Original Object : "** + as);  
 System.***out***.println(**"Cloned Object : "** + cs);  
 }

**public static void** main(String[] args) {  
 *//Original Object* Student actualStudent = **new** Student(**"John"**, **"Chemistry"**);  
 System.***out***.println(**"Original Object: "** + actualStudent);  
  
 *//Clone Object* Student clonedStudent = (Student) actualStudent.clone();  
 System.***out***.println(**"Cloned Object: "** + clonedStudent);  
  
 actualStudent.setName(**"Dan"**);  
 actualStudent.getSubj().setName(**"Physics"**);  
  
 *show*(actualStudent,clonedStudent);  
  
 System.***out***.println(**"-------- After Original Object Changed --------"**);  
 System.***out***.println(**"Original Object : "** + actualStudent);  
 System.***out***.println(**"Cloned Object : "** + clonedStudent);  
  
 System.***out***.println(**"-------- After Cloned Object Changed --------"**);  
 clonedStudent.setName(**"Smith Clone"**);  
 clonedStudent.getSubj().setName(**"Biology"**);  
  
 System.***out***.println(**"Original Object : "** + actualStudent);  
 System.***out***.println(**"Cloned Object : "** + clonedStudent);  
 }  
}

Output is:

Original Object: Student{name='John', subj=Subject{name='Chemistry'}}

Cloned Object: Student{name='John', subj=Subject{name='Chemistry'}}

--------- Inside show() ---------

Original Object : Student{name='Dan', subj=Subject{name='Physics'}}

Cloned Object : Student{name='John', subj=Subject{name='Physics'}}

---- After Method Modification -----

Original Object : Student{name='Trio', subj=Subject{name='Musicology'}}

Cloned Object : Student{name='Music', subj=Subject{name='Musicology'}}

-------- After Original Object Changed --------

Original Object : Student{name='Trio', subj=Subject{name='Musicology'}}

Cloned Object : Student{name='Music', subj=Subject{name='Musicology'}}

-------- After Cloned Object Changed --------

Original Object : Student{name='Trio', subj=Subject{name='Biology'}}

Cloned Object : Student{name='Smith Clone', subj=Subject{name='Biology'}}

In this example, all I did is, implement the class that you want to copy with Clonable interface and override clone() method of Object class and call super.clone() in it. If you observe, the changes made to "name" field of original object (Student class) is not reflected in cloned object but the changes made to "name" field of contained object (Subject class) is reflected in cloned object. This is because the cloned object carries the memory address of the Subject object but not the actual values. Hence any updates on the Subject object in Original object will reflect in Cloned object.

**What is Deep Copy?**

**A deep copy copies all fields, and makes copies of dynamically allocated memory pointed to by the fields. A deep copy occurs when an object is copied along with the objects to which it refers.**

**Well, here we are with what shallow copy and deep copy are and obviously the difference between them. Now lets see how to implement them in java.**

**class** Subject {  
  
 **private** String **name**;  
  
 **public** Subject(String name) {  
 **this**.**name** = name;  
 }  
  
 get()/set() Methods

@Override  
 **public** String toString() {  
 **return "Subject{"** +  
 **"name='"** + **name** + **'\''** +  
 **'}'**;  
 }  
}  
  
**public class** Student **implements** Cloneable {  
 **private** Subject **subject**;  
 **private** String **name**;  
  
 **public** Student(String name) {  
 **this**.**name** = name;  
 }  
  
 **public Student(Student student ) { 🡸 Using Copy Constructor  
 this(student.getName());  
 Subject subject = new Subject(student.getSubject().getName());  
 this.setSubject(subject);  
 }**  
 **get()/set() methods**

@Override  
 **public** String toString() {  
 **return "Student{"** +  
 **subject** +  
 **", name='"** + **name** + **'\''** +  
 **'}'**;  
 }

@Override *//Clone Method for deep copy*  
 **protected** Object clone() **throws** CloneNotSupportedException {  
 Student student = **new** Student(**this**.**name**);  
 Subject subject = **new** Subject(**this**.getSubject().getName());  
 student.setSubject(subject);  
 **return** student;  
 }  
}

Test Program and output

**public class** TestDeepCopy {  
  
 **public static void** main(String[] args) **throws** Exception {  
  
 Subject sub1 = **new** Subject(**"Physics"**);  
 Student student = **new** Student(**"John"**);  
 student.setSubject(sub1);  
  
 *//Create the deep copy of the object* Student deepCopy\_1 = **new** Student(student);  
 deepCopy\_1.setName(**"Vidya Balan"**);  
 deepCopy\_1.getSubject().setName(**"Chemistry"**);  
  
 *//Create another deep copy of the object* Student deepCopy\_2 = **new** Student(deepCopy\_1);  
 deepCopy\_2.setName(**"Rani Mukherjee"**);  
 deepCopy\_2.getSubject().setName(**"Mathematics"**);  
  
 *//Deep copy using clone* Student cloned1 = (Student) student.clone();  
 cloned1.setName(**"Amitav"**);  
 cloned1.setSubject(**new** Subject(**"Psychology"**));  
  
 System.***out***.println(**"Normal Copy :::"** + student);  
 System.***out***.println(**"Deep Copy 1:::"**+deepCopy\_1);  
 System.***out***.println(**"Deep Copy 2:::"**+deepCopy\_2);  
  
 System.***out***.println(**"Deep Copy using Clone 1::::"**+cloned1);  
  
  
 }  
}

**Output**

Normal Copy :::Student{Subject{name='Physics'}, name='John'}

Deep Copy 1:::Student{Subject{name='Chemistry'}, name='Vidya Balan'}

Deep Copy 2:::Student{Subject{name='Mathematics'}, name='Rani Mukherjee'}

Deep Copy using Clone 1::::Student{Subject{name='Psychology'}, name='Amitav'}

**There is an alternative way for deep copy.**

Yes, there is. You can do deep copy through serialization. What does serialization do? It writes out the whole object graph into a persistent store and read it back when needed, which means you will get a copy of the whole object graph when you read it back. This is exactly what you want when you deep copy an object. Note, when you deep copy through serialization, you should make sure that all classes in the object's graph are serializable.

Its very simple that if the object has only primitive fields, then obviously you will go for shallow copy but if the object has references to other objects, then based on the requirement, shallow copy or deep copy should be chosen. What I mean here is, if the references are not modified anytime, then there is no point in going for deep copy. You can just opt shallow copy. But if the references are modified often, then you need to go for deep copy. Again there is no hard and fast rule, it all depends on the requirement.