Shallow Copy

When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. The following example demonstrates the same.

FileName: ShallowCopyExample.java

```
class ABC
{
// instance variable of the class ABC
int x = 30;
}
public class ShallowCopyExample
{
// main method
public static void main(String argvs[])
{
// creating an object of the class ABC
ABC obj1 = new ABC();

// it will copy the reference, not value
ABC obj2 = obj1;

// updating the value to 6
// using the reference variable obj2
obj2.x = 6;

// printing the value of x using reference variable obj1
System.out.println("The value of x is: " + obj1.x);
}
```

```
The value of x is: 6
```

Explanation: In the above example, we are updating the value of x using the reference variable obj2 and displaying the value of x using the reference variable obj1. In the output, we see the updated value 6 and not the original value 30. It is because obj1 and obj2 are referring to the same memory location. Therefore, whatever update we do use the reference variable obj2, the same changes will be reflected using the reference variable obj1.

Deep Copy

When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference. The following example demonstrates the same.

FileName: DeepCopyExample.java

```
class ABC
{
// instance variable of the class ABC
int x = 30;
}
public class DeepCopyExample
{
// main method
public static void main(String argvs[])
{
// creating an object of the class ABC
ABC obj1 = new ABC();

// it will copy the reference, not value
ABC obj2 = new ABC();

// updating the value to 6
// using the reference variable obj2
obj2.x = 6;

// printing the value of x using reference variable obj1
System.out.println("The value of x is: " + obj1.x);
}
}
```

```
The value of x is: 30
```

Explanation: In the above example, we are updating the value of x using the reference variable obj2 and displaying the value of x using the reference variable obj1. In the output, we see the original value 30, not the updated value 6. It is because obj1 and obj2 are referring to different memory locations. Therefore, whatever update we do use the reference variable obj2, the same is not reflected using the reference variable obj1.

Differences Between Shallow Copy and Deep Copy

Shallow Copy	Deep Copy
It is fast as no new memory is allocated.	It is slow as new memory is allocated.
Changes in one entity is reflected in other entity.	Changes in one entity are not reflected in changes in another identity.
The default version of the clone() method supports shallow copy.	In order to make the clone() method support the deep copy, one has to override the clone() method.
A shallow copy is less expensive.	Deep copy is highly expensive.
Cloned object and the original object are not disjoint.	Cloned object and the original object are disjoint.