Shallow Copy Vs. Deep Copy in Java

In this section, we will discuss the key difference between **shallow copy and deep copy in Java.** Let's understand the shallow and deep copy.

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);
}
}
```

Output:

```
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);
}
}
```

Output:

```
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

After learning about shallow and deep copy, let's see the differences between shallow 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.

Scenarios where deep and shallow copies are similar

Although there are differences between deep and shallow copy, there are some scenarios where it makes no sense to consider whether the copy is deep or shallow.

In Strings

Let's learn what will happen when doing a copy of the string. We all know strings are considered as the objects of the class String present in *java.lang* package. So, similar to other objects, when we do the copy, the reference is copied.

FileName: StringCopyExample.java

```
public class StringCopyExample
{
// main method
public static void main(String argvs[])
{
// an object of the String class is created
String obj1 = new String("JavaTpoint is a very good site.");

// copying obj1 to obj2
String obj2 = obj1;
```

```
// printing the hash code using the reference variable obj1.
System.out.println("The hash code is: " + obj1.hashCode());

// printing the hash code using the reference variable obj2.
System.out.println("The hash code is: " + obj2.hashCode());
}
}
```

Output:

```
The hash code is: -2026030341
The hash code is: -2026030341
```

Explanation: The output of the program shows that the hash code shown by the reference variables obj1 and obj2 are the same. It means that the same memory location is referenced by the reference variables obj1 and obj2. But the question is, can we say that we have done the shallow copy in the above program, as the reference is the same? The answer is no. The following example gives us enough evidence to validate the given answer.

FileName: StringCopyExample1.java

```
public class StringCopyExample
{
// main method
public static void main(String argvs[])
{
// an object of the String class is created
String obj1 = new String("JavaTpoint is a very good site.");

// copying obj1 to obj2
String obj2 = obj1;
// we have updated the string for reference variable obj2
obj2 = "JavaTpoint is very good.";
```

```
// printing the hash code using the reference variable obj1.
System.out.println("The hash code is: " + obj1.hashCode());
// printing the string
System.out.println("The string is: " + obj1 + "\n");



// printing the hash code using the reference variable obj2.
System.out.println("The hash code is: " + obj2.hashCode());
// printing the string
System.out.println("The string is: " + obj2);
}
}
```

Output:

```
The hash code is: -2026030341
The string is: JavaTpoint is a very good site.

The hash code is: 1724527163
The string is: JavaTpoint is very good.
```

Explanation: The output of the program tells us that the hash code shown by the reference variable obj1 is not equal to the hash code shown by the reference variable obj2. Also, the changes done using the reference variable obj2 is not shown by the reference variable obj1. It is because **strings in Java are always immutable**. Therefore, when obj2 changes the content of obj1, it ends up creating an entirely new string. Thus, the previous string remains untouched, and the reference variable obj2 points to a new memory location where the new string object is sitting.

We have seen that a change in a string results in the creation of a new string object implicitly. Therefore, copying a string can neither be termed as the deep nor as the shallow copy. In fact, there is no difference between deep and shallow copy when we are dealing with strings in Java.

In Primitive Data Types

Let's learn what will happen when doing a copy of primitive data types. Unlike strings, the primitive data types are not objects. However, similar to strings, there is no concept of deep or shallow copy in the primitive data types. Observe the following example.

FileName: DataTypeCopyExample.java

```
public class DataTypeCopyExample
{
// main method
public static void main(String argvs[])
{
int x = 9;
int y = x;

System.out.println("The value of x & y are: " + x + ", " + y);

// updating the value of y
y = 10;

System.out.println("The value of x & y are: " + x + ", " + y);

boolean b1 = false;

// copying the value of b1 in b2
boolean b2 = b1;

System.out.println("The value of b1 & b2 are: " + b1 + ", " + b2);

// updating the value of b2
b2 = true;
System.out.println("The value of b1 & b2 are: " + b1 + ", " + b2);
```

```
}
}
```

Output:

```
The value of x & y are: 9, 9
The value of x & y are: 9, 10
The value of b1 & b2 are: false, false
The value of b1 & b2 are: false, true
```

Explanation: When the value of y is updated, it is not impacting the value of x. It is because y already has its own memory allocation. It is not referring to the memory location of x. Therefore, the statement y = x; only copies the value of x in y. Therefore, any update in y never impacts x. A similar concept can be applied to the other primitive data types as well. Here, memory allocation for y as well for b2 is happening implicitly.

In Java, there is no rule stating that when to use shallow copy and when to use the deep copy. It is up to the programmers or developers to decide what they want to use. Therefore, it is recommended to understand the requirement then decide judiciously between deep and shallow copy.

Apart from deep and shallow copy, there is another term called **lazy copy**. Lazy copy is, in fact, the mixture of the deep and shallow copy. In lazy copy, the shallow copy is used at the starting stage. When one modifies the original content, the program checks whether the content is shared among the other objects or not with the help of a counter. If the content is shared, then the deep copy mechanism is applied.

← Prev

Next →