Clone() method in Java

Object cloning refers to creation of exact copy of an object. It creates a new instance of the class of current object and initializes all its fields with exactly the contents of the corresponding fields of this object.

Using Assignment Operator to create copy of reference variable
In Java, there is no operator to create copy of an object. Unlike C++, in Java, if we use assignment operator then it will create a copy of reference variable and not the object. This can be explained by taking an example. Following program demonstrates the same.

```java
// Java program to demonstrate that assignment
// operator only creates a new reference to same
// object.
import java.io.*;

// A test class whose objects are cloned
class Test
{
    int x, y;
    Test()
    {
        x = 10;
        y = 20;
    }
}

// Driver Class
class Main
{
    public static void main(String[] args)
    {
         Test ob1 = new Test();

         System.out.println(ob1.x + " " + ob1.y);

         // Creating a new reference variable ob2
         // pointing to same address as ob1
         Test ob2 = ob1;
```

```
            // Any change made in ob2 will be reflected
            // in ob1
            ob2.x = 100;

            System.out.println(ob1.x+" "+ob1.y);
            System.out.println(ob2.x+" "+ob2.y);
        }
    }
```

Output:

```
10 20
100 20
100 20
```

Creating a copy using clone() method

The class whose object's copy is to be made must have a public clone method in it or in one of its parent class.

- Every class that implements clone() should call super.clone() to obtain the cloned object reference.
- The class must also implement java.lang.Cloneable interface whose object clone we want to create otherwise it will throw CloneNotSupportedException when clone method is called on that class's object.
- Syntax:

```
protected Object clone() throws CloneNotSupportedException
```

Usage of clone() method -Shallow Copy

```
// A Java program to demonstrate shallow copy
// using clone()
import java.util.ArrayList;

// An object reference of this class is
// contained by Test2
class Test
{
    int x, y;
}

// Contains a reference of Test and implements
// clone with shallow copy.
class Test2 implements Cloneable
{
    int a;
    int b;
    Test c = new Test();
    public Object clone() throws
                   CloneNotSupportedException
    {
        return super.clone();
    }
}

// Driver class
public class Main
{
    public static void main(String args[]) throws
                          CloneNotSupportedException
    {
       Test2 t1 = new Test2();
       t1.a = 10;
       t1.b = 20;
       t1.c.x = 30;
       t1.c.y = 40;

       Test2 t2 = (Test2)t1.clone();

       // Creating a copy of object t1 and passing
       //  it to t2
       t2.a = 100;

       // Change in primitive type of t2 will not
       // be reflected in t1 field
       t2.c.x = 300;

       // Change in object type field will be
       // reflected in both t2 and t1(shallow copy)
       System.out.println(t1.a + " " + t1.b + " " +
                          t1.c.x + " " + t1.c.y);
       System.out.println(t2.a + " " + t2.b + " " +
                          t2.c.x + " " + t2.c.y);
    }
}
```

Output:

```
10 20 300 40
100 20 300 40
```

In the above example, t1.clone returns the shallow copy of the object t1. To obtain a deep copy of the object certain modifications have to be made in clone method aft obtaining the copy.

Deep Copy vs Shallow Copy

- **Shallow copy** is method of copying an object and is followed by default in cloning. In this method the fields of an old object X are copied to the new object Y. While copying the object type field the reference is copied to Y i.e object Y will point to same location as pointed out by X. If the field value is a primitive type it copies the value of the primitive type.
- Therefore, any changes made in referenced objects in object X or Y will be reflected in other object.

Shallow copies are cheap and simple to make. In above example, we created a shallow copy of object.

Usage of clone() method – Deep Copy

- If we want to create a deep copy of object X and place it in a new object Y then new copy of any referenced objects fields are created and these references are placed in object Y. This means any changes made in referenced object fields in object X or Y will be reflected only in that object and not in the other. In below example, we create a deep copy of object.
- 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.

```java
// A Java program to demonstrate deep copy
// using clone()
import java.util.ArrayList;

// An object reference of this class is
// contained by Test2
class Test
{
    int x, y;
}


// Contains a reference of Test and implements
// clone with deep copy.
class Test2 implements Cloneable
{
    int a, b;

    Test c = new Test();

    public Object clone() throws
                CloneNotSupportedException
    {
        // Assign the shallow copy to new reference variable t
        Test2 t = (Test2)super.clone();
        t.c = new Test();

        // Create a new object for the field c
        // and assign it to shallow copy obtained,
        // to make it a deep copy
        return t;
    }
}

public class Main
{
    public static void main(String args[]) throws
                                CloneNotSupportedException
    {
       Test2 t1 = new Test2();
       t1.a = 10;
       t1.b = 20;
       t1.c.x = 30;
       t1.c.y = 40;

       Test2 t3 = (Test2)t1.clone();
       t3.a = 100;

       // Change in primitive type of t2 will not
       // be reflected in t1 field
       t3.c.x = 300;

       // Change in object type field of t2 will not
       // be reflected in t1(deep copy)
       System.out.println(t1.a + " " + t1.b + " " +
                          t1.c.x + " " + t1.c.y);
       System.out.println(t3.a + " " + t3.b + " " +
                          t3.c.x + " " + t3.c.y);
    }
}
```

Output:

```
10 20 30 40
100 20 300 0
```

In the above example, we can see that a new object for Test class has been assigned to copy object that will be returned in clone method.Due to this t3 will obtain a deep copy of the object t1. So any changes made in ‘c’ object fields by t3 ,will not be reflected in t1.

Advantages of clone method:

- If we use assignment operator to assign an object reference to another reference variable then it will point to same address location of the old object and no new copy of the object will be created. Due to this any changes in reference variable will be reflected in original object.

- If we use copy constructor, then we have to copy all of the data over explicitly i.e. we have to reassign all the fields of the class in constructor explicitly. But in clone method this work of creating a new copy is done by the method itself.So to avoid extra processing we use object cloning.

This article is contributed by **Ankit Agarwal.** If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Top IT MNCs hiring freshers - CTC:25k-5Lac/
Engineering, BCA, MCA, BSc Graduates & Diploma Holders c

Recommended Posts:

Java.util.BitSet class methods in Java with Examples | Set 2
Shadowing of static functions in Java
How does default virtual behavior differ in C++ and Java ?
How are Java objects stored in memory?
How are parameters passed in Java?
Are static local variables allowed in Java?
final variables in Java
Default constructor in Java
Assigning values to static final variables in Java
Comparison of Exception Handling in C++ and Java
Does Java support goto?
Arrays in Java
Inheritance and constructors in Java
More restrictive access to a derived class method in Java
Comparison of static keyword in C++ and Java

Improved By : Akanksha_Rai