Appendix 10. Cloning

•

- A Clone of an Object
  - » Exact Copy of the object
  - » Exact : A Clone should have exactly the same data values as the original object
  - » Copy : should be a distinct object and not simply another name for the original one

- A Clone is made by invoking the method named clone.
  - » Clone method is defined in the class Object
  - » Redefined for a specified class (Override)

- ArrayList
  - » defines its own clone method

```
ArrayList<String> aList = new ArrayList<String>();
<Some code to fill aList>
```

We then can make an identical copy of `aList`, so that we have two separate copies, by invoking `ArrayList`'s method `clone`:

```
ArrayList<String> duplicateList =
                    (ArrayList<String>)aList.clone();
```

- 예) the Class PET
  - » the class must implement the standard interface Cloneable
    - – Actually empty

```
public class Pet implements Cloneable
```

    - – the heading for the method clone in the class Object

```
protected Object clone()
```

    - – Pet will override this method with the following public Verson

```
public Object clone()
```

LISTING A10.1 A Simple Implementation of the Method clone

```
public Object clone()
{
    try
    {
        return super.clone();//Invocation of Object's clone
    }
    catch(CloneNotSupportedException e)
    {//This should not happen.
        return null;  //To keep the compiler happy.
    }
}
```

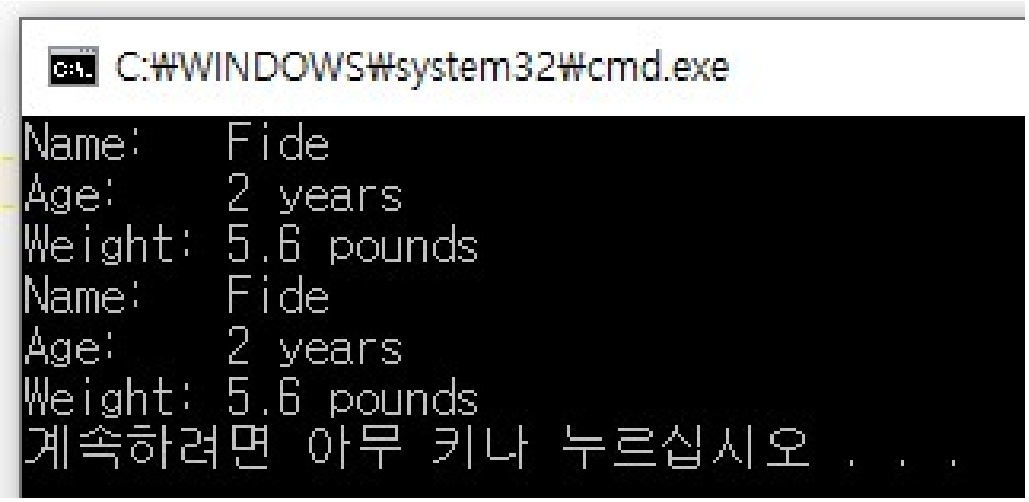
Works correctly for a class like **Pet**, *in which each instance variable has either a primitive type or the type* **String**. *Will not work correctly in most other cases.*

예#1) Pet Class

```
public class Pet implements Cloneable
{
   private String name;
   private int age;      //in years
   private double weight;//in pounds
   ……
   ……
 public void writeOutput( )
   {
      System.out.println("Name:   " + name);
      System.out.println("Age:    " + age + " years");
      System.out.println("Weight: " + weight + " pounds");
   }
   public Object clone()
   {
                        try
                        {
                                    return super.clone(); // invocation of Object's clone
                        }
                        catch (CloneNotSupportedException e)
                        {
                                    // This should not happen
                                    return null; // Too keep the complier happy
                        }
    }
}
```

Pet Demo Class

```
import java.util.Scanner;

public class PetDemo
{
   public static void main(String[] args)

   {
                    Pet originalData = new Pet("Fide", 2, 5.6);
                        originalData.writeOutput( );
                    Pet duplicateData = (Pet) originalData.clone();
                        duplicateData.writeOutput( );
   }
}
```

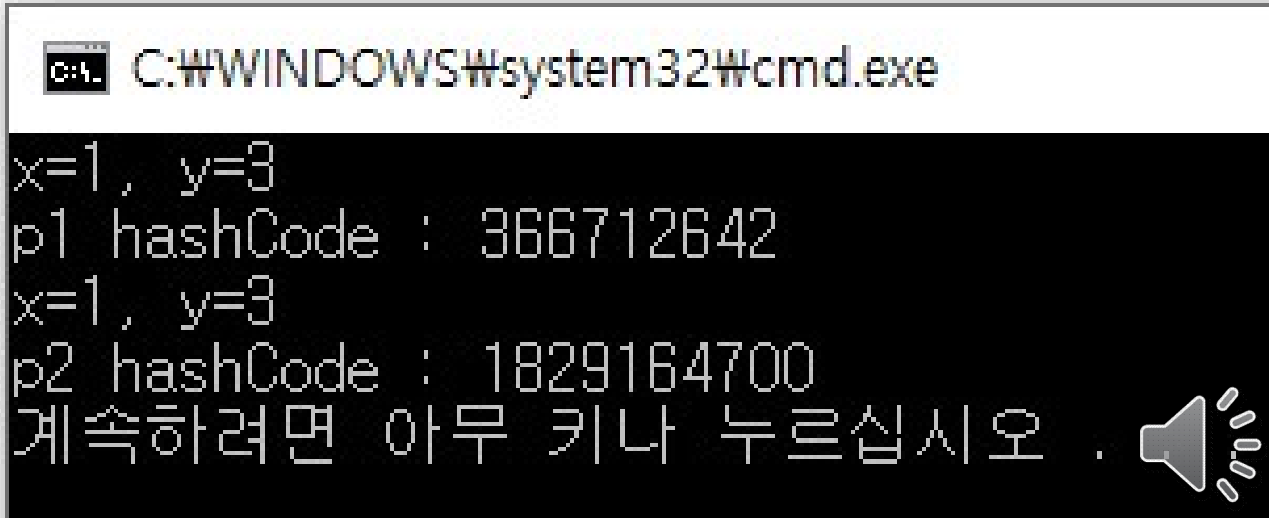


예#2) Point Class

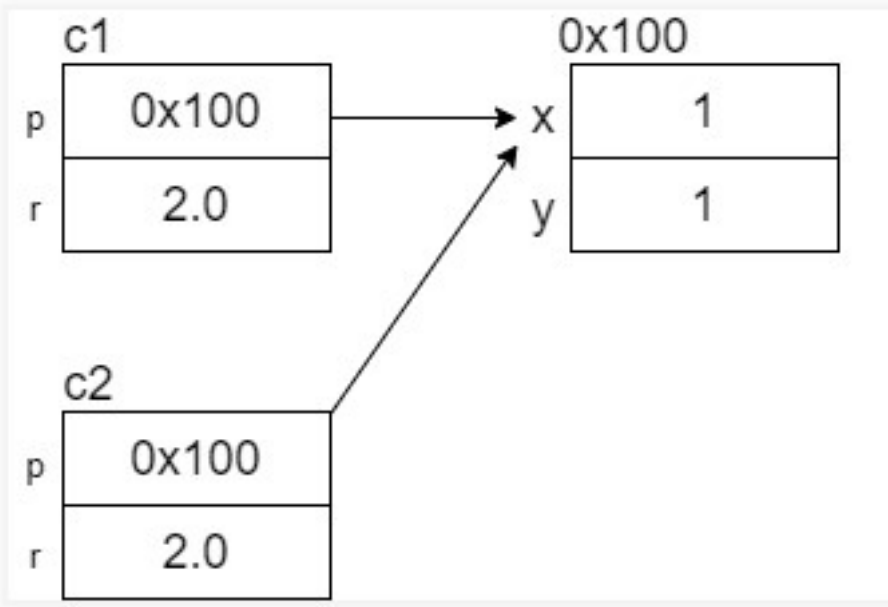
```
public class Point implements Cloneable {

    private int x;
    private int y;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString(){
        return "x=" + x + ", y=" + y;
    }

  @Override
    public Point clone(){
        Object obj = null;
        try{
            obj = super.clone();
        } catch (CloneNotSupportedException e){
            e.printStackTrace();
        }
        return (Point) obj;
    }

}
```

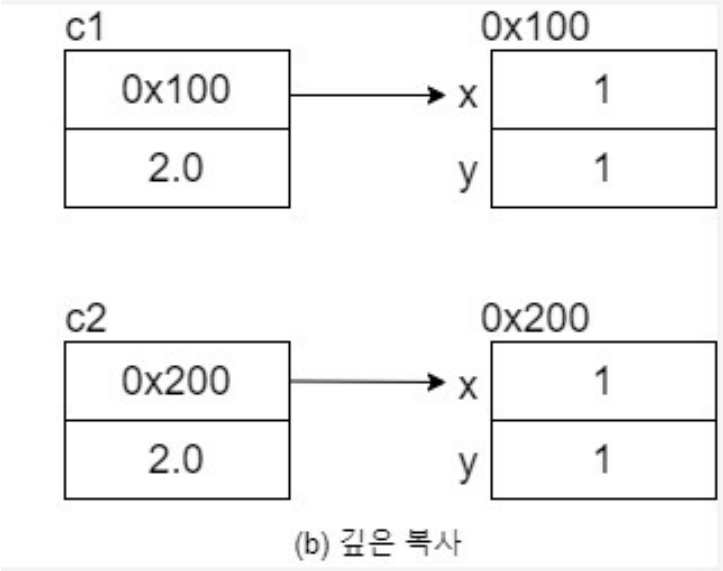
Point Demo Class

```
public class PointDemo {
    public static void main(String[] args){
        Point p1 = new Point(1,3);
 //      Point p2 = (Point)p1.clone();
         Point p2 = p1.clone();
        System.out.println(p1.toString());
        System.out.println("p1 hashCode : " + p1.hashCode());
        System.out.println(p2.toString());
        System.out.println("p2 hashCode : " + p2.hashCode());
    }
}
```



Shallow Copy, Deep Copy





(b) 깊은 복사

예#3) Point Class

```
public class Point {

    int x;
    int y;

    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString(){
        return "x=" + x + ", y=" + y;
    }

}
```

Circle Class

```
class Circle implements Cloneable {

Point p; //원점
double r; //반지름

Circle (Point p, double r) {
this.p = p;
this.r = r;
}

public Circle shallowCopy(){
Object obj = null;
try {
        obj = super.clone();
        } catch (CloneNotSupportedException e) {}
        return (Circle) obj;
}
```

```
public Circle deepCopy() { //깊은 복사
        Object obj = null;
        try {
                obj = super.clone();
        } catch (CloneNotSupportedException e) {}
        Circle  c = (Circle) obj;
        c.p = new Point(this.p.x, this.p.y);
        return c;
}
public String toString() {
        return "p=" + p + " /r= " + r;
        }
}
```

ShallowDeepCopyMain Class

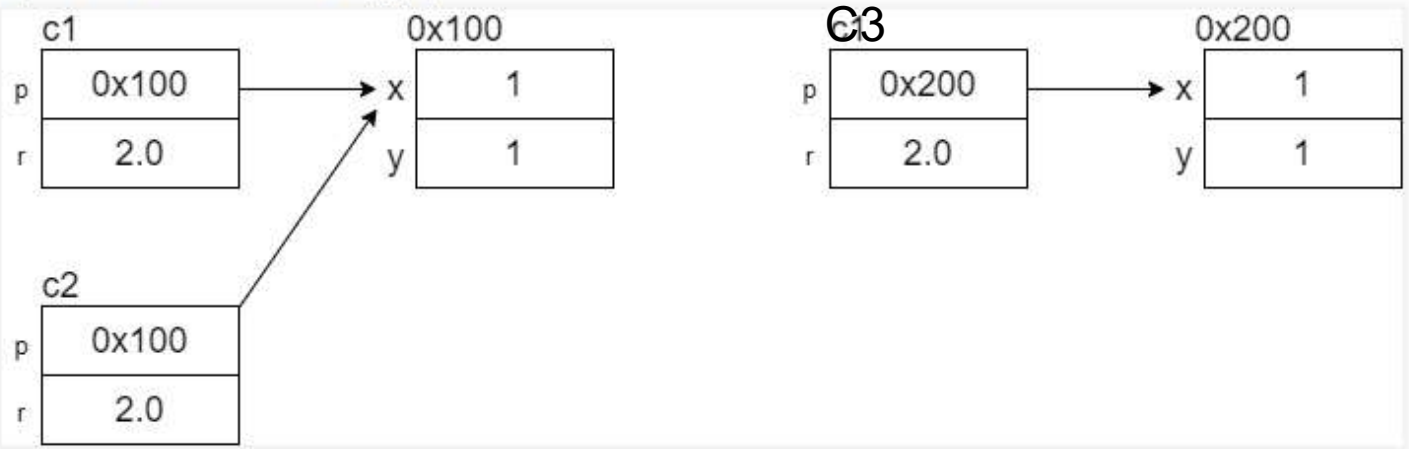
```
public class ShallowDeepCopyMain {
public static void main(String[] args){
        Circle c1 = new Circle(new Point(1,1), 2.0);
        Circle c2 = c1.shallowCopy(); // 얕은 복사
        Circle c3 = c1.deepCopy(); // 깊은 복사

        System.out.println("c1 : " + c1);
        System.out.println("c2 : " + c2);
        System.out.println("c3 : " + c3);

        c1.p.x = 9;
        c1.p.y = 9;
        System.out.println ("c1의 변경 후") ;
        System.out.println("c1 : " + c1);
        System.out.println("c2 : " + c2);
        System.out.println("c3 : " + c3);
        }
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
c1 : p=x=1, y=1 /r= 2.0
c2 : p=x=1, y=1 /r= 2.0
c3 : p=x=1, y=1 /r= 2.0
c1의 변경 후
c1 : p=x=9, y=9 /r= 2.0
c2 : p=x=9, y=9 /r= 2.0
c3 : p=x=1, y=1 /r= 2.0
계속하려면 아무 키나 누르십시오 . . .
```

인스턴스 c1을 생성한 후에 얕은 복사로 c2을 생성하고 깊은 복사로 c3을 생성했다.

```
Circle c1 = new Circle(new Point(1, 1), 2.0);
Circle c2 = c1.shallowCopy(); // 얕은 복사
Circle c3 = c1.deepCopy();    // 깊은 복사
```

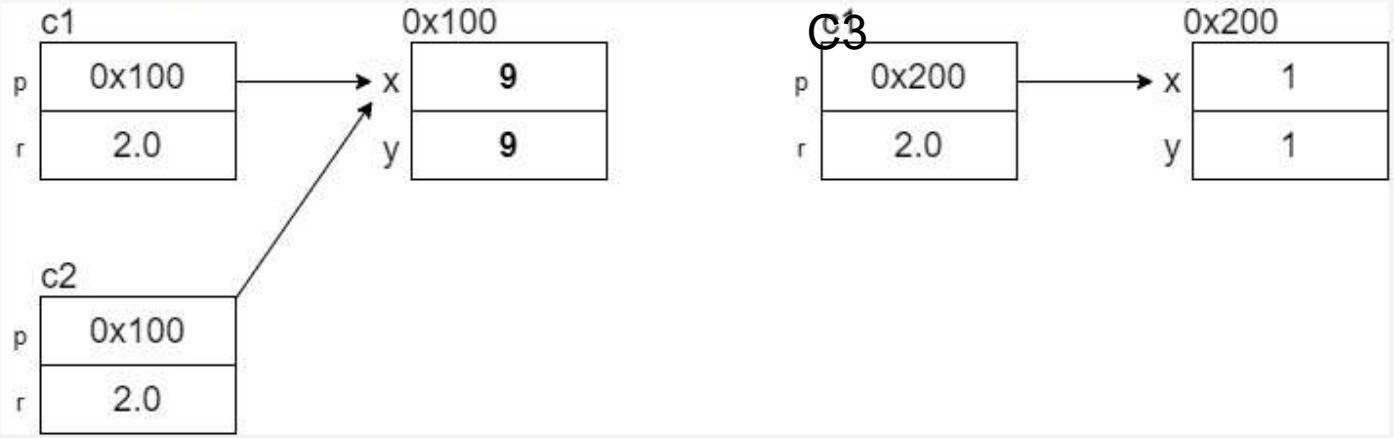
위 상황을 그림으로 표현하였다.



그다음 c1이 가리키는 Point인스턴스의 x와 y의 값을 9로 변경한다.

```
c1.p.x = 9;
c1.p.y = 9;
```

c1 만 변경했는데 c2도 영향을 받는다, c3은 전혀 영향을 받지 않는다.



- Cloning an object of ArrayList
  - » does not clone the objects in the list //ShallowCopy
  - » While aList and duplicateList does not clone the objects in the list, they Share the same Strings

```
ArrayList<String> aList = new ArrayList<String>();
<Some code to fill aList>
```

We then can make an identical copy of aList, so that we have two separate copies, by invoking ArrayList's method clone:

```
ArrayList<String> duplicateList =
                    (ArrayList<String>)aList.clone();
```

끝