Java.lang - Object Ⅱ

하나의 약속! equals( ) Method

Java에서는 이미 인스턴스의 참조값 비교를 목적으로 Object Class에 equals 메소드를 정의해 놓았다.

i. 인스턴스 참조값 비교 (boolean값 반환)

```
object1.equals(object2)
```

i. Overriding을 통한 내용 비교

```
Class Inst{
     int variable = 0;

     Public Inst(int n){
          variable = n;
     }

     @Override
     public boolean equals(Object obj){
          if(this.variable == ((Inst)obj).variable) // 인자가 Object 형이므로 비교에 맞게 Inst 형으로 Casting!
               return true;
          else
               return false;
     }
}
```

이제는 이해할 수 있는 재미있는 사실! 문자열 비교의 진실

문자열도 String Class의 인스턴스 이므로, equals로 비교해야 한다! 그렇다면 문자열이 어떻게 비교가 가능한지 알아보자!

i. 우선, "Hello"라는 문자열을 코드상에 작성하면 Java는 String 인스턴스를 하나 만든다.

ii. String Class도 Object의 자식 Class이므로 equals( ) Method가 있다. 이 때의 equals Method는 인스턴스의 참조값을 비교한다.

iii. "Hello".equals( ) 메소드의 인자가 들어갈 자리에 "Hello"라고 입력해보자! → "Hello".equals("Hello")

iv. 중요한 사실은 이전에 배운 Stirng Class의 내용을 다시 보자

"동일한 문자열로 String Intance가 생성되면 새로 생성하지 않고, 이전의 동일한 문자열의 인스턴스의 메모리 주소값(참조값)만 반환한다."

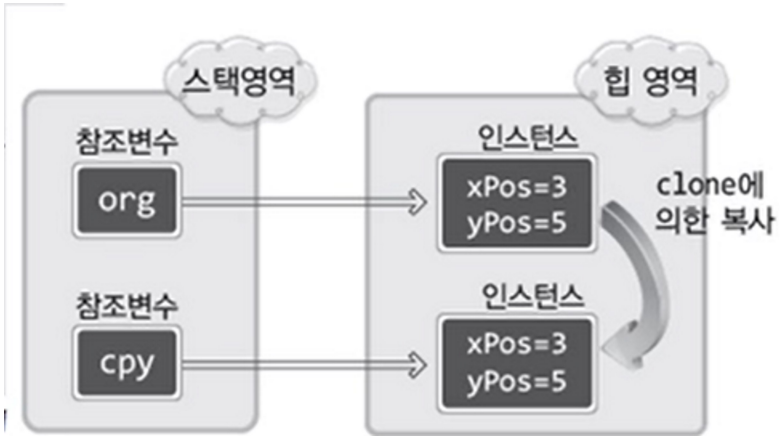
v. 결국 동일한 참조값을 가진 두 문자열을 비교하게 되므로 true가 반환된다!

인스턴스의 복사 : clone Method

- Object Class에는 인스턴스의 복사를 목적으로 clone이라는 메소드가 정의되어 있다.
- 복사된 인스턴스의 참조값이 반환.

- 단, 이 메소드는 Cloneable 인터페이스를 구현하는 클래스의 인스턴스 만이 호출 가능하다!
  (Cloneable 인터페이스의 구현의미 : 이 클래스를 복사해도 됩니다!)
- 사실, 인스턴스의 복사는 매우 민감한 작업이다. 따라서 클래스를 정의할 때 복사의 허용여부를 결정하도록 Cloneable 인터페이스를 통해서 요구하는 것이다. (표시목적을 위한 인터페이스)

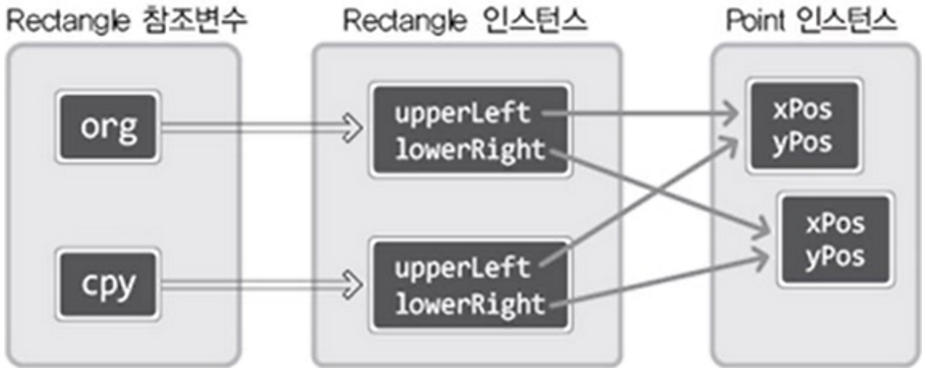
```
class Point implements Cloneable {
     private int xPos;
     private int yPos;
     . . . . .

     // clone 메소드는 protected로 선언되어 있지만, 외부에서 호출 가능하도록 public 선언
     @Override
     public Object clone() throws CloneNotSupportedException {
          return super.clone();
     }
}
```




```
Point org = new Point();
Point cpy = (Point)org.clone(); // 반환형이 Object 형이므로 참조변수 타입에 맞게 Casting!
```

얕은 복사 (Shallow Clone)

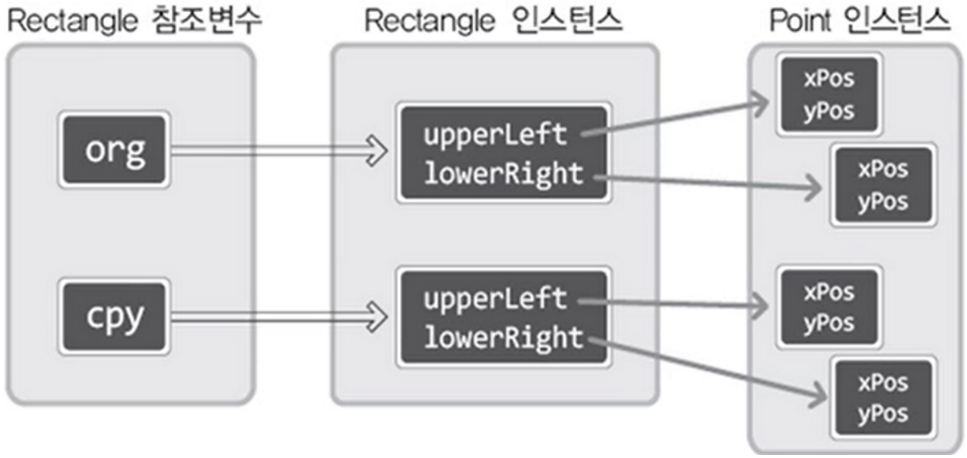
```
class Rectangle implements Cloneable {
     Point upperLeft; lowerRight;

     public Rectangle(int x1, int y1, int x1, int x2){
          upperLeft = new Point(x1, y1);
          lowerRight = new Point(x2, y2);
     }
      . . . . .

     // 외부에서 접근을 허용하기 위해 메소드를 Overriding해서 public으로 확장 해준다.
     public Object clone() throws CloneNotSupportedException {
          return super.clone();
     }
}

class ShallowClone {
     public static void main(String[] args){
          Rectangle org = new Rectangle(1, 1, 9, 9);
          Rectangle cpy = null;

          try{

               cpy = (Rectangle)org.clone();
               org.changePos(2, 2, 7, 7);
               org.showPosition();
               cpy.showPosition();

          }catch(CloneNotSupportedException e){
               e.printStackTrace();
          }
     }
}
```



- Rectangle의 참조변수와 참조값만 복사되고 Point 인스턴스 자체를 복사해서 새로 만들지 않는다.
- 그러므로 원본과 복사본의 인스턴스는 동일한 Point 인스턴스를 참조하는 모양이 되어버린다.

깊은 복사 (Deep Clone)

```
class Rectangle implements Cloneable {
     Point upperLeft; lowerRight;

     public Rectangle(int x1, int y1, int x1, int x2){
          upperLeft = new Point(x1, y1);
          lowerRight = new Point(x2, y2);
     }
      . . . . .

     public Object clone() throws CloneNotSupportedException {
          Rectangle copy = (Rectagnle)super.clone();

          // 참조변수의 인스턴스까지 통째로 복사!
          copy.upperLeft = (Point)upperLeft.clone();
          copy.lowerRight = (Point)lowerRight.clone();

          return copy;
     }
}
```



String 인스턴스와 배열 인스턴스의 복사 (얕은 복사)

- String 인스턴스에 저장된 문자열은 바꿀 수 없다 (상수화된 인스턴스) 따라서, 굳이 깊은 복사를 할 필요가 없다.

- 배열 인스턴스를 복사할 경우 배열 인스턴스의 복사본이 반환되는데, 배열과 배열에 저장된 참조값은 복사되지만,배열의 참조값이 참조하는 인스턴스까지는 복사되지 않는다.