13.04 Virtual Lecture Notes

For example, System.out.println(rect), where rect is of type Rectangle, prints a reference to the rect object. If we wanted the same code to print out the rectangle's dimensions, then we would need to override the toString() method.

This is accomplished by adding a toString() method to our Rectangle class. It will look like this:

```
        public String toString()
        {
           return "This Rectangle is " + length + " X " + width;
        }
```

- Download the Rectangle2.java file to your demo programs folder. Open it to see the complete class.

Now, **System.out.println(rect)** will print the dimensions of our rectangle (i.e., This Rectangle is 10 x 4). This is much more useful than the default toString() method we would have inherited from the Object class.

The question remains: what happens with Box? Assume that Box2 is the same as the original Box, but extends Rectangle2. Assume myBox is of type Box2; then System.out.println(myBox) should show the dimensions of the Box, right? Wrong. Since we have not added a toString() to the Box2 class, we would get a message saying, "This Rectangle …" Obviously, this is not what we want. So we need to override toString() again.

We do this by adding a toString() method to our Box2 class (which is just a copy of the original Box class).

- Download the Box2.java file to your demo programs folder. Open it to see the complete class.

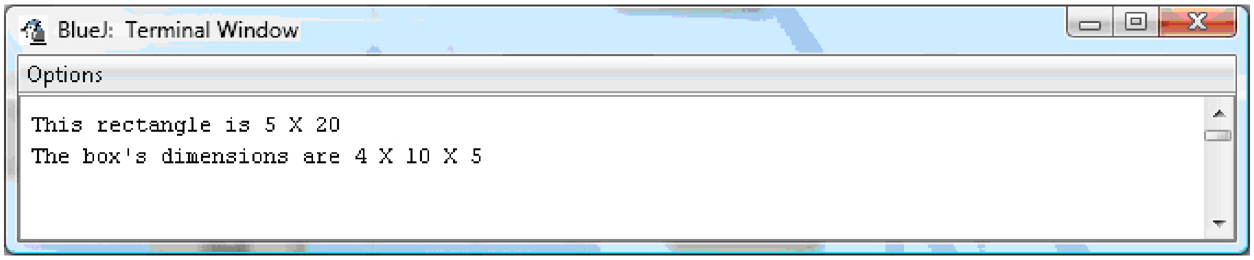
Now, we will get what we want from System.out.println(myBox).

For example, consider this test program:

```
public class test3
{
        public void tester()
        {
       Rectangle2 one = new Rectangle2(5, 20);
       Box2 two = new Box2(4, 10, 5);

            System.out.println(one);
            System.out.println(two);
        }
}
```

The following output will be generated:



```
This rectangle is 5 X 20
The box's dimensions are 4 X 10 X 5
```

- Download the Test3.java file to your demo programs folder. Open and run it to make sure you understand how the classes work.

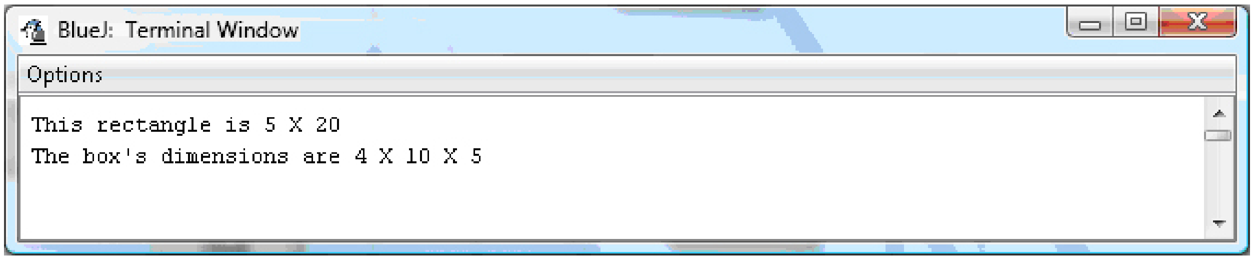
Polymorphism and overriding can be used in combination. For example, in test4, a showEffectBoth method is created that demonstrates the usage of both polymorphism and overriding.

Here is how showEffectBoth looks:

```
        public void showEffectBoth(Rectangle2 r)
        {
           System.out.println(r);
        }
```

Notice that it is polymorphic and then makes use of the toString() overriding in its call to System.out.println().

Here is the output from running test4:



```
This rectangle is 5 X 20
The box's dimensions are 4 X 10 X 5
```

Notice that the output is the same.

- Download the Test4.java file to your demo programs folder. Open it to see the complete class.