15.03 Virtual Lecture Notes

Ms. Mills wants to be able to have fadeJeans() and addHoles() methods for all her jean items in her store. Now, not all of the items are jeans. There are skirts, shirts, jackets and jeans. While all are made of jean material, they are not from the same superclass. So, Ms. Mills needs to create an interface. That way she can use the process on all items in her store.

In order to accomplish this, we are going to make an interface called JeanEffects. We start off by declaring JeanEffects to be an interface.

public interface JeanEffects

Now, we add methods to JeanEffects, but we do not define them. Because it is an interface, the methods should be public and abstract.

Our first attempt at writing the rest of the interface then becomes this:

```
public interface JeanEffects
{ public abstract void fadeJeans(); public abstract void addHoles();
}
```

As a rule though, all methods included in an interface are automatically public and abstract; therefore, we do not have to actually use those keywords; however, you will avoid confusion if you always use public and abstract in your code.

- Be sure you downloaded the JeanEffects.java file to the current project folder.

Now, for any class that needs these methods, we need to implement the JeanEffects interface. Ms. Mills wants a JeanJacket class that implements the JeanEffects interface. She intends for it to also have beads and a design method similar to the BeadedJean class, but it will not extend the BasicJean class. It will, however, have faded effects and some holes for that worn look.

To implement the JeanEffects interface, the first line of the JeanJacket class will be:

public class JeanJacket implements JeanEffects

To finish the implementation, JeanJacket must define the abstract methods fadeJeans() and addHoles(). For JeanJacket this means that holeLocation will be set to “elbows” and fadeLevel will be set to “moderate.” holeLocation and fadeLevel are instance variables for the JeanJacket class.

Here are the two methods:

```
public void fadeJeans() {
        fadeLevel = "moderate"; } public void addHoles() {
        holeLocation = "elbows"; }
```

Now to test it out, we need a TestJacket class. For testing, TestJacket creates three jackets and then tests the design(), fadeJeans(), and addHoles() methods.

- You should have previously downloaded the JeanEffects.java, JeanJacket.java, and TestJacket.java file to the current project folder.
- Compile the project and run the TestJacket class to see how the interface works.

Now, Ms. Mills wants to have a FadedJean class which will extend BasicJean, but implement JeanEffects. However, unlike JeanJacket, FadedJean will have no holes. The way we accomplish this is to create an addHoles() method that does nothing. Remember that a class that implements an interface needs to implement the abstract methods. Ms. Mills wants fadeJeans() to set the fadeLocation to “knees.” Notice that, this time, fadeJeans() functions differently than in JeanJacket.

The TestFadedJean class tests the FadedJean class.

- Be sure your previously downloaded the FadedJean.java and the TestFadedJean.java files to the current project folder.
- Compile project and run TestFadedJean class to see everything in action.

Make sure you understand how the demo programs work before you proceed. This is tricky material and will take some careful study of the code. It is a good time to do a thorough desk check of each class with these Virtual Lecture Notes as a guide.