Lab 11: Working with Interfaces

Objective

This lab will take us back to the **Box** and **Cube** example but add some additional classes, interfaces and behavior. We will also need to re-factor the existing functionality somewhat in order to be more consistent with the new design.

Overview

In this lab you will:

- Create several new classes in an inheritance hierarchy
- Create interfaces to model common behavior between unrelated classes
- Test your new functionality

Step by Step Instructions

Exercise 1: Expand the existing system

1. For this lab, you will be working in the **com.lq.exercises** package of the **ClassExercises** project.

2. We currently have a **Box** class in place as well as a **Cube** class that extends **Box**. However, we also want to model other objects. This forces us to re-think the current architecture. To start the process, create an `abstract` class named **Shape**. We will never just create **Shape** objects. It will only be used to contain common characteristics in our new class structure. Your new **Shape** class should:
   a. Be declared as `public` and `abstract.`
   b. Contain two `private` attributes, `color` and `name`, which are both of type `String`.
   c. Provide `public` getter and setter methods for the new attributes.
   d. Do not worry about writing any constructors for **Shape**. We will just use the set methods to change the `name` and `color`.

3. Modify the **Box** class so that it extends **Shape**.

4. Create a **Rectangle** class that also extends **Shape**. It should have the following characteristics:
   a. Contains two `private` attributes, `length` and `width`, which are both of type `double`.
   b. Has public getter and setter methods for the new attributes.

   c. Has a `public` constructor that accepts two arguments to set the `length` and `width`. This constructor should also set default values for name and color, “Unknown” and “White”.

   d. Has a public constructor to accept all four attributes of a **Rectangle(**two for **Rectangle** and two for **Shape)**

5. Create a **Square** class that extends **Rectangle**. It should have the following characteristics:
   a. No new attributes.
   b. Contain a `public` constructor that accepts one argument and calls the base class constructor.

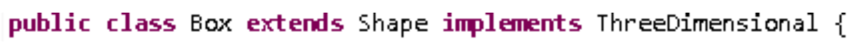
<u>Exercise 2: Working with Interfaces</u>

6. As we think about our system, we know it will grow considerably as we begin to add additional shapes. We want to guarantee that all of our 3-dimensional shapes provide methods that calculate the volume and the surface area. We want to guarantee that all of our 2-dimensional shapes provide methods to calculate the area and perimeter. This is less practical to do with inheritance since each shape will calculate these values differently. We will use interfaces instead. This will leave the details to the implementing class but ensure that classes provide common behavior.

7. Create a new interface in the **com.lq.exercises** package named **ThreeDimensional**. Declare the following public methods in this interface:
   a. `public double getVvolume();`
   b. `public double getSurfaceArea();`

8. Remember, an interface cannot actually implement a method. It can only declare a common signature for a method that must be written by any class that chooses to implement the interface. Don't forget to end each method declaration with a semi-colon.

9. Go back to the **Box** class and change the class declaration as follows:



```
public class Box extends Shape implements ThreeDimensional {
```

10. Notice that we do not even need to change any of the code in **Box**. We have already written the methods that implement this interface.

11. Now we will do the same for our 2 dimensional classes. Create another interface and name it **TwoDimensional**. Declare the following public methods in this interface:
   c. `public double getArea();`

d. `public double getPerimeter();`

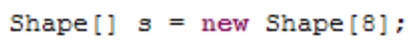
12. Go back to the **Rectangle** class and change the class declaration to implement the new **TwoDimensional** interface.

13. This time you should see that there are compilation errors in your **Rectangle** class. This is because **Rectangle** does not actually implement the interface yet. Add the definitions for the methods in the interface to **Rectangle**. They should be fairly easy to write.

Exercise 3: Test your design

14. Create a new class named **ExerciseShapes**. Ensure that it has a `main()` method.

15. Create an array of 8 **Shape** objects.



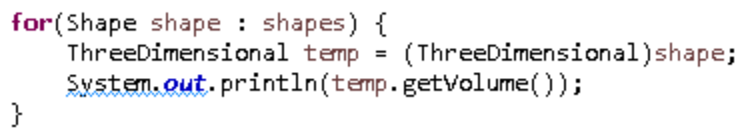
```
Shape[] s = new Shape[8];
```

Add 2 Rectangles, 2 Squares, 2 Cubes and 2 Boxes to the array.

16. Use a for-each to loop through the array and call `setColor()` for each object. Why does this work? Since all objects in the array inherit from **Shape**, they all have a `setColor()` method defined.

17. Now loop through the array and call `getVolume()` for each object. Does it work? No! The **ThreeDimensional** getVolume() method is not defined in the **Shape** class. It is part of the **ThreeDimensional** interface.

18. Change the loop so that it looks as follows:

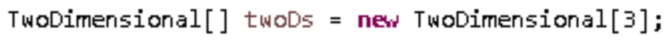


```
for(Shape shape : shapes) {
    ThreeDimensional temp = (ThreeDimensional)shape;
    System.out.println(temp.getVolume());
}
```

This will compile because we are temporarily casting the object in the array to the interface type. However, when we run the code, this will cause a run-time exception because the first 4 objects in the array are 2 dimensional. They do not implement the **ThreeDimensional** interface and cannot be cast that way. We can fix this by starting the loop with `j=4` but this is dangerous because it forces us to know exactly which element type we are working with. Let's fix that. **[Comment out the loop above since it will continue to fail when we run it.]**

19. Create a new class named **Circle**. **Circle** should extend **Shape** and implement **TwoDimensional**. Provide the appropriate attribute (`radius`) and constructor. Also, implement the `getArea()` method [Hint: PI * `radius`^{2}] and getP`erimeter`() method [Hint: 2 * PI * `radius`].

20. Go back to the **ExerciseShapes** class and create a new array that will hold three objects that are all 2 dimensional. It should look as follows:



```
TwoDimensional[] twoDs = new TwoDimensional[3];
```

21. Add a **Circle**, **Rectangle** and **Square** to the new array. Now, loop through the array calling the `getArea()` and getP`erimeter()` methods (print out the info they return). Does it work? Yes it does, because all of the objects are related by the interface that declares that `getArea()` and getP`erimeter()` must exist.