CPE 372 Object Oriented Analysis and Design
Exercise 5
Interfaces and Java Event Handling

1. Download the following Java source files from my **Exercise4** subdirectory under **LabSolutions.** (Or if you want, you can use your own code from Exercise 4 as a starting point.) Get the shape input files (**shapes1.txt,** etc.), also, or create your own for testing.

   ```
   Square.java
   Triangle.java
   Diamond.java
   Circle.java
   AbstractShape.java
   TextFileReader.java
   ShapeReader.java
   ShapeFileTester.java
   ```

2. Get `DrawingCanvas.java` and `FigureViewer.java` from the **Exercise3** solution directory.

3. Modify `ShapeFileTester.java` so that you draw the shapes as you read them. This means you will need to create a `FigureViewer` instance as Exercise 3. Alternatively, you can change `ShapeFileTester.java` itself to extend `JFrame` and embed the drawing canvas directly in the main program (see `SimpleVectorApp.java` in the CPE111Demos from the first lecture).

4. In the loop that reads the shapes, add code to draw each one as it is created. *You can remove the code printing the perimeter*.

5. Now add code that will allow you to select one shape by clicking with the mouse on the drawing canvas. When a shape is selected, you should *redraw it, filled*.

   To do this, you will need to change some class so that it implements the `MouseListener` interface. This could be the `FigureViewer` class or the `ShapeFileTester` class. You should think about which one would be best and why.

   Whichever class you select, that class needs to have code for all five methods in `MouseListener`. However, the bodies for most of these methods can be empty. For the `mouseClicked()` method, you must write code that will:

   - Get the X,Y position where the mouse was clicked.
   - If that position is inside a shape, clear the drawing canvas, then draw the selected shape.
   - If that position is not inside a shape, ignore it.

   Finding the shape inside of which the mouse was clicked is actually quite a difficult thing to do. Therefore, you should use a shortcut, using the so-called ***bounding box***. The bounding box of a shape is the smallest rectangle that encloses a shape. For a square, it will be the square itself. For the other shapes we have implemented, you need to do some simple calculations to find the bounding box corners.

You should add an abstract method to `AbstractShape`, for example:

```
public boolean inShape(int X, int Y);
```

Then implement the method for each specific shape. The method should return true if the passed point is inside the bounding box of the shape, otherwise false. (If the X of the point is between the minimum and maximum X values of the bounding box, and the Y of the point is between the minimum and maximum Y values of the bounding box, the point is inside.)

Hint: it's more efficient to calculate the bounding box of a shape only once and store the results in private member variables. In fact, you could make the bounding box be member data in `AbstractShape` . If you do this, then the `inShape()` method should be implemented in `AbstractShape` as well; it does not have to be abstract and overridden by each concrete class.

Note that a point could be inside multiple shapes (because their bounding boxes overlap). For simplicity, you can make your code stop after finding the first shape. (This means you will need to iterate through all the shapes in the `allFigures` list, calling `inShape()` and stopping for the first shape that returns true.) Alternatively, you can find and fill *all* the shapes that are selected.

Think about encapsulation and information hiding when you write these methods. Where should the code to find the clicked shape be located? Do not expose more information than absolutely necessary!

5. Add a button to the `FigureViewer` that will redraw all the shapes. Note that this class already extends the `ActionListener` interface to handle the "Clear" and "Exit" buttons. This will allow you to redraw all the shapes and try selecting different ones.

Upload **all** the Java source files, whether you modified them or not, by putting them in a ZIP or TAR file. DO NOT SUBMIT RAR FILES. If you change a source file, be sure to add code to the header comment with your name, ID and the date, explaining briefly what you did.