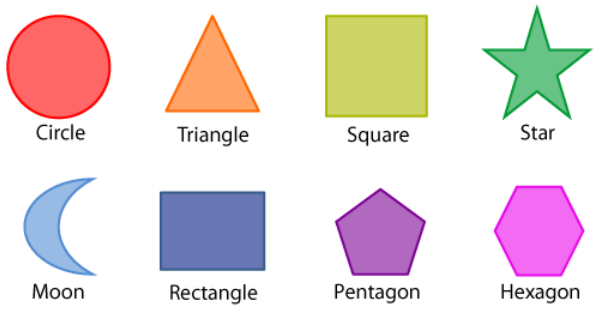
Applying the Adapter Pattern

Re-submit Assignment

Due Mar 21 by 11:59pm **Points** 100 **Submitting** a website url

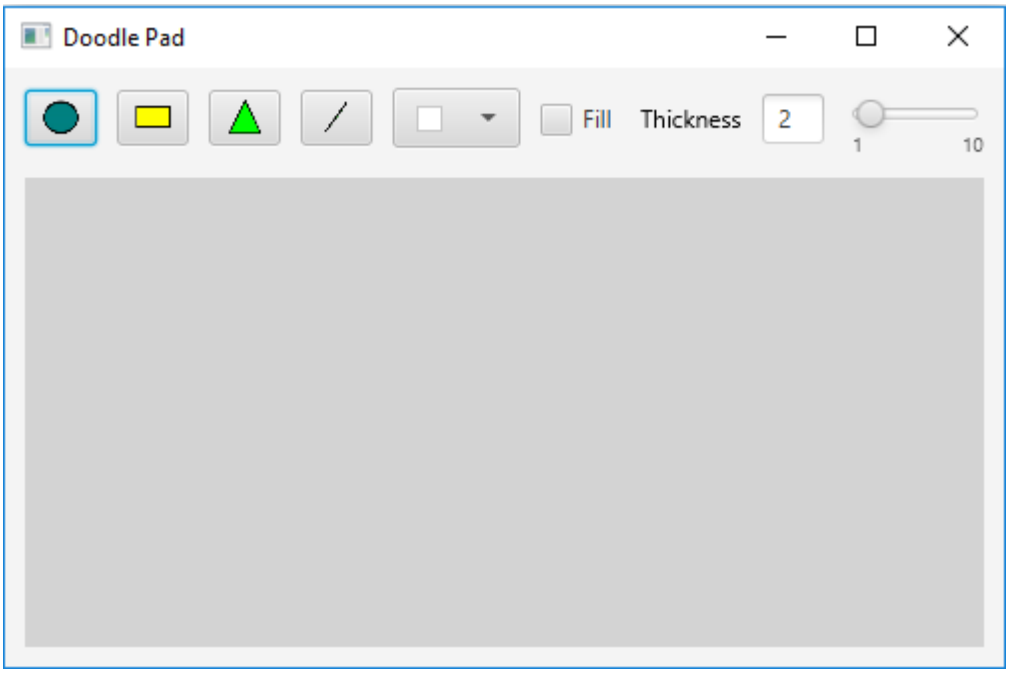
Applying the Adapter Pattern

This assignment explores one way to handle incompatible interfaces. Along the way you will build a small graphics program that allows a user to stamp simple shapes on a Canvas. To accomplish this you must first take several existing shape classes and make them compatible with your program.

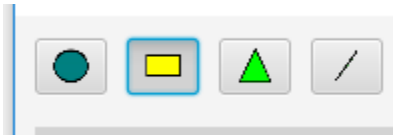


The Doodle Pad Program

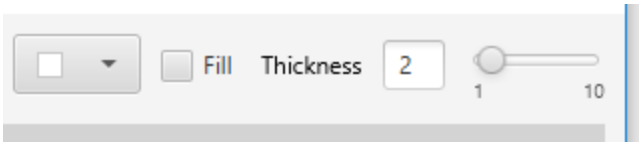
Below is a screenshot of a Java FX application that you will build.



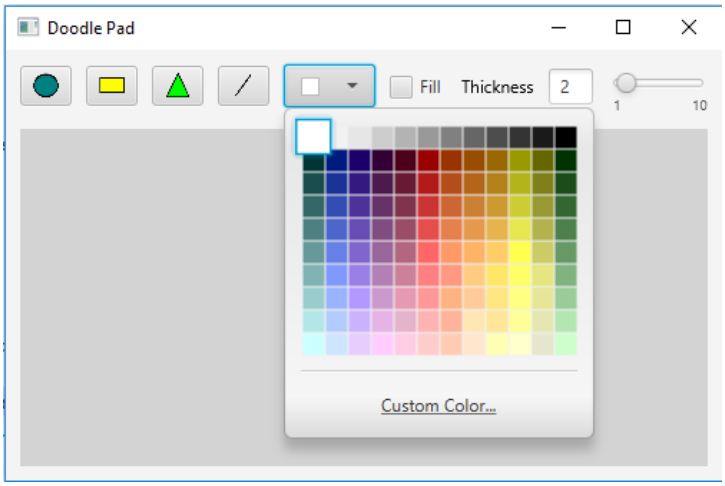
At the top of the application is several ToggleButton controls that allow a user to select one of four shapes.



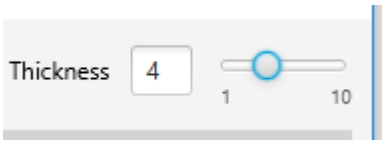
After a shape has been chosen the user can then choose a color, fill-mode and stroke thickness for the shape.



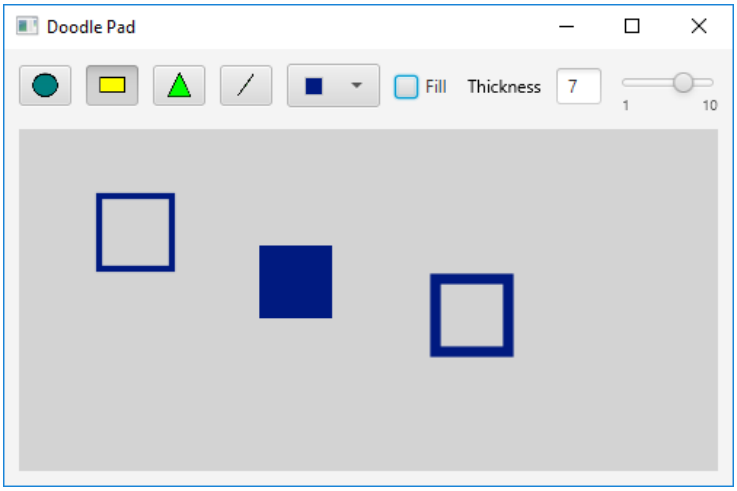
When a user clicks on the first control a drop-down is displayed that allows the user to select a color. This control is provided for you from the Java FX libraries.

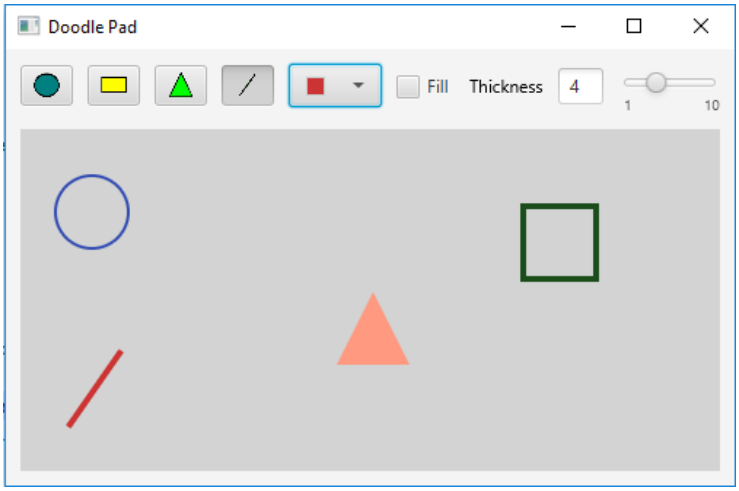


The next control determine whether the shape drawn on the UI will be filled with a color, or just an outline. The last control determines how thick of a line to draw when using an outline.



After all settings have been chosen, a user can click on the Canvas control located in the bottom part of the interface (with the gray background) and the associated shape will be shown at the mouse's location.





Getting Started

Part of your project files have already been written for you, found here. You have been given a class that stores shape objects and will draw them to a GraphicsContext object. This class is described below:

SavedShapes

- shapes : List<IShape>
+ SavedShapes() + update(shapeToUpdate : IShape, thickness : double, color : Color, filled : boolean) : boolean + drawShapes(graphics : GraphicsContext) : void

Notice that this class stores a list of IShape objects.

<<interface>> **IShape**
+ setThickness(value : double) : IShape + setColor(value : Color) : IShape + setFilled(value : boolean) : IShape + double getX() : double + double getY() : double + double getThickness() : double + Color getColor() : Color + boolean getFilled() : boolean + drawShape(GraphicsContext graphics) : void

You have also been given a set of pre-existing shape classes that are incompatible with the SavedShapes class described above. i.e. none of these classes use the IShape interface, or even contain the methods located in the interface

Shape
- x : double - y : double - thickness : double - color : Color
+ Shape(x : double, y : double, thickness : double, color : Color) + getX() : double + getY() : double + getThickness() : double + getColor() : Color

+ setX(x : double) : void + setY(y : double) : void + setThickness(thickness : double) : void + toString() : String

Rectangle extends Shape
- width : double - height : double
+ getWidth() : double + getHeight() : double + toString() : String

Triangle extends Rectangle
+ Triangle(x : double, y : double, width : double, length : double, thickness : double, color : Color)

Line extends Shape
- x2 : double - y2 : double
+ Line(x : double, y : double, x2 : double, y2 : double, thickness : double, color : Color) + getX2() : double + getY2() : double + toString() : String

Circle extends Shape
- radius : double
+ Circle(radius : double, x : double, y : double, thickness : double, color : Color) + getRadius() : double + toString() : String

Writing the Adapters

Your program will make direct use of the SavedShapes class to store shapes that are added to the user interface. Our goal is to use the existing shape classes seen above, but to make them compatible with our SavedShapes class.

- According to the open-closed principle we should not begin editing these existing classes.
- Instead create an adapter class for each of the four classes: Oval, Rectangle, Line and Triangle.
- Your adapter class will store an instance of one of the shape classes and implement the IShape interface.
- This directly follows our use of the Adapter pattern in class.

Below is a description of the methods from the IShape interface

Method Chaining

The first three methods of the interface use a technique called method chaining. A method can be called upon an object changing some state within the object and then the object itself is returned from the method. This can lead to some very concise code segments that perform several operations at once. The following example line of code can be found in the SavedShapes class:

```
shapeToUpdate.setThickness(thickness).setColor(color).setFilled(filled);
```

After each call to the methods setThickness(), setColor() and setFilled() the object in the shapeToUpdate variable is returned and a new method is invoked on the same object. When implementing these methods you should be returning an IShape object, not one of the shape classes provided.

Note: This should look familiar to how the String class is used in Java.
Note: This is also how the jQuery library is written.

Getters

These getters return the location and settings for drawing an instance of a shape.

The drawShape() Method

This method is responsible for drawing the shape to a GraphicsContext object. This context has methods for drawing primitive shapes to an associated control. In our case, the context should be taken from the Canvas.getGraphicsContext2D() method.

Finishing the Doodle Pad Program

After you have created compatible adapters you should focus on building your user interface and writing all event handlers. Use the following suggestions as guidelines:

- Each of the buttons used are ToggleButton objects. These behave similar to RadioButton controls and require an associated ToggleGroup object.
- The color drop-down is a ColorPicker control. To use the control simply instantiate a ColorPicker object and add it as a control to your interface. The selected color can always be retrieved from the control using ColorPicker.getValue(). To learn more about the ColorPicker control, read the following: **http://docs.oracle.com/javafx/2/ui_controls/color-picker.htm (http://docs.oracle.com/javafx/2/ui_controls/color-picker.htm)**
- The thickness setting uses a Slider control which varies from 1 to 10. I used a ChangeListener event handler to update the text field nearby with the current value of the slider. To learn more about the slider control, please read the following: **http://docs.oracle.com/javafx/2/ui_controls/slider.htm (http://docs.oracle.com/javafx/2/ui_controls/slider.htm)**
- Since we are not defining the width and height of shapes in our UI, pick sensible default values.
- Similarly, when drawing a line shape using the mouse location for one point and a random location for the other point.

Restrictions

- No changes may be made to any of the classes you have been given.
- Anyone ignoring this restriction will receive an immediate zero.

Submission

- All projects must be completed through the IntelliJ IDE
- Your project should be a Maven project with appropriate settings
- Your project files should be saved to a private GIT repository which is shared with myself
- All files must have proper documentation (comments and full Javadocs)
- All files must follow our style guidelines

Extra Credit (5 points)

Add a new type of shape to your toggle buttons. Points will be awarded based on effort. For example, it would be interesting to enter a polygon or star shape.

Applying the Adapter Pattern Rubrics

Criteria	Ratings		Pts
An adapter is written for the Rectangle class that is compatible with the SavedShapes class.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
An adapter is written for the Oval class that is compatible with the SavedShapes class.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
An adapter is written for the Triangle class that is compatible with the SavedShapes class.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
An adapter is written for the Line class that is compatible with the SavedShapes class.	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
All drawShape() methods correctly draw their corresponding shapes to the graphics context.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
All controls for adding shapes and their settings are provided at the top of a user interface.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
The user is able to use the controls on your interface to add different types of shapes to a Canvas.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
IntelliJ project is set up correctly as a Maven project. All project files are saved to GIT in a private repository.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Full Javadocs are provided for all classes and public methods.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
All code follows the style guide provided.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
			Total Points: 100.0