**ITIS 1213 – Assignment 4**

The main idea for this assignment is to show that you can work with basic GUI elements and combine them with object-oriented design.

**Prep Work**

Download the Assignment4DrawingProgram.zip project from Moodle. Import this project into NetBeans. Open it and run the project to see how it works.

This project runs a very simple drawing program, that allows you to draw lines and rectangles and allows you to switch between two outline colors and three fill colors. The UML diagram for this program is below. You will be making significant improvements to the design of the program as well as adding a bunch of new features to this program.



**Part 1 – Add Shape class and inheritance (20%)**

The project contains a ‘Line’ class and a ‘Rectangle’ class, both of which are quite similar. Shapes are drawn by the user pressing down, dragging and then releasing the mouse cursor, which generates two sets of coordinates. You can think of these coordinates as defining a bounding box in which a shape is drawn.

One of the problems with the code you are given is due to the fact that it keeps track of lines and rectangles separately. If you look inside the DrawingModel class, you’ll note that there are separate ArrayLists to hold Line objects and Rectangle objects. If you look at the paintComponent method in DrawingCanvas, you’ll see that we iterate through each of these ArrayLists, one at a time, drawing all the lines first and then all the rectangles. Try to draw a picture in the program that has lines on top of rectangles and you’ll see that it is impossible. That’s because all the lines are drawn first (even if the user added them last). You could switch the order and draw all the rectangles first (try it!), but then you wouldn’t be able to have any rectangles in front of lines.

So, the idea in this part is to think about how to generalize this code. You should create an abstract Shape class that contains the stuff that would be similar across all shapes, and change Line and Rectangle so that they extend Shape. Then, in the DrawingModel class, you should be able to create a single array of Shape objects and add both lines and rectangles to that list. In your DrawingCanvas class, you should be able to iterate through this single ArrayList of shapes and draw all the shapes, in the order they were added.

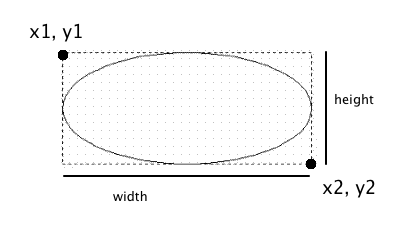
Hints:

1. Draw a partial UML diagram to help you think through what should be in the shape class and what should be in the subclasses. It may help to think that most shapes are going to be more like rectangles with a line and fill color and that the Line shape is probably a bit of a special case.
2. Your DrawingModel class should become simplified, since you don’t have to have methods to handle each type of shape and to handle each ArrayList separately.
3. You don’t need to put the static version of the drawing method in the Shape class – you can’t have a method that is both static and abstract. Each of your shape sub-classes should just keep its own static drawing method.

This part of the assignment isn’t super fun, because you’re not adding any features or functionality to the program. When you are done with this code redesign the DrawingProgram should compile and run and (from the user’s perspective) it should work exactly the same as when you got it. But, you are making the program design much more general, which will make Parts 2 and 3 much easier.

**Part 2 – Add two more shapes (20%)**

Now that your program uses inheritance, it should be really easy to add two more shapes to the program. For each shape all you really have to do is figure out the drawing code. You can add circle, triangle, hexagon, etc. For all of these shapes, remember to think about the user specifying the bounding box of the shape, like this:



Here is the list of things you’ll need to do to add each new shape:

1. Create a class for that shape, and implement the draw(…) method.
2. Add the shape to the enumerated ShapeType list.
3. Add cases to the switch statements in both the DrawingCanvas.java paintComponent(…) method and the drawingCanvasMouseReleased(…) method in DrawingWindow.java.
4. Test it all to make sure it works!

**Part 3 – Add undo and redo functionality (20%)**

We all make mistakes! What if I draw a shape and I don’t like it? I should be able to get rid of it, right? We want to add Undo and Redo functionality to this program. To do this, there are a few steps:

1. Add two buttons to the toolbar for Undo and Redo.
2. Add a new ArrayList to your DrawingModel class to hold the shapes that you undo (in case you want to Redo them).
3. Add an undo method to your DrawingModel class that takes the last shape out of your shape list and puts it into your undone shapes list.
4. Add a redo method to your DrawingModel class that takes the last shape undone, and puts it back on the shape list.
5. Set up the button listeners in your DrawingWindow so that they call the undo/redo methods appropriately and update the canvas.

Make sure this code is robust, and doesn’t try to undo shapes when there are no shapes left, and similarly doesn’t try to redo more shapes than have been undone. You can use the ArrayList isEmpty() method to check your lists.

**Part 4 – Add real color pickers for line color and fill color (20%)**

Right now the drawing program has very limited color functionality and using toggle buttons for colors is really poor (not scalable!) design. One of the nice things about using a mature programming language like Java is that it already has lots of built in functionality you can take advantage of. For instance, there is a Swing color picker that you can use so that your users can pick any color they want. For this part of the Assignment you need to add ColorPicker functionality. Follow these steps:

1. First, add a new JFrame form and call it something like ColorPickerWindow.
2. Delete the main method that NetBeans automatically creates for you in the frame.
3. Drag the Swing ColorChooser element onto this window.
4. Stretch out the window and add two buttons to the bottom, one that says Choose, and another that says Cancel.
5. Pass the DrawingModel to this window by modifying the constructor
6. Add a new button to your DrawingWindow toolbar that is labeled Fill Color.
7. In the listener for this Fill Color button, create a new ColorPickerWindow (passing in the DrawingModel), and set the window to visible. Test this to make sure that the window is showing up.
8. Right now the window doesn’t do anything and you can’t even make it go away. Go back to the ColorPickerWindow and add code for the Cancel button. This could should simply call this.dispose() to make the ColorPickerWindow go away.
9. Now, add code to the Choose button, to get the selected color (the colorChooser that you added to the window has a getColor() method) and then set that fill color in your DrawingModel. After doing this, the ColorPickerWindow should be disposed.
10. Test this to make sure it works for changing the current fillColor.

Once you have this working for fill color, you need to make it work for Line Color. You should reuse the same DrawingWindow and follow the procedure above. However, the trick is that your Choose button code will need to know whether the chosen color should be applied to the fill color in your model or the line color in your model. You’ll need to pass something in to the DrawingWindow constructor to tell it which kind of color is being chosen. When you are done, test to make sure that you can now separately choose different fill colors and line colors. Then make sure to get rid of all the old Toggle buttons in the Drawing Window.

**Possible Bonus Work:**

* Add line width functionality
* Add ability to change background color of canvas
* Show a dashed line to represent the bounding box during interactive drawing
* Change the undo and redo buttons to be appropriately enabled or disabled depending on whether there are shapes to undo/redo

**Coding Style – (15%)**

This grade is awarded for proper coding style. This includes:

* Your name in comments at the top of every Java file you edit
* Appropriate method, variable, field, object and class names
* Proper indentation
* Good JavaDoc and within method commenting (explains what code is doing)
* Well-organized, elegant solutions for each part

Files to submit:

* Your project (in a ZIP file format)
* Assignment Contract

**Appendix III – Grading Rubric (see the up to date rubric on Moodle)**