CS1331 HW10B

Paint Program + Animation

Due Friday November 30th

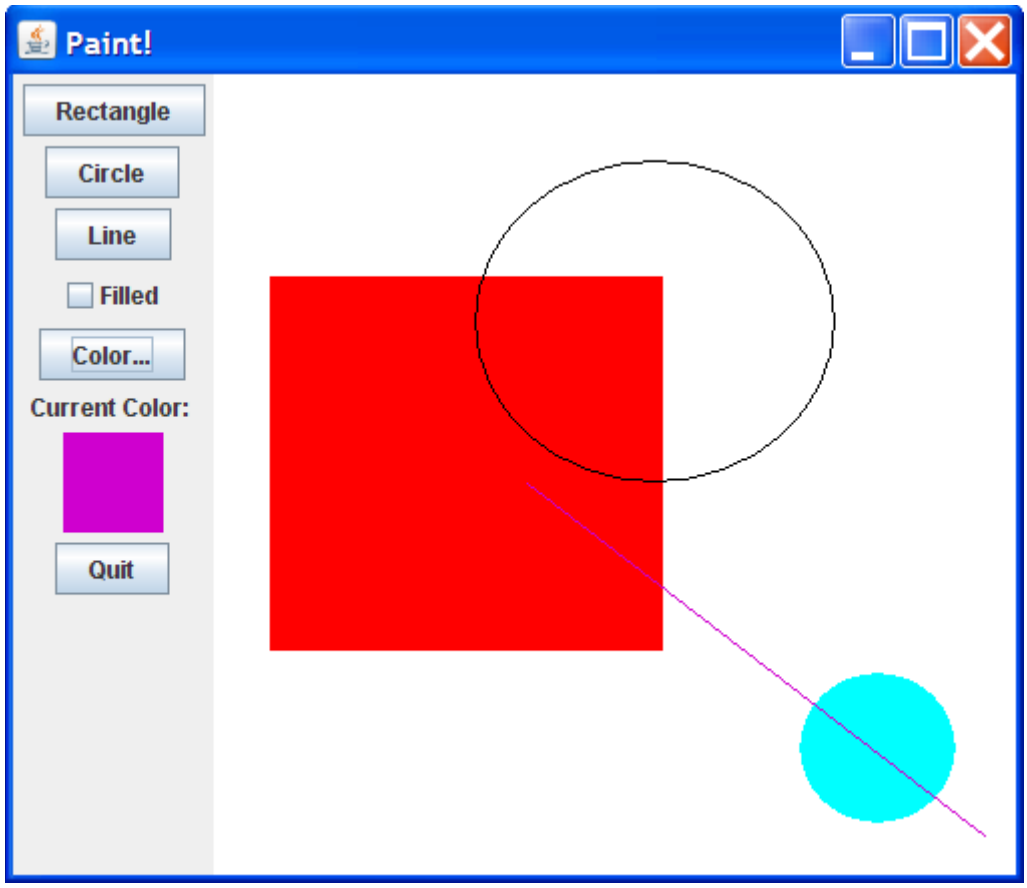
Introduction

This homework will cover inheritance, abstract, ArrayLists, `MouseListener`, and `MouseMotionListener`.

You will be creating a program that can paint several different shapes on the screen by clicking and dragging the mouse.

Be sure to name your classes as required by the instructions. Also be sure to use good coding style and indentation, and to use appropriate and descriptive variable names.

Do not forget about the javadocing and commenting detailed in homeworks 2 and 3.


Paint!
Rectangle
Circle
Line
Filled
Color...
Current Color:
Quit

Requirements

For this homework, you are allowed to use any methods you feel are necessary. However, you must meet these requirements:

1. There should be an abstract class called *`Shape`*, from which all the other shapes extend from.
2. Provide `Line`, `Rectangle`, and `Oval` options for the user to draw.
3. Provide a way to draw filled and unfilled shapes (with the exception of line).
4. When drawing a shape, the shape should resize as the user drags the mouse. That is, as you drag the mouse, the shape is continually resized on the screen.
5. Provide a way to choose different colors.
6. Provide a display to show the current color.
7. Create a single Generic ArrayList to hold all the shapes, and **only** shapes.
8. Handle refresh and resizing the window (use an appropriate layout).
9. Provide a quit button that exits the program.
10. Put the main method in a class called *`PaintGUI`*.
11. Use proper object oriented programming (don't put everything in one class or method, and use proper visibility and protection on your variables!).

Class Hierarchy Outline

- **This is only an outline to get you started.** You may need to provide **additional** instance variables and think about constructor implementation.
- While this is an outline, the classes and implementations listed below are **requirements.**

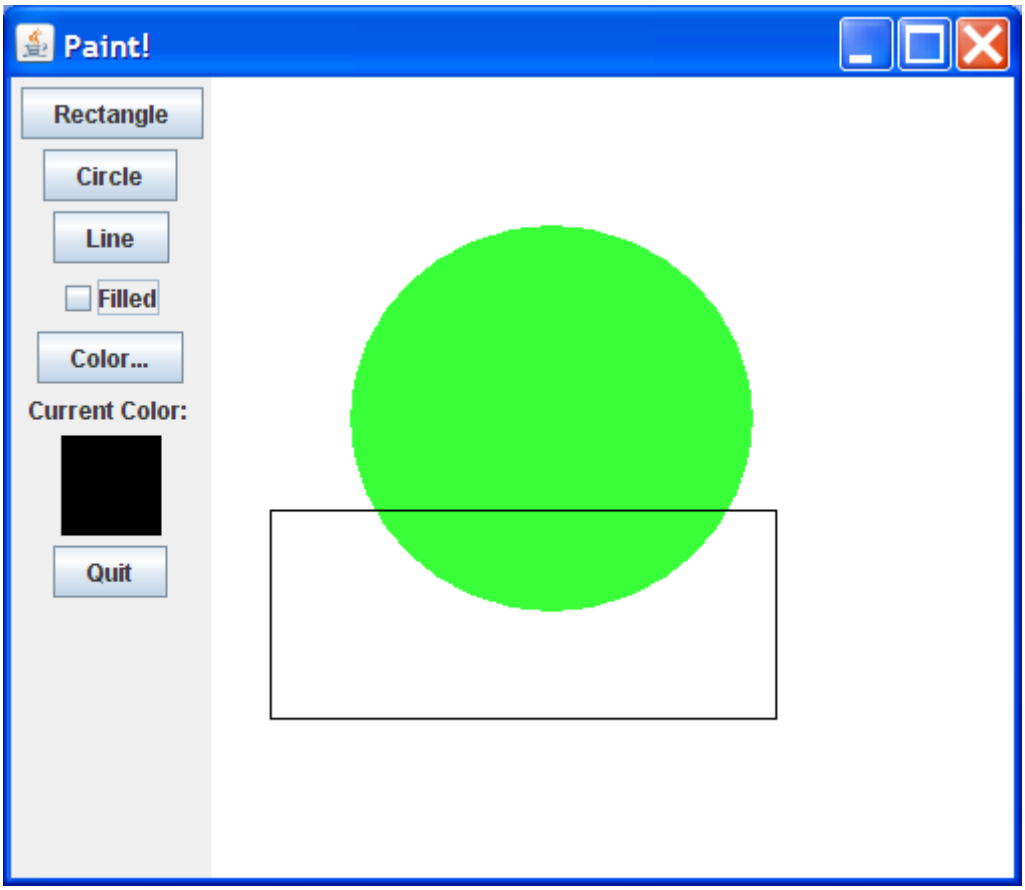
- Abstract class *`Shape`*: The root of all the other shapes, it holds the instance data for the shapes and has an abstract draw method. It also contains getters and setters for its instance variables.
- Subclasses of *`Shape`*: These are the shapes which can be drawn on the screen. Contains the specific implementation for how each shape is drawn.

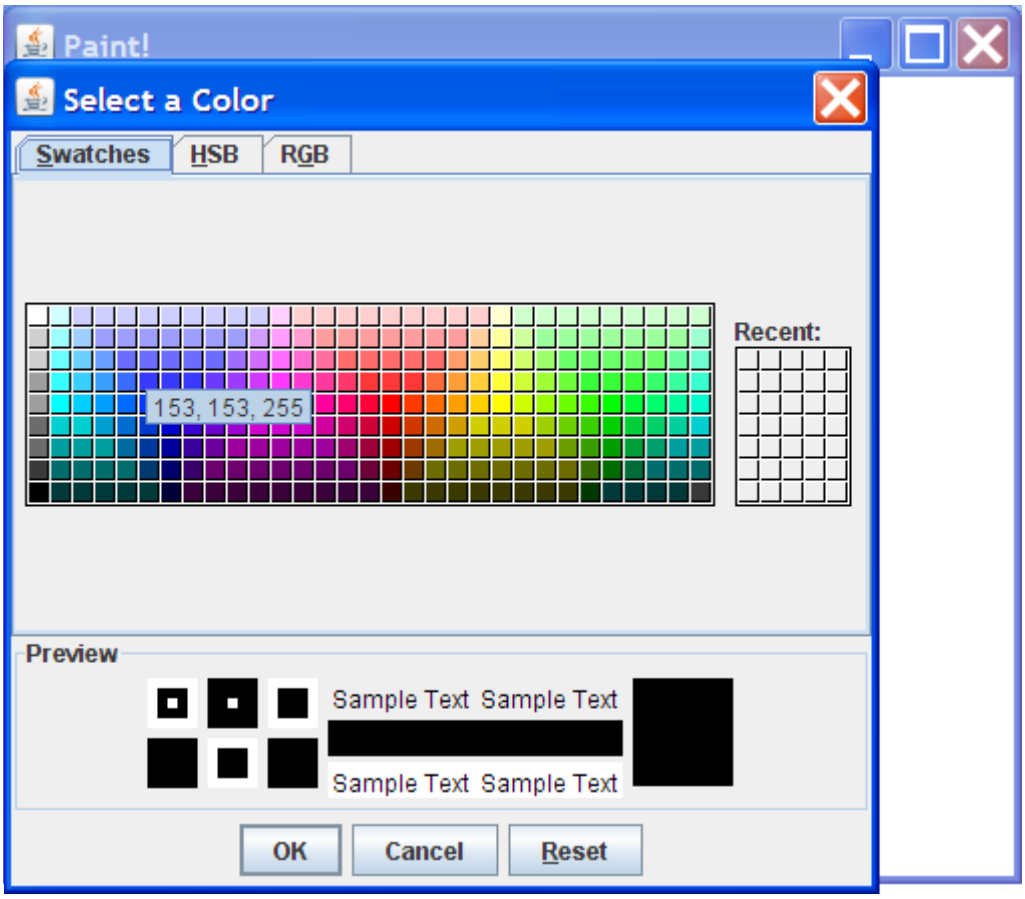
1. Create a class hierarchy to represent several forms of shapes:
- **Shape class**: This class is the super class for the other shapes. It holds 2 sets of coordinates (x,y pairs) which represent the upper-left and bottom-right corners of the shape. It should:
    - Be abstract
    - Have two instance variables, int x1 and int y1 representing the start location of the shape (top-left). These should have appropriate visibility.
    - Have two instance variables, int x2 and int y2. These should also have *appropriate visibility*, taking into account that other classes will be subclassing this one.
    - Have a constructor that takes in at least the initial x1 and y1 position indicating the start of the shape.
    - Have a way to update the bottom-right corner (x2, y2) of the shape.
    - Have an abstract `draw` method that takes in a Graphics object.
- **Rectangle class**: This class represents a rectangle. It should:
    - Extend `Shape`.

    - Have a constructor that takes in the x1 and y1 location. It should use the super constructor to set these values.
    - You may need to modify the constructor parameters and instance variables, ie to deal with filled and unfilled counterparts.
    - Override the abstract methods.
    - Remember that subclasses inherit all the variables and methods of the parent class (Shape).
- **Oval class**: This class represents a Oval. It should:
    - Extend `Shape`.
    - Have a constructor that takes in the x1 and y1 location. It should use the super constructor.
    - You may need to modify the constructor parameters and instance variables to deal with filled and unfilled counterparts.
    - Override the abstract methods.
- **Line class**: This class represents a line. It should:
    - Extend `Shape`.
    - Have a constructor that takes in the x1 and y1 location. It should use the super constructor.
    - Override the abstract methods.

1. *`InputPanel`* class: A *`JPanel`* than holds the buttons and how the shape is going to be drawn. Contains *`ActionListener`***s** for the buttons and getters for the current color, whether the shape is filled, and what shape is to be drawn.
2. *`PaintPanel`* class: A *`JPanel`* that holds all the shapes that have been drawn. Contains a *`paintComponent`* method to draw the shapes, and a *`MouseAdapter`* to handle dragging and releasing the mouse. Needs to be able to communicate with the `InputPanel`.

More Pictures


Paint!
Rectangle
Circle
Line
Filled
Color...
Current Color:
Quit

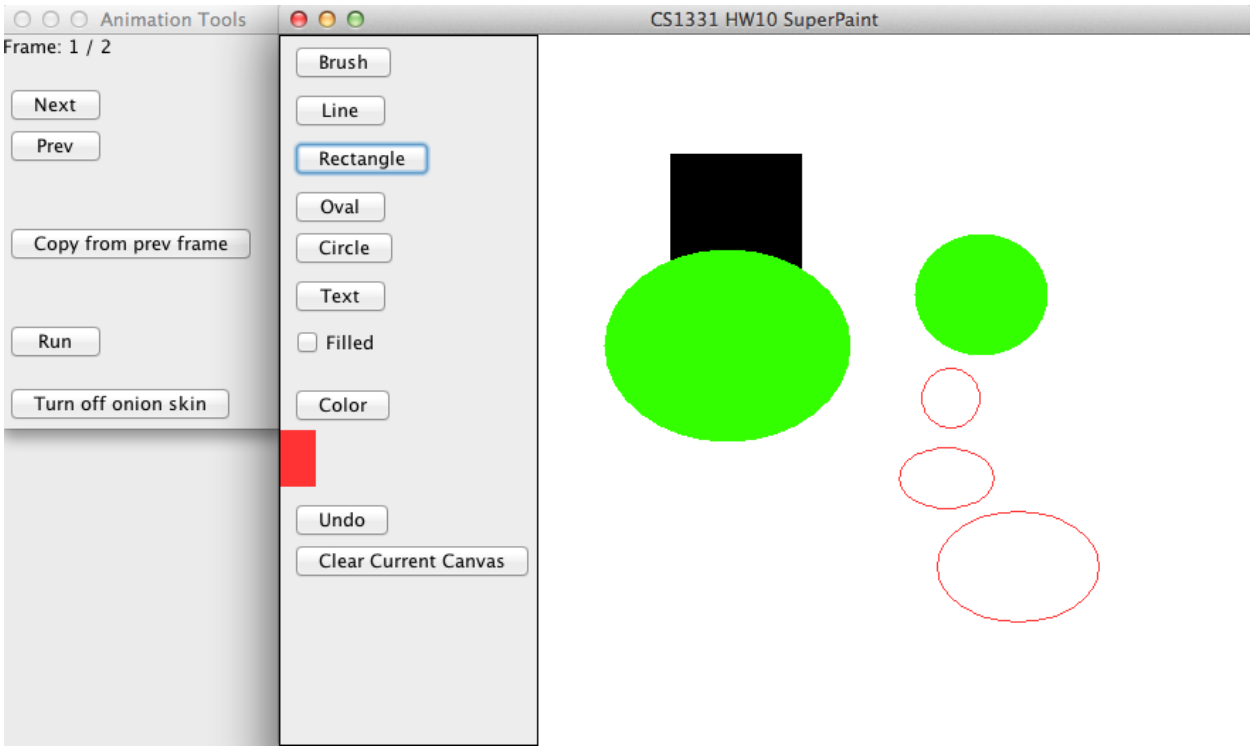


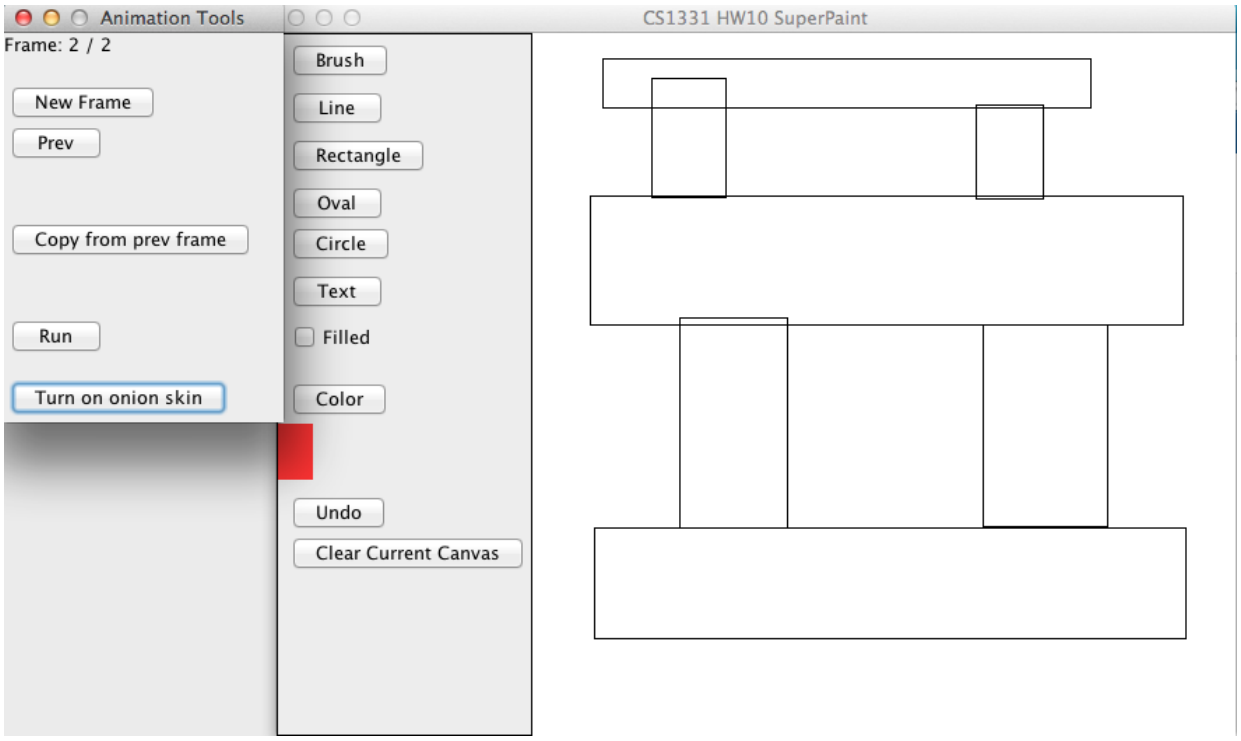
Part 2- Animation

For part 2, you will be adding animation functionality to your paint applet. You must give the user the ability to:

1. Add slides/frames to the animation
2. Navigate to different slides
3. Run the slides quickly in an animation form
4. Stop the animation if it is running

You can make all of these functions accessible through buttons on the control panel. The program should list what frame the user is currently on, and how many total frames are in the animation. Here is an *example* of a submission that has an Animation panel (on the left, in a separate JFrame. Also note that this example has a "Next Frame" button that changes to a "New Frame" button when the user is at the final slide.):



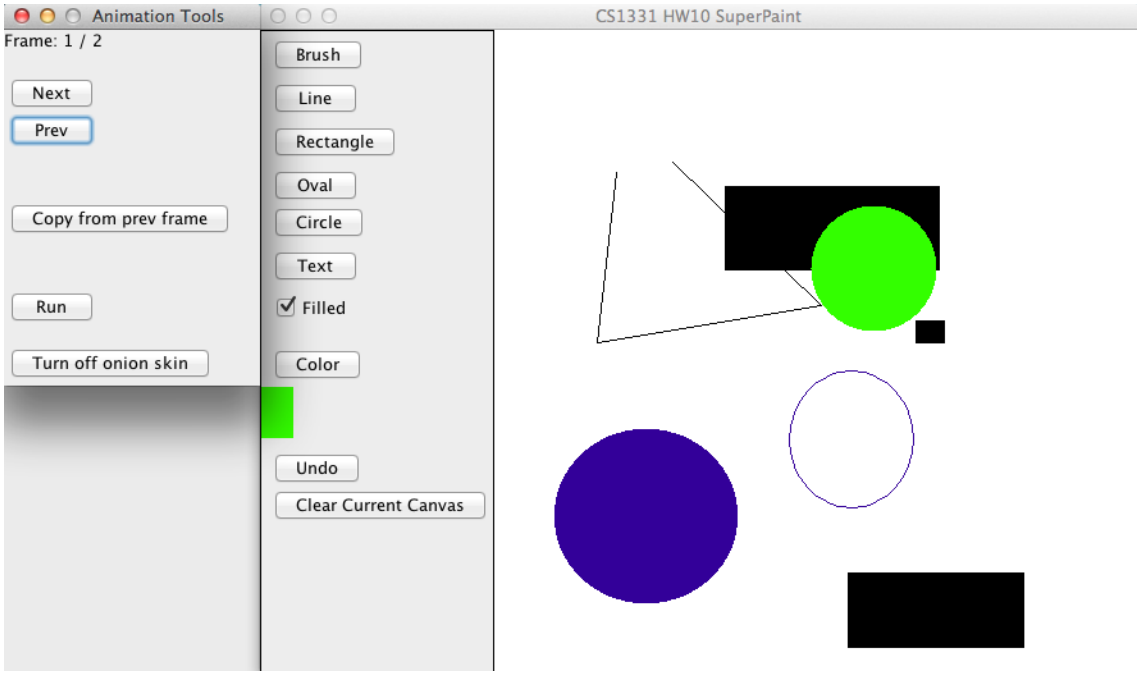


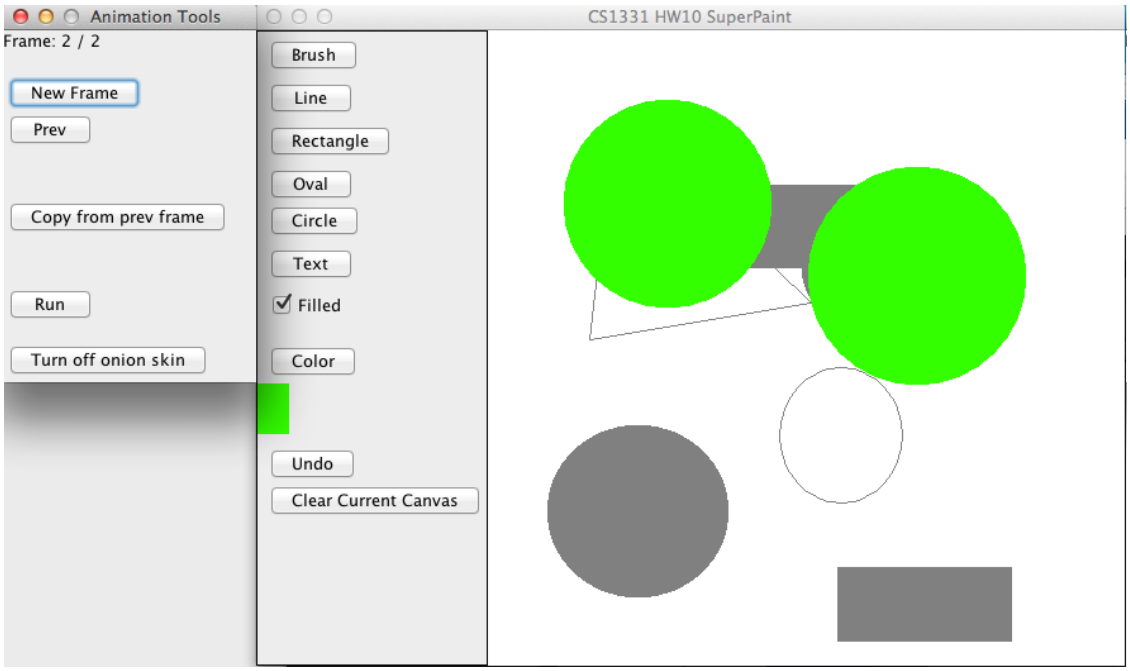
This example also has some extra functionality (like a Clear button, Brush, Undo, and Onion Skin) which might make testing your application easier. You aren't required to have them, but adding them can be a good programming exercise.

Extra Credit

(+10 maximum)

- (+5) Save/load animations.
- (+5) Have a toggle-able feature called onion skin, where the shapes in the previous frame show up as transparent shapes in the current slide, and cannot be edited there. See below:





Hints and other notes

- Your program only needs to handle “click-and-drags” to the bottom and right of the initial click point. (Although brownie points for those that can do both directions)
- `JColorChooser` provides several useful methods, including `showDialog`. See http://download.oracle.com/javase/6/docs/api/javax/swing/JColorChooser.html for more details.
- Consider using some sort of component that allows you to select between two options for the filled/unfilled option. The Java API is your friend.
- You might need to use aspects of both `MouseListener` and `MouseMotionListener` to accomplish the desired draw effects.
- The `getPoint` method can be called on the `MouseEvent` to determine where the mouse is.
- If you set up your hierarchy correctly, you should be able to add all the shapes to a single ArrayList of generic type Shape. From there, you can call a draw method on every shape. In other words, if you declare an ArrayList<Shape>, it can hold Rectangles, Ovals, and Lines. You can call *any* of Shape's methods on the objects retrieved from the list.
- A paintComponent(Graphics g) method inside a JPanel allows you to draw on a JPanel similar to a paint method in applets.
- For two Panels to be able to communicate with each other, consider having an instance variable for the other panel in one of the panels classes. You then create the other panel first, and pass it's reference in as a parameter to the constructor of the second panel.
    - To simplify that, consider creating an InputPanel first, and then passing that reference to the constructor of the PaintPanel and saving it.
    - Or vice-versa.
- You should not be creating a new InputPanel inside PaintPanel. You need to work with the same InputPanel added to the main container by reference passing.
- You need a container to hold the two panels. Know that a JFrame can hold more than one JPanel, and uses BorderLayout by default.
- Think of how you can use x1 and x2 to calculate width, and y1 and y2 to calculate height so that you might utilize the Graphics drawing methods you learned in the beginning of the semester.

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have *submitted* and not just saved as draft.

- Shape.java
- Rectangle.java
- Oval.java
- Line.java
- PaintGUI.java
- Any panel classes (ie InputPanel, PaintPanel)
- Any other files needed to run your program

All .java files should have a descriptive javadoc comment.

Verify the Success of Your HW Turn-In

Practice "safe submission"! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
    a. It helps insure that you turn in the correct files.
    b. It helps you realize if you omit a file or files.**

    (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    c. Helps find last minute causes of files not compiling and/or running.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect.