Object Oriented Programming

Lab 2 Report  
Part 1

Drawing and painting application

Prepared by

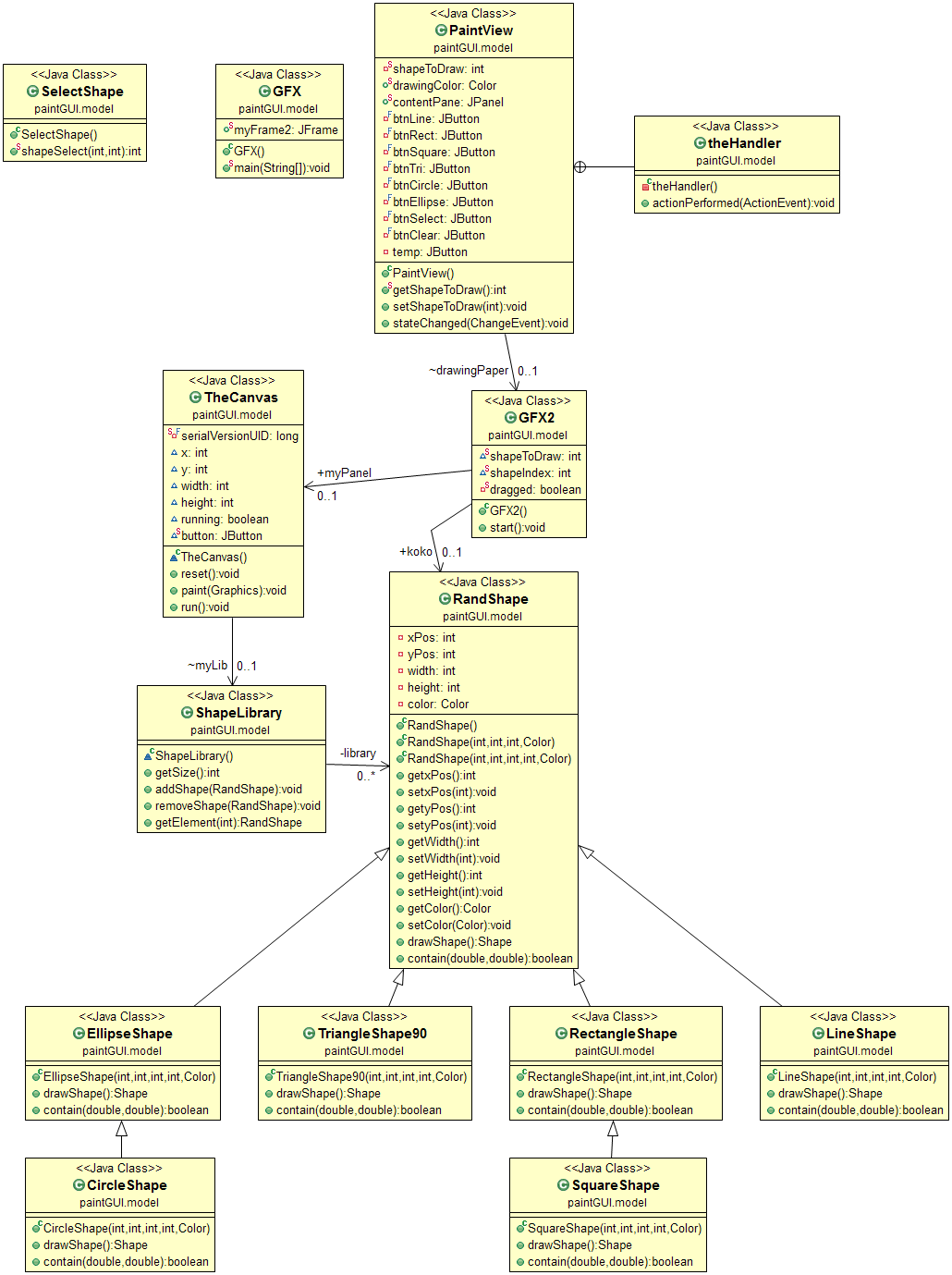
Amr Saleh Mohammed Ali (37)

Ahmed Abd El-Wahed El-Khatib (7)

Table of contents:

1. UML class diagram.
2. Listing of classes and a brief description of each class.
3. Sneak peek into the GUI.
4. User guide manual.

\*NOTE: Any assumptions will be indicated.

1.  UML class diagram:
2. Listing of classes:
3. RandShape:

This class is the main parent for all shapes included in the program, whether they are included internally or imported by the user in the future.

This class has 5 private instance variables (for now). They describe the top left point of the shape, its length and height (sometimes the bottom right point) and the color. Setters and getters are imminent.

3 constructors were implemented. Firstly, you can see an empty constructor. The purpose of this constructor is still unclear, and might prove useless in the end. Second constructor is used for symmetric shapes, and will construct an object with the same width and height. The third one is used for general shapes.

Lastly, 2 functions were implemented. The first one is drawShape(), which returns and object of the type Shape (imported internally) and generally, it returns null. The second one is contain(double x, double y). It returns true if the point (x,y) exists inside the shape drawn in the canvas. Generally, it returns false.

1. LineShape extends RandShape:

This class constructs line shapes and is able to return drawable line Shape objects.

As expected, there is a constructor which overrides the third constructor in RandShape and constructs a RandShape object with parameters (first point x value, first point y value, second point x value, second point y value).

Then there is the function contain(x,y) overriding the parent method, which constructs a new Shape object with the line parameters and checks whether the coordinates exist very close to it (5 pixels precision). This is used to select the shape after it was drawn in the canvas. The 5 pixels precision makes it easier to select the line without getting the user frustrated.

The last function , which is drawShape(), overrides the parent method. It returns a Shape which can be drawn as a line using the Graphics2D class during runtime.

\*NOTE: Most $Shape classes share the same aspects of this class, so there will be no detailed description for the upcoming 5 classes.

1. EllipseShape extends RandShape:

There is a constructor which overrides the third parent constructor but this time, it constructs the object with parameters (top left x coordinate, top left y coordinate, width, height).

The contain method checks whether the user clicked inside the ellipse area.

The drawShape method returns a Shape object that can be drawn as an ellipse.

1. CircleShape extends RandShape:

Again, there is a constructor which overrides the third parent constructor but this time, it constructs the object with parameters (top left x coordinate, top left y coordinate, diameter, diameter).

The contain method checks whether the user clicked inside the circle area.

The drawShape method returns a Shape object that can be drawn as a circle.

1. RectangleShape extends RandShape:

Bah, there is a constructor which overrides the third parent constructor but this time, (boring isn’t it?) it constructs the object with parameters (top left x coordinate, top left y coordinate, width, height).

The contain method checks whether the user clicked inside the rectangle area.

The drawShape method returns a Shape object that can be drawn as a rectangle.

1. SquareShape extends RandShape:

Again, there is a constructor which overrides the third parent constructor but this time, it constructs the object with parameters (top left x coordinate, top left y coordinate, width, width).

The contain method checks whether the user clicked inside the square area.

The drawShape method returns a Shape object that can be drawn as a square.

\*NOTE: We’re almost done with the boring part.

1. TriangleShape90 extends RandShape:

Alright, new stuff is here. The assumption here is that the user will want to draw a right angled triangle. The constructor here overrides the third parent constructor but this time, it constructs the object with parameters (top left x coordinate, top left y coordinate, width (which is the horizontal side of the angle), height(which is the vertical side of the angle).

The contain method checks whether the user clicked inside the triangle area. (determined using polygons)

The drawShape method returns a Shape object that can be drawn as a triangle, using polygon.

I hope the copy-paste part wasn’t too boring to read. Fear no more, though. The next part is interactive.

1. ShapeLibrary:

This constructs our storage place. There is an instance variable of type ArrayList which stores RandShapes. There is a constructor which initializes our ArrayList. There are methods to add or remove objects, get an object by its order in the Arraylist and get the size of it.

1. TheCanvas extends JPanel implements Runnable:

This is a class that allows us to construct objects of its type and paint inside them. Never been so intense huh? There are a number of instance variables which are later passed on as parameters to create the RandShape family. Of course they’re the x coordinate, y coordinate, width and height (generally). There is a flag to check whether changes are being made to the canvas or not, and a declaration of a ShapeLibrary object.

Then, there is a constructor for TheCanvas, which changes the flag to true, and instantiates the ShapeLibrary object.

Next, we meet our paint(Graphics g) method. It takes in an object of type Graphics (internal) and calls its parent method. Then we create a Graphics2D object and assign it the value of the Graphics object, after casting it to Graphics2D (sounds complicated).

We set some attributes to our object such as stroke, Anti Aliasing and color. Next, we iterate on our ShapeLibrary object, drawing every RandShape that was once created and not deleted. Then, we draw the object that is being drawn on the screen but not created yet (of course if it existed) and finally, we draw the selection shape if a shape was selected on the screen.

Then, comes a function reset(). All it has to do is to initialize TheCanvas parameters and reset the shapeIndex value to indicate that nothing should be drawn on the screen. It’s called when clearing the canvas.

Last but not least, the run() method. This method runs forever to check whether the canvas needs to be repainted and sets the bounds of TheCanvas object. It uses javax.Swing.Timer to do the infinite loop.

1. GFX2:

This class controls most of our mouse actions and creates the RandShapes that are later painted in TheCanvas class. There are a few variables such as int shapeToDraw, which indicates which shape should be drawn while dragging / after releasing mouse. There’s the shapeIndex which tells which shape in the ArrayList iss currently selected. There’s a dragging flag to know if I dragged after pressing or not. Lastly, there are 2 declarations for objects of types RandShape and TheCanvas.

The constructor here instantiates TheCanvas object and starts listening to the mouse. In case the user clicked the select button and clicked on an object, it’s selected. Otherwise, nothing happens. The object selection detects mouse coordinates and loops the ArrayList backwards to detect the most recent object drawn which contains the coordinates of the mouse. It uses the contain methods inside the RandShape children.

In case the user clicked a button for drawing a shape then pressed, the listener starts drawing the shape at a high frame rate while the user is dragging. When releasing, the object is added to the ShapeLibrary and can be selected later. Also, the RandShape object is nullified after adding it to the library. There are a few if statements to correct the coordinates when dragging in the 4 quadrants. This might cause a compatibility risk when adding other shapes which aren’t handled in this file, but nothing to worry about now.

Finally, there is a start() method which invokes the run() method for TheCanvas object.

1. SelectShape:

Just one simple method here, described in the previous class. shapeSelect(x coordinate, y coordinate) loops the ArrayList backwards to check if an object was clicked. It returns the index of the object inside the array, and returns -1 otherwise.

1. PaintView implements ChangeListener:

Here comes the fun part. The GUI file. Almost every button and dialog appears thanks to this file.

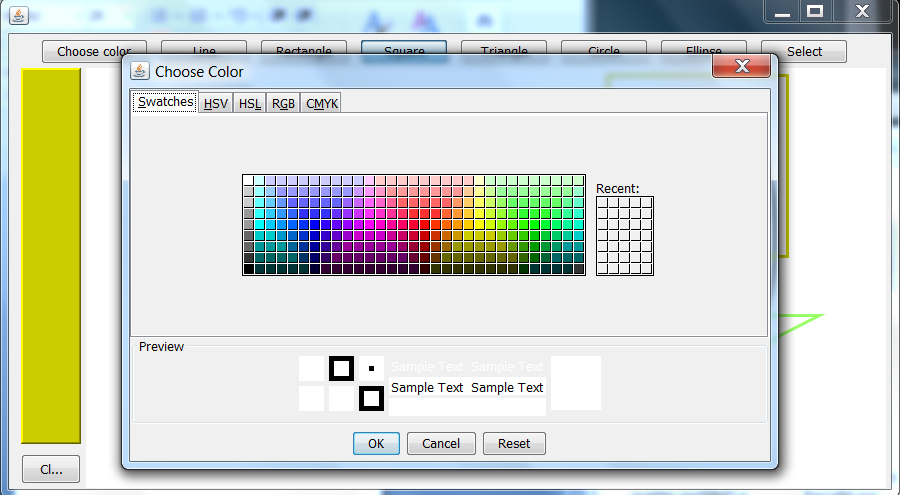
It contains a private class inside it theHandler implements ActionListener, which handles ActionEvents for most buttons. It uses a button that keeps clicked buttons selected (Sorry, what?) to make it easy for the user to know what he’s about to draw. It also changes the value of the integer that determines which shape is about to be drawn. If the clear button is clicked, the method reset() is invoked and the ShapeLibrary object is reset and of course, the canvas is emptied.

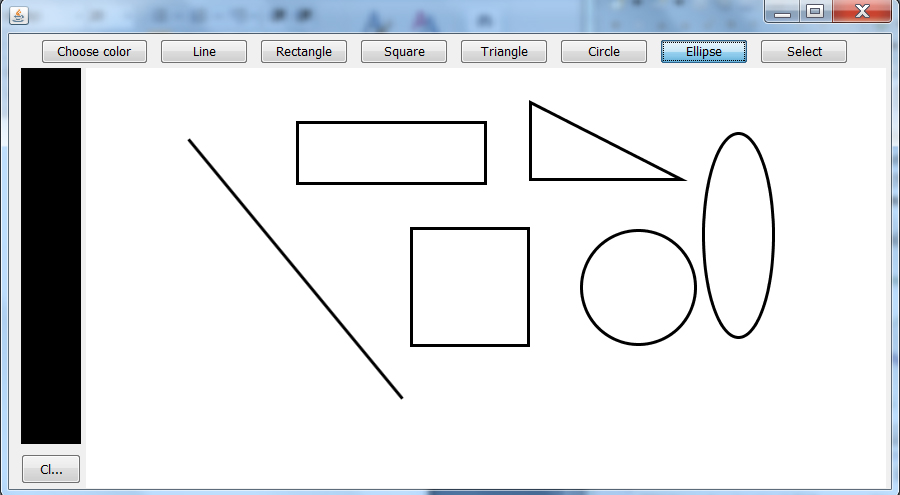
Back to Paintview, it contains a declaration for an object of type GFX2 and another of type JPanel. We create the necessary buttons here (Shape buttons, selecting, choosing color and clear).

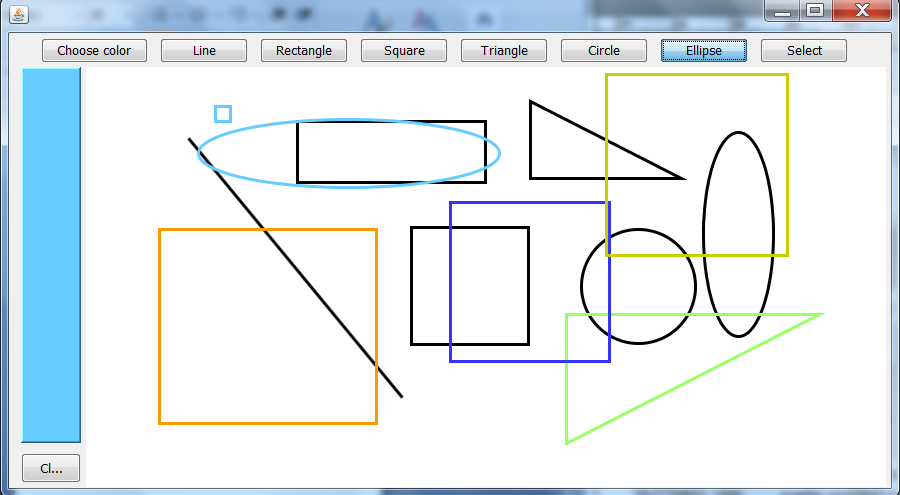
Then comes the constructor for PaintView which instantiates the JPanel object and initializes it. Another panel is created, where we can see the current color selected to be used when drawing. An object of type theHandler and all buttons are instantiated as well. Then, the GFX2 object is instantiated. There is the ActionListener for the color select button, which shows a built-in dialog showing JColorChooser. That’s all for this class.

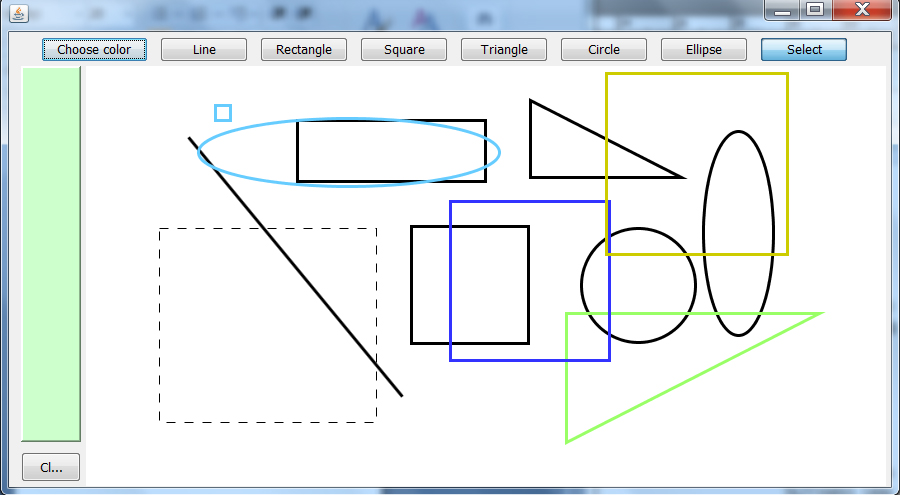
1. GFX:

Last class, declares a JFrame, and has the long-awaited main method. setLookAndFeel improves the way buttons look, and a PaintView object is created. The JFrame is instantiated and made visible, and the start() methodis invoked upon the PaintView object. Case closed.

1. Snapshots:







1. User guide manual:

Hello and welcome to this easy-to-follow guide. The program is too simple that it doesn’t even require a guide, but marks are marks. Anyway, right after running the program, you can see a window with a white canvas, a few buttons and a black panel.

You can only draw inside the white canvas.

The panel displays the color you’re about to use when drawing the shapes, which is black by default. You can change it by clicking the “Choose color” button. Then, a dialog appears. Pick the color from either the swatches tab or whatever tab you’d like to use. After choosing the color, click OK to confirm and change the color you’ll draw with. Click cancel to abort and use the old color. Click reset to set the color to white. You can select recent colors from the small palette near the color palette.

The shape drawing buttons allow you to draw various shapes with the color currently appearing in the left panel. For example, click the Line button (clicked by default). Now, you can press the left mouse button to select the first point of your line. Drag the mouse while pressing the LMB to decide the direction of the line. Release the mouse LMB to decide the ending point of the line. Pretty basic, huh? The same concept applies for the Rectangle, Square, Triangle, Circle and Ellipse buttons. Notice that the Triangle button draws only right angled triangles,

The select button works a bit differently. Click it and you’ll be in select mode. Pressing the LMB and dragging will have no effect at all. You need to click inside one of the shapes you drew before in order to select it. Notice that shapes overlay each other even if they are not filled. If you draw a small circle, and above it you draw a bigger circle that encloses the small one fully, you’ll be unable to select the small circle anymore. The select shape appears as dashed lines.

Lastly, the clear button clears all objects drawn on screen. Don’t click it unless you’re sure you don’t need the drawing anymore.

That concludes the user guide. Wish you good luck drawing something that makes sense with this program. HAPPY DRAWING!