Assignment III: Android Paint App

(Due on April 5th, 2013)

In this assignment you will complete the Paint app we started in class by implementing tools to draw the shape primitives. Touch events trigger a method calls on a View object. We will use these methods to add shapes to a picture. The analogy will be that of a conventional toolbox: a collection of tools for specific tasks, which we can use one at a time. In our case, each tool will help us to generate a specific shape primitive.

1 Preliminaries

1.1 Creating a ToolBox

A ToolBox is an non-visual component of the Paint app. In the DrawingView constructor, create a ToolBox object to hold the paint settings and the current tool. Implement a class ToolBox with the following:

1. All tools will use this set of attributes when adding a Shape object to the picture:

   ```
   private int strokeWidth;
   private int strokeColor;
   private int fillColor;
   ```

   Include getters and setters for these fields.

2. Add a reference to the view with a getter:

   ```
   private DrawingView drawingView;
   ```

   It is set on toolbox initialization and never changed thereafter.

3. A "preview" Paint object,

   ```
   private Paint previewPaint;
   ```

   for shape previews (see below). This sequence of method calls creates a dotted-line paint object:

   ```
   previewPaint = new Paint();
   previewPaint.setStyle(Paint.Style.STROKE);
   previewPaint.setColor(Color.GRAY);
   previewPaint.setStrokeWidth(1);
   previewPaint.setStrokeCap(Paint.Cap.ROUND);
   previewPaint.setPathEffect(new DashPathEffect(new float[]{4.0f, 4.0f}, 1.0f));
   ```

It is set on toolbox initialization and never changed, and can be accessed with the `getPaintPreview()` getter.

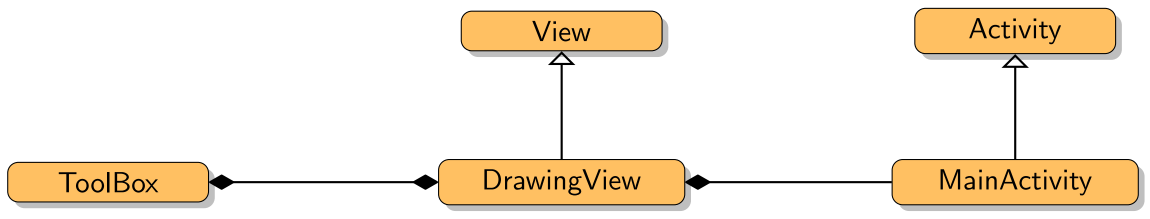
4. A *current* Tool

   ```
   private Tool currentTool;
   ```

   that can be accessed with the `getCurrentTool()` getter. At runtime, the reference `currentTool` will refer to one of the subclasses of Tool and therefore exhibit *polymorphic* behaviour. The current tool will be set using a method `changeTool(ToolName name)`, which you should implement to dispatch (i.e.: `switch`) on the provided name and set `currentTool` to the desired tool: RectangleTool, LineTool, etc. . .

1.2 Toolbox's relationship with DrawingView and MainActivity.

Create a ToolBox in the constructor of the DrawingView. The MainActivity class will require a toolbox reference to pass to the settings dialog. Add a ToolBox getter method in the DrawingView.



1.3 Methods to override in DrawingView

The DrawingView class extends the View class, of whose methods two need to be overridden:

1. `onDraw(Canvas canvas)`. All shapes should be drawn, followed by any tool preview (if there is one).

2. `onTouchEvent(MotionEvent event)`. This method is called when a touch event occurs. The `event` parameter contains a method `getActionMasked()`, whose value we can use to call the correct touch event handler in the current Tool. When this value is either `ACTION_DOWN` or `ACTION_POINTER_DOWN`, the current tool's `touchStart(event)` method should be called. When this value is `ACTION_UP` or `ACTION_POINTER_UP`, the current tool's `touchEnd(event)` method should be called. Otherwise, the current tool's `touchMove(event)` method should be called.

Remember: the DrawingView is redrawn when it's `invalidate()` method is called. Make sure you call this method appropriately.

1.4 App structure

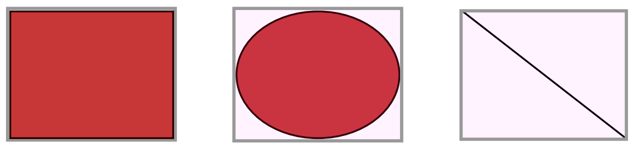
Your solution should have the following package structure to manage the multitude of classes introduced for shapes and tools.

`ca.qc.johnabbott.cs603.asg3`	MainActivity, DrawingView, Toolbox, etc...
`ca.qc.johnabbott.cs603.asg3.shapes`	Shape and subclasses
`ca.qc.johnabbott.cs603.asg3.tools`	ToolName, Tool, and subclasses

2 Tools for primitives Line, Rectangle and Ellipse

A ToolBox will contain Tool objects which can be used to add shapes to a picture. Specifically, at runtime, calling the `getCurrentTool()` method will return an alias (a reference) to the currently selected tool.

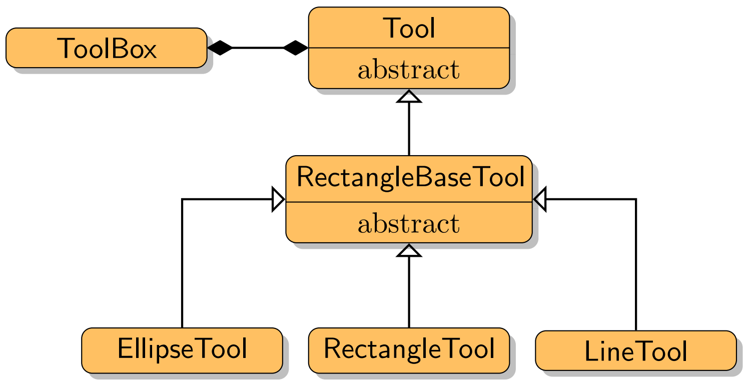
Rectangle based We can think of the shapes Rectangle, Ellipse and Line as being contained within a bounding rectangle:



For a user to create these shapes, they would need to do the following:

1. Touch somewhere on the View to indicate one corner of the bounding rectangle.
2. Move their finger to the desired size.
3. Release touch to indicate the other corner of the bounding rectangle. One of the above shapes will be added to the picture, touching the edges of the bounding rectangle.

With this in mind, we will implement the tools using the following class hierarchy:

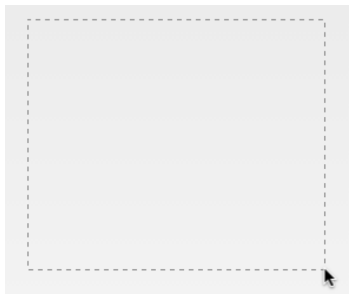
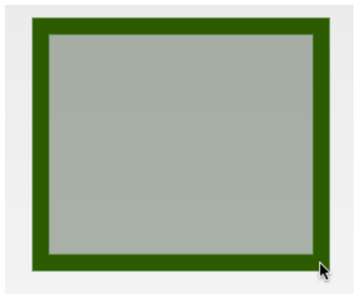
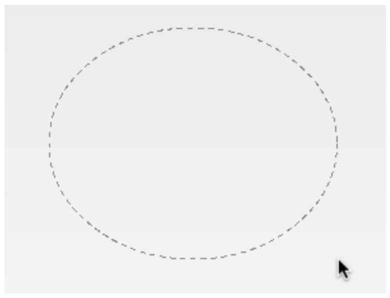
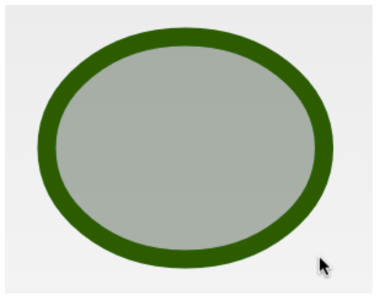
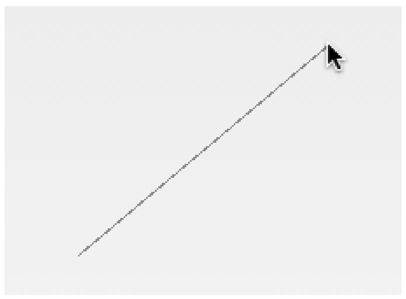
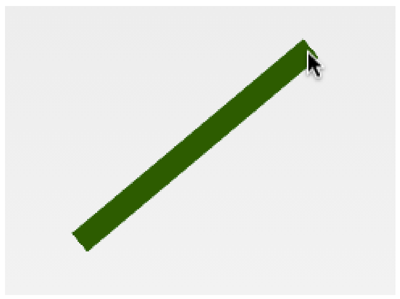


The abstract Tool has been provided.

In RectangleBaseTool, override the "touch" methods of the Tool class to create the bounding rectangle. In the subclasses of RectangleBaseTool, override the remaining methods for each shape.

2.1 Previews

To help the user create a shape, each tool should generate a *preview* on the DrawingView. Implement the `drawPreview(Canvas)` method of each tool class. The preview uses the `previewPaint` created in the ToolBox for drawing. Be sure to set the `preview` field of the Tool class accordingly so your DrawingView knows there is a preview to draw...

Preview	Resulting Shape
	
	
	

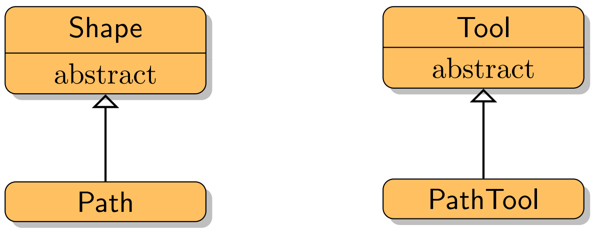
3 Paths

Add a new shape "Path" to your app. A path is a series of (x, y)-coordinates that are strung together with line segments. The Android class Canvas is capable of drawing paths (see documentation for Canvas's `drawPath` method and the class android.graphics.Path). They support both filled and unfilled path objects.

Create a class Path and associated PathTool. The latter will allow a user to create paths in the following manner:

1. Touch somewhere on the View to start the path.
2. Move along the desired path. The `touchMove` event will be sent to the PathView to record the points at regular intervals.
3. Release touch to indicate the other end of the path.

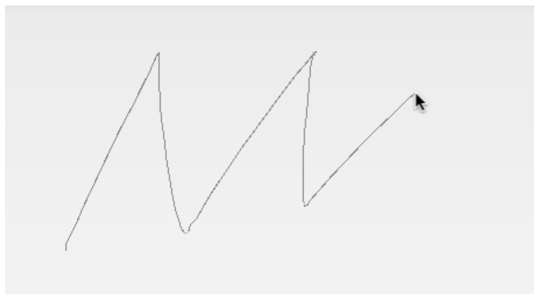
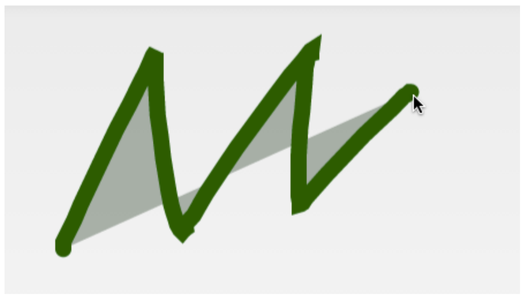
These classes should extend the abstract classes Shape and Tool respectively.



Remember to add a Path demo object to the ToolSettingsDialog and the PathTool to the toolbox.

3.1 Path previews

Add a preview to your path tool. It should use the preview Paint to draw the path constructed so far.

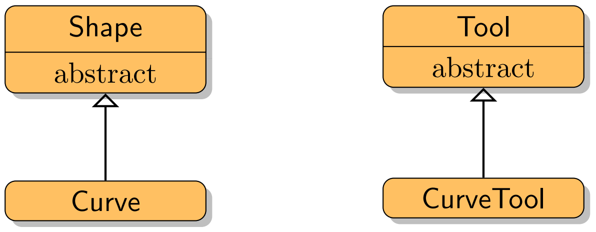
Preview	**Resulting Shape**
	

4 Quadratic Bezier Curves

Add a new shape "Curve" to your app. A bezier curve is a constructed mathematically using three points: the endpoints (x_1, y_1) and (x_2, y_2), as well as a control point (x_c, y_c) used to "bend" the line. Use the android.graphics.Path to construct a curve with it's `quadTo()` method.

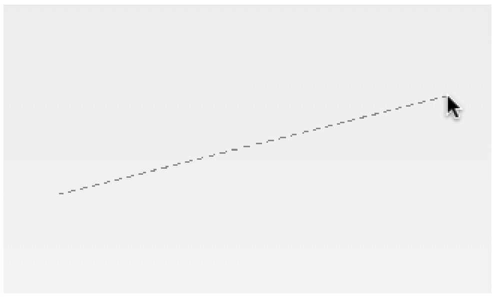
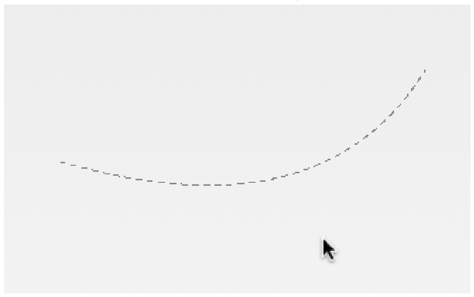
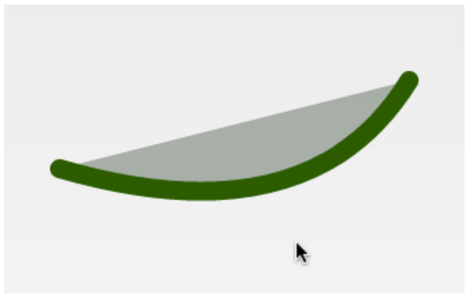
Create a class Curve and associated CurveTool. The latter will allow a user to create paths in the following manner:

1. In the first phase: create a line in the same way as you created a line above. Touch-press defines one endpoint and touch-release defines the other.
2. In the second phase: the user touches the screen a second time and when they lift their finger, this point is used as the control point.



4.1 Curve preview

Add a preview to your curve tool. It should use the preview Paint to draw the curve. For phase 1, this will be a line (you can still use the android.graphics.Path for this). For phase 2, until the user lifts their finger, their finger position should be considered the control point.

Preview	Resulting Shape
 followed by 	

5 Requirements

1. Build on the starter Android project provided.
2. (Optional) Implement the missing previews in ToolSettingsDialog.

3. Implement tools for all 5 shapes.
4. Implement previews for all 5 shapes.
5. Your ToolBox and DrawingView should use *polymorphism*. The ToolBox should store a subclass of Tool as the current tool and the DrawingView should keep an array containing subclasses of Shape.

6 Deliverables

- ☐ The minimum target framework should be Android 4.0, the maximum target framework should be Android 4.2.
- ☐ Code should be commented and variable names chosen appropriately. Insufficient commenting will be penalized.
- ☐ Submit the entire project directory (from Eclipse workspace) compressed, on Léa.