Patrick Aung

CSC 413

Software Development

Project #3 Write Up

The goals of this assignment is to create an android application that will draw a (randomly sized) rectangle or a circle to the screen. It will also have the shapes fade as more objects are outputted to the screen, which eventually would make them disappear completely. The application also has a button that will clear the entire display of all the objects on it and reset the shape counter to zero for both the rectangles and the circles. This write up is to explain the process that I endeavored in completing this assignment, as well as the resources that I used in order to achieve the goals that were required. The goal was to learn how to create java classes that would subclass each other. This also allows us to implement a factory design pattern in a hands on matter. The software design pattern allows a small amount of abstraction applied to our class, which allows each of the child classes to implement their own version of the method. In this class, the Rectangle and the Circle have their own version of onDraw(). The onDraw() method is the method that will handle most of the drawing to the screen deciding size and color of the object that needs to be drawn. This assignment allows us to add some abstraction to our classes and using a factory design pattern in Android Studio. The design pattern adds more readability to our classes and creates layers that keep us from having to edit code in each class to do a single thing (in this case transparency). The sections below will describe the steps I took in developing the application and what resources I used to complete the assignment.

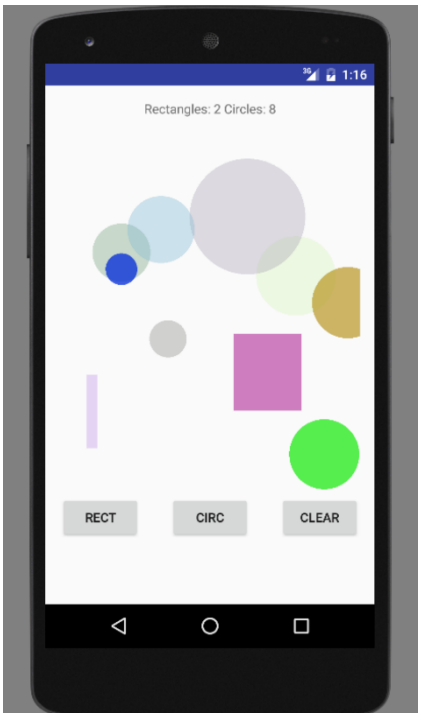

Figure 1: Completed UI of Fading Shapes App

I. The Layout

The first part I undertook to get the assignment started was by creating and formatting the different layouts I would need for the assignment. As stated by the instructions, at the very minimum I would need a text view that would display shape count, a relative layout for drawing shapes, and 3 buttons that would perform different functions (clear, rectangle, and circle). The layout of the assignment was relatively easy to set up since the last assignment had us learn a fair amount of the basics of Android. The ability to drag and drop items such as buttons and layouts to the design layout was probably my favorite feature of android studio. It allows me to size and define boundaries of each layout a lot easier than having to type it out.

II. Class Definition

The second thing I worked on for the assignment was to start creating all of the classes I would need for this assignment. The java classes that I would need for this application were ShapeFactory, Shape, Circle, and Rectangle. I decided to create all of the java classes first instead of working on each one. The Shape class extended the View class which was stated by the assignment's instructions. After adding all of the method to the shape class I noticed that the IDE required that I made the whole class abstract if it contained any abstract methods. Stackoverflow.com and the Android Studio developer page helped as a resource for creating the Shape class which extended View. I initially followed the Android Studio's format of passing both the Context and the Attributes as parameters for the initialization of Shape, but decided to remove the Attribute property because it was not going to be needed for the objects I that was drawing. Once I removed the attribute property, the process of creating of each Shape was cleaner and also did not require me to initialize every child class with the attribute parameter. Since the shape class had only had a few concrete methods (non-abstract), I decided to work on those before I got to the abstraction. The setShapeAlpha required me to do a little research on the View class as well as what methods were available to change the transparency. After searching the Android View class manual page, I was able to find the setAlpha() method and use that to change the transparency.

III. Shape Class

For the next portion of the Shape class, I did some research about the abstract key word. I found that when I declared the method abstract, it would require that I make the entire class abstract too. I was able to find a blog post from Javacoffeebreak.com, that showed examples of creating a child class when the parent was abstract. I also found that tutorialspoint.com had a post about Abstract Factory Design which also helped a lot in creating and formatting the ShapeFactory. I also found that the resource from tutorialspoint.com may provide more useful information on project 4 because it showed good examples of creating interfaces. I declared the method headers of the abstract methods getShapeType() and onDraw() in the shape class and decided that I needed to implement them in my Rectangle and Circle class. The only odd thing I ran into when building the class was finding out that I had to declare the whole class abstract if the class at least had 1 abstract method. The mistakes that I ran into with the abstract methods was defining a body for an abstract method and "class must either be declared abstract or

implement abstract method". I solved them by just declaring the whole class abstract and deleting the entire body of the abstract methods (left only the method header). I referred to stackoverflow.com for the errors and also oracle documentation on abstract classes. I was also initially going to use an enumeration for the ShapeType, but decided to just go with a String type; it made it a slight bit easier to read. In my resources, I have a link that I referred to for reference yet decided to not use it from a blog called Crunchify.com.

```
public class SimpleDrawingView extends View {
    // ...variables and setting up paint...
    // Let's draw three circles
    @Override
    protected void onDraw(Canvas canvas) {
      canvas.drawCircle(50, 50, 20, drawPaint);
      drawPaint.setColor(Color.GREEN);
      canvas.drawCircle(50, 150, 20, drawPaint);
      drawPaint.setColor(Color.BLUE);
      canvas.drawCircle(50, 250, 20, drawPaint);
    }
}
```

*Figure 2: Code Snippet from CodePath**

IV. Rectangle & Circle Class

After I completed the Shapes class, I moved on to implementing both the Circle and Rectangle classes. I knew that both classes were going to be similar but call different onDraw methods to draw onto the canvas. First, I had to do some research on how to draw to the canvas and what parameters I needed to complete the task. I did so by watching a short video by a Youtuber called mybringback, who goes step by step on drawing rectangles to the canvas. I also got supporting information from an Android Coding blog called Code Path, which showed me how to draw circles to the canvas (Figure 2). I also went to the Android developer pages to search up what parameters were required to use for drawCircle() and drawRectangle(). Both shapes required that the placement, size and color were all random. I made all of these random by implementing a random object that would return values bounded by a certain max depending on the category that I was making random. The size was depending on canvas.getWidth() and canvas.getHeight(). I decided to make it depending on the canvas instead of constant integer values so that it would be the same regardless of the devices screen size. The x and y coordinates of the object was also randomly decided bounded by the limits of the dimensions of the canvas. I also had to setup a Paint() object that was one of the parameters of the objects to decide the color. I set each object to be fully visible and had the RGB values to be randomly decided with rand.getNextInt(). However, I made one small mistake at first while building the circle class by setting the visibility at 256 instead of 255. This caused the circles to not show up on the canvas, but I was easily able to find the error by comparing my Rectangle and Circle classes. After the shape classes were complete, I went and worked on the ShapeFactory class which mainly had a switch statement returning an initialized Shape() (a rectangle or circle).

V. MainActivity

Finally, after finishing all of the classes, I was able to move on to the MainActivity that would handle the usage of all of the classes and put the complete application together. As suggested by the directions of the assignment, I decided to create a vector of type Shape that would handle all of the current elements on the Canvas. I had to refresh myself on how to handle vectors by looking up the documentation of the Vector class on the Oracle/Java manual pages. I initialized also some int values that would keep track of the count of each shape. I started to format the buttons and the method calls they would need to create and draw the shape. I decided to allow updateShapeCount() to handle all of the counting since it would reset all of the counts to zero and walk through the vector to recount the objects in case one was removed due to its visibility being zero. The updateShapeCount() also handled the output to the total rectangle and circle count displayed at the top of the screen. The last method I handled was the setShapeAlpha() since it was mainly dependent on its algorithm. Since it was a float, I decided that it would cycle through the vector and calculate its alpha depending on the total shape count. I created a simple if else statement that would decide if it was put back into the array (if alpha >0) or it would be removed and the shape count was updated.

VI. Final Thoughts

Out of my final thoughts on this assignment, one was that this assignment really forced me to look into a lot more different online resources and examples of the methods that I wanted to use from the Android Studio manual pages. Looking at the class header and seeing what class the shape class was extending also helped in finding what classes I would need to look up. Android's developer page showing how to make a custom view class marked my initial start point for what I would need to implement in both Circle and Rectangle. This assignment required that I look and read through extensive documentation that would allow me to put together the goals that were required for the assignment. Since the bulk of the assignment was code that we had to write, I found that keeping track of what needs to be done very important. I decided to keep a document by keeping track of notes on what needed to be done and the possible ways that the goals for the assignment could be achieved. Even though the assignment was simple in design, the assignment would not have come together without any organization and the right resources.

Project Resources

Google Developer Page (Canvas)

http://developer.android.com/reference/android/graphics/Canvas.html

Oracle Java Page (Vector)

https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html

Google Developer Page (Create a View Class)

http://developer.android.com/training/custom-views/create-view.html#subclassview

Crunchify Blog, Why use Enumerations (format on how to use) for ShapeType

http://crunchify.com/why-and-for-what-should-i-use-enum-java-enum-examples/

Youtube Canvas Tutorial (Drawing Rectangles)

https://youtu.be/IUGRQqfHb8k

Android Canvas Page

http://developer.android.com/reference/android/graphics/Canvas.html

Shape Factory Tutorials Point

http://www.tutorialspoint.com/design_pattern/factory_pattern.htm

Code Path Tutorials (Extending View)

https://guides.codepath.com/android/Basic-Painting-with-Views

Blog for Drawing Circle to Canvas

http://android-coding.blogspot.com/2012/04/draw-circle-on-canvas-canvasdrawcirclet.html

Abstract Factory Pattern (Tutorials Point)

http://www.tutorialspoint.com/design_pattern/abstract_factory_pattern.htm

Abstract Classes (for methods and extending)

http://www.javacoffeebreak.com/faq/faq0084.html

Android set Alpha (for changing alpha of shape)

http://developer.android.com/reference/android/view/View.html#setAlpha(float)

Oracle Abstract Classes (java)

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html