**Project 3 Fading shapes App**

**Project Description**

In this project we use factory software design pattern to create shapes and display shapes on the view using buttons and Views in android. The shapes disappear on the view when new shapes are added. We create two shapes circle and rectangle, and two buttons are created to trigger these shapes. When we press the clear button it clears the contents on the screen. The app also displays the number of rectangles and circles on the top of the layout in text view.

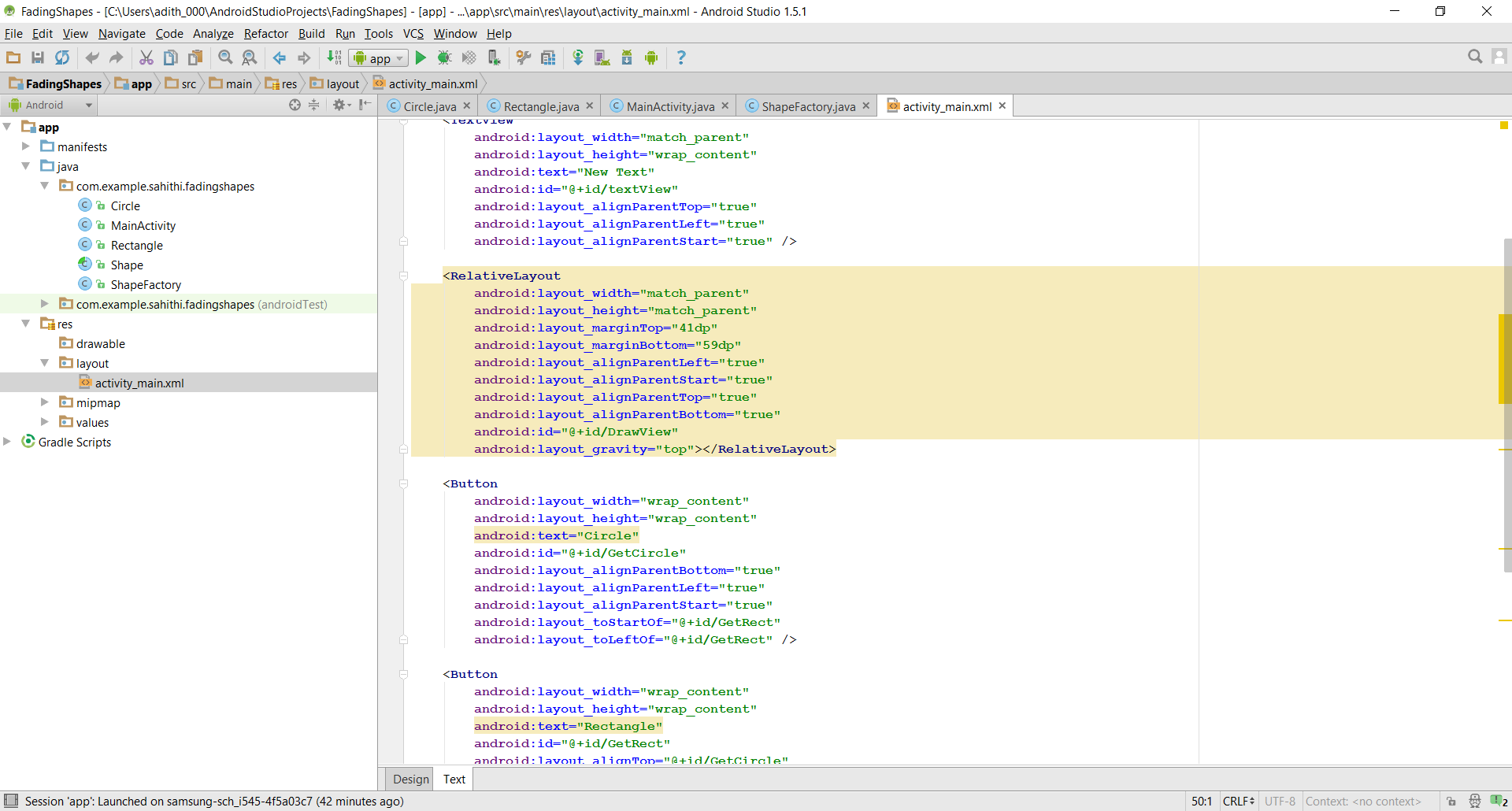
**Goals of the Project**

1. Use Relative layout for viewing the shapes and create buttons to trigger appropriate actions.
2. Use text view to display the count of appropriate shapes.
3. Use factory design pattern to derive classes and needed functions to trigger functionality of the shapes.
4. Get familiarized with androids 2d graphics and onDraw methods.
5. Knit all the moving parts in the Main activity for making the application working.

**Steps to achieve the goals of the project**

1. **Layout**

Our basic User interface is simple. On the top of the screen we use text view to display the count of the shapes. In the layout we create another relative layout to do our drawing of the shapes. On the bottom of the layout we create three buttons for two shapes and clearing the view. The code for our basic layout in xml is displayed in the figure 1 below.



**Figure 1- A snippet of our MainActivity.xml layout**

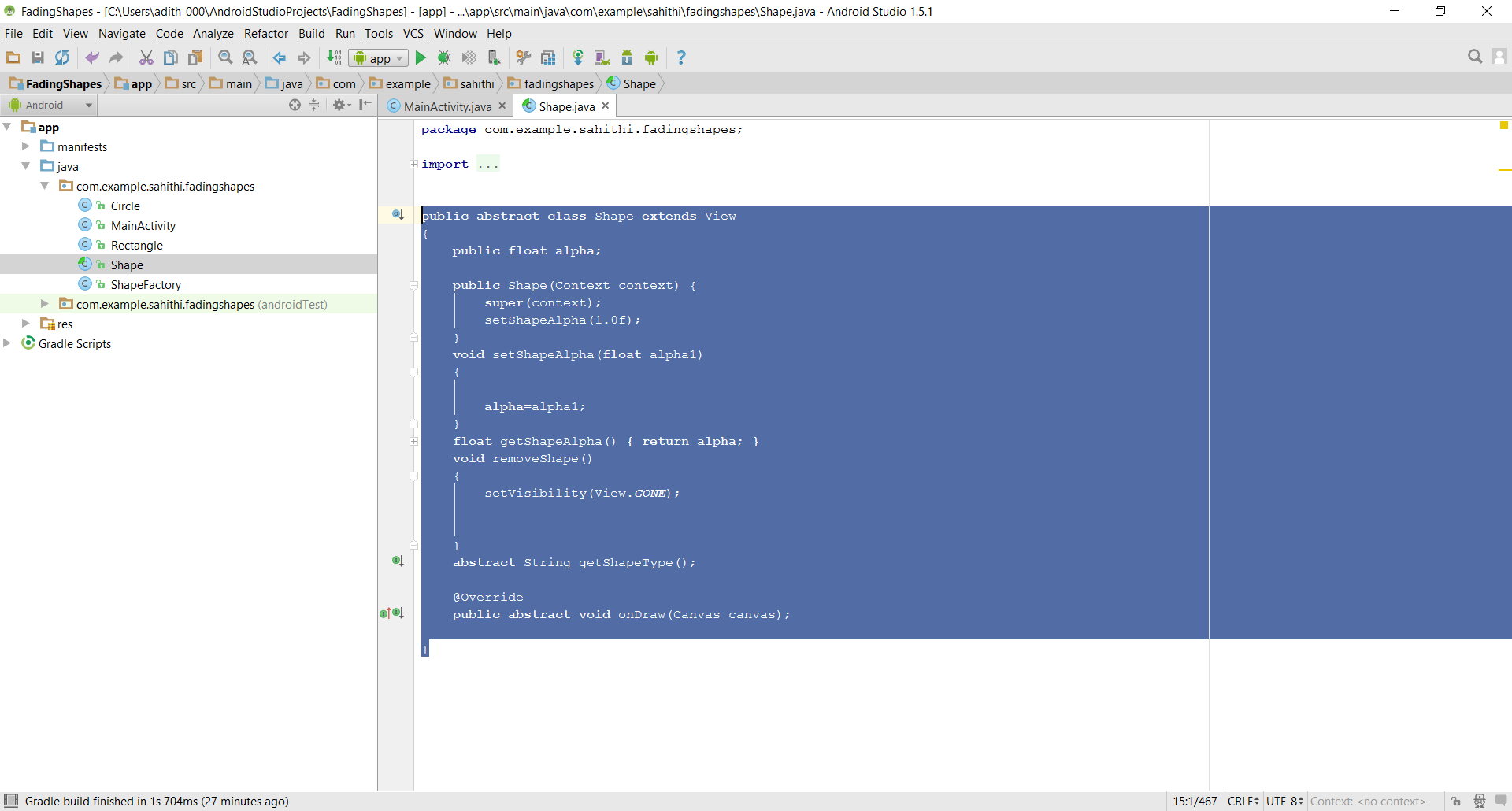
1. **Factory design pattern for shapes**

In this project we use factory design pattern to draw our shapes. Factory patterns are the most used design patterns in java. Factory pattern comes under creational patterns because it helps us to create objects without exposing the creating logic to the client. In this app I created a shape view and corresponding classes such as Rectangle and circle implementing the shape views. A factory class Shape Factory is defined in the last step.

**Shape.Java**

Shape extends view and is the parent class from which our shapes circle and rectangle are derived. Shape class has the following classes inside it. The code for Shape.java can be referred in figure 2.

* **public shape(Context context)-** This is the constructor for our specific class with super calling the context.
* **void setShapeAlpha(float alpha) –** This function is useful in setting the alpha of our current shape. Alpha is defined as the opacity of the object and is a float value between 0.0f-1.0f with 0.0 being transparent and 1.0 being opaque.
* **float getShapeAlpha()-** This function returns the alpha of our current shape.
* **abstract string getShapeType()-** This returns and indication of what type of shape this is.
* **public abstract void onDraw(Canvas canvas) –** this function is implemented by all the child classes and helps the child classes for drawing circles and rectangles.

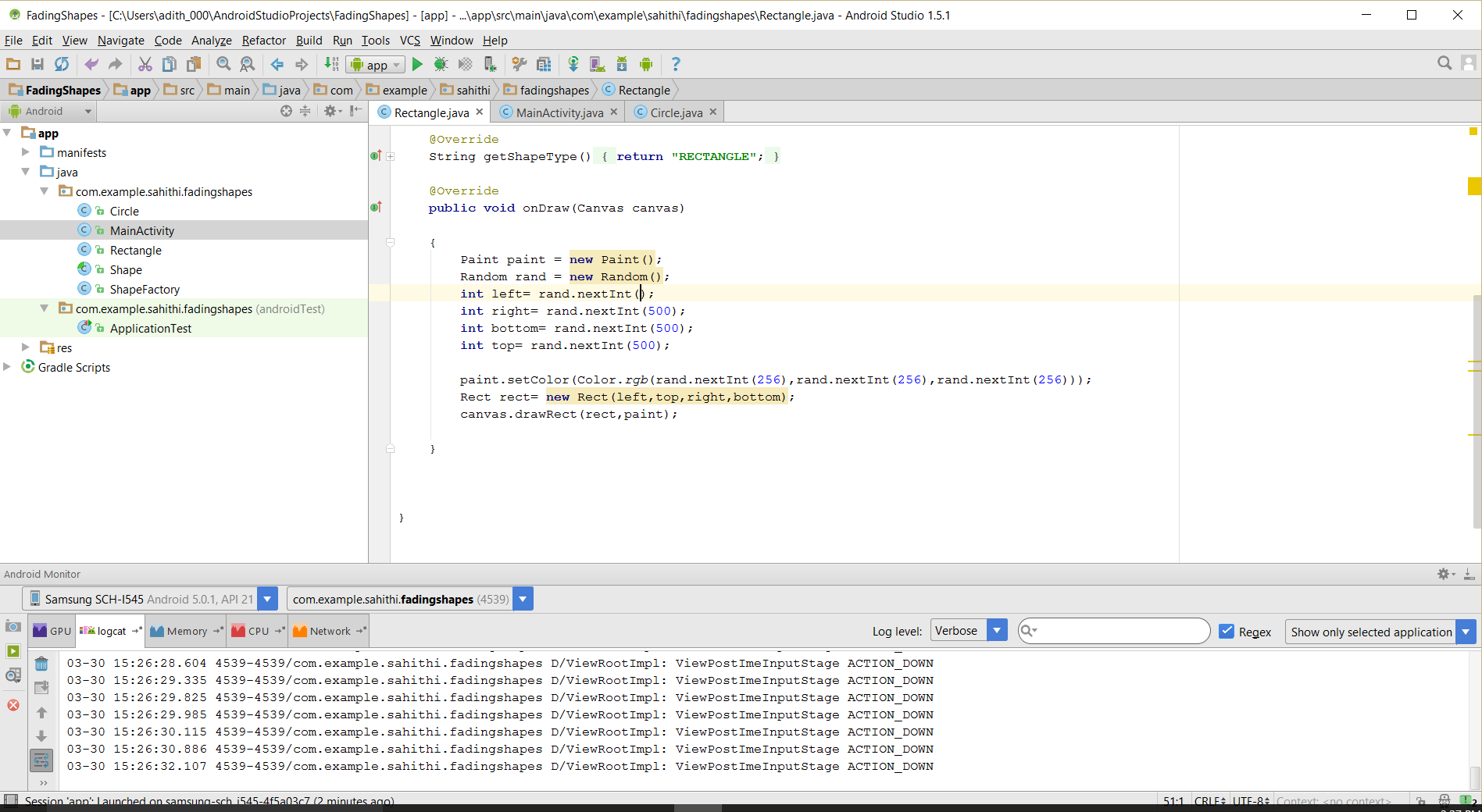


**Figure 2- A snippet of our Shape parent class.**

**Rectangle.Java**

Rectangle.java has three classes. It extends shape and has onDraw class it draws a rectangle on to the canvas. It also implements the getShapeType method. Code for rectangle.java can be found in figure 3.

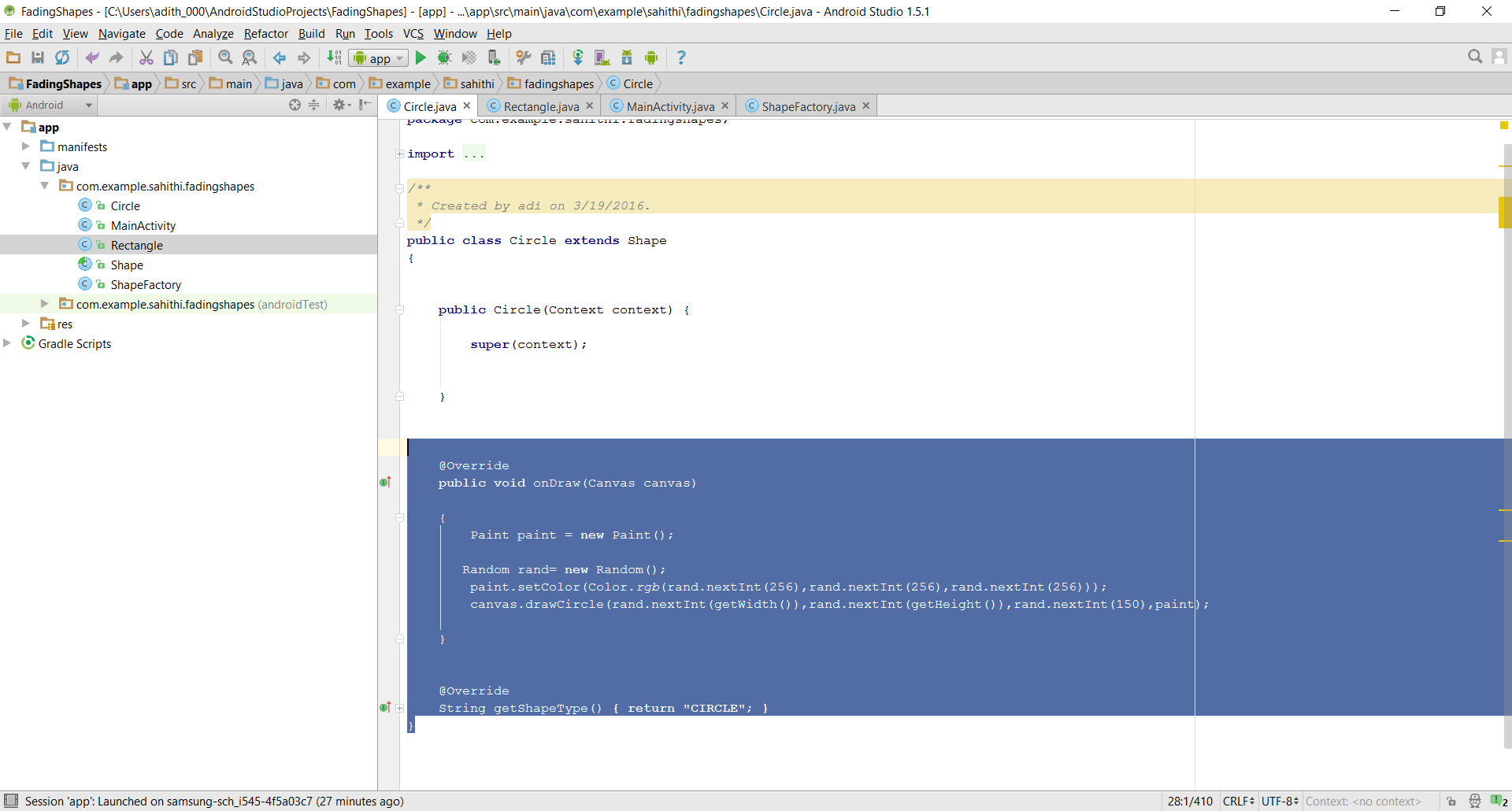
**public void onDraw(Canvas canvas) –** the most important step in drawing a custom view is to override the onDraw() method. The parameter to onDraw() is a canvas object that the view can use to draw itself. The canvas class defines methods for drawing text, lines, bitmaps, and many other graphics primitives. We need to use paint function to color our objects. The good thing about drawing class is it already has custom shapes defined. We just need to import them and draw on our view. For our rectangle class we use canvas.drawRect() method to draw our rectangle on to our view. Our rectangle class draws random rectangles and with random colors.



**Figure 3- A snippet of our onDraw class for rectangle.java**

**Circle.Java**

Circle also extends our shape class. In onDraw, the class draws a circle on to the view. It also implements the getShapeType () method. Code for circle.java can be found in fig 4.

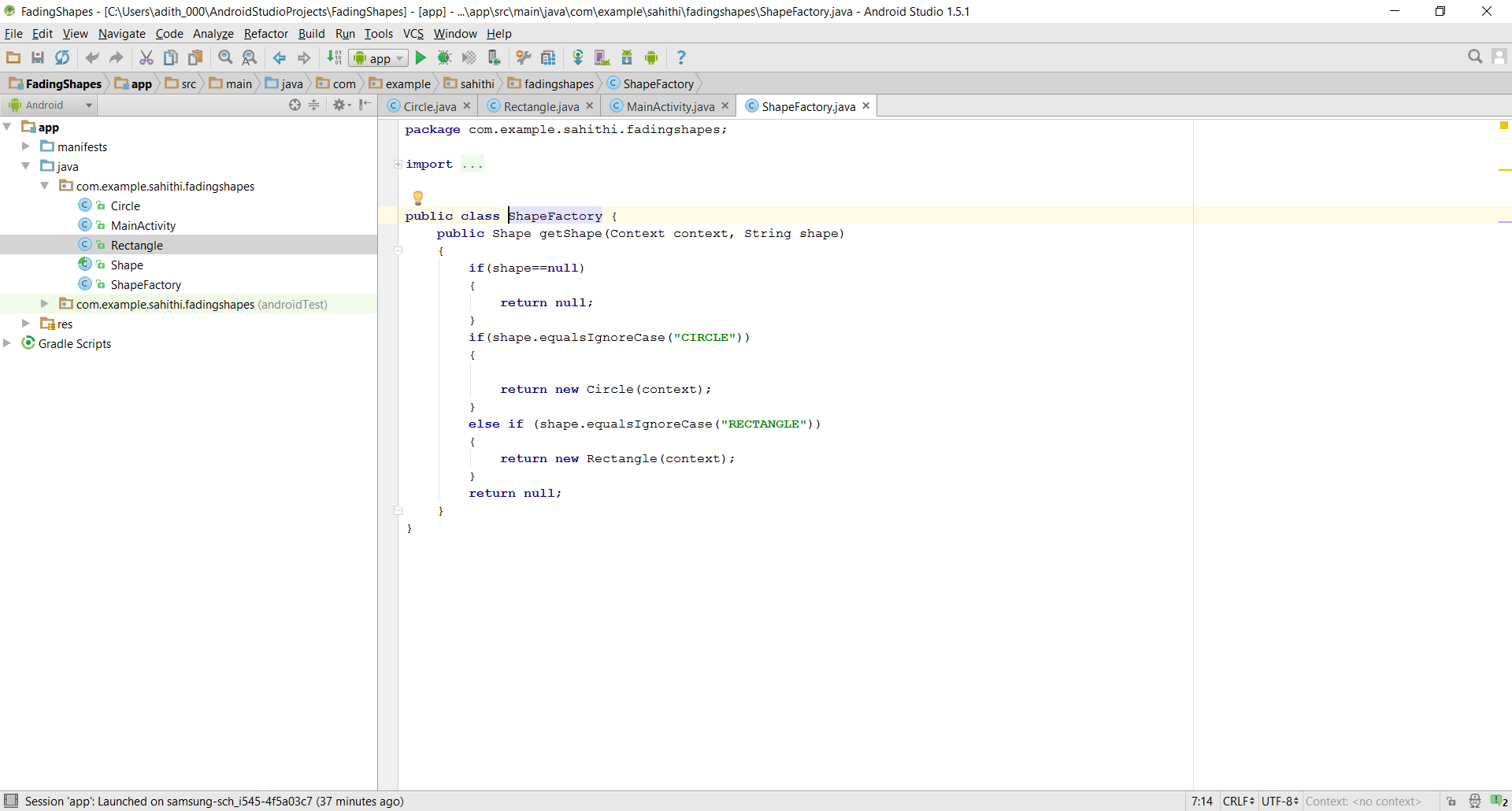


**Figure 4- A snipper of our Circle.java class**

**ShapeFactory.java**

ShapeFactory provides a standard implementation of our factory software design pattern. This creates a factory to generate object of concrete class based on given information. In this app it returns the shape objects of the requested type. It has one method. Code for ShapeFactory.java can be found in figure 5.

**public shape getShape(Context context, String shape)-** The context in this method is obtained by view.getcontext() in our onclick methods.

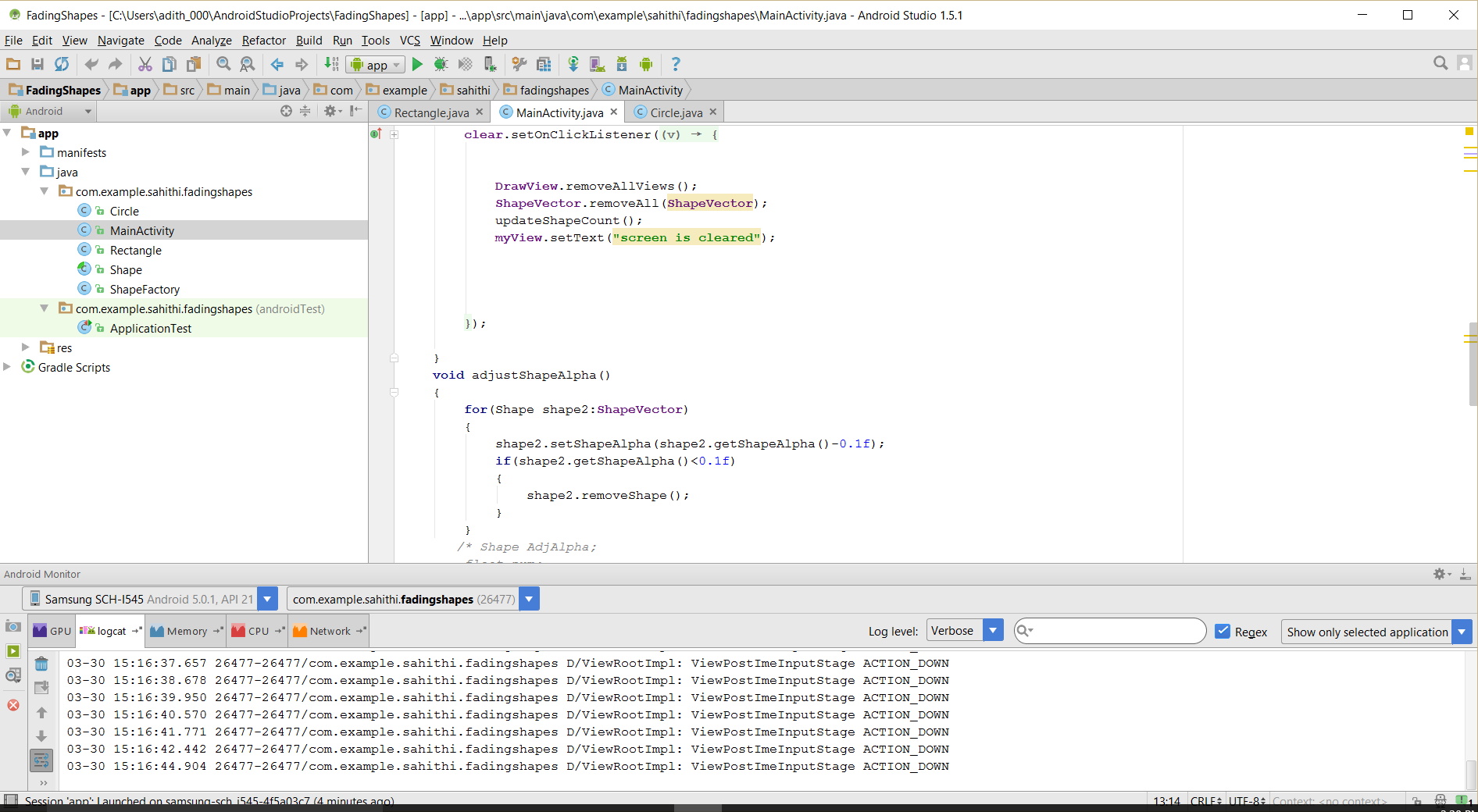


**Figure 5- A snippet of the ShapeFactory class.**

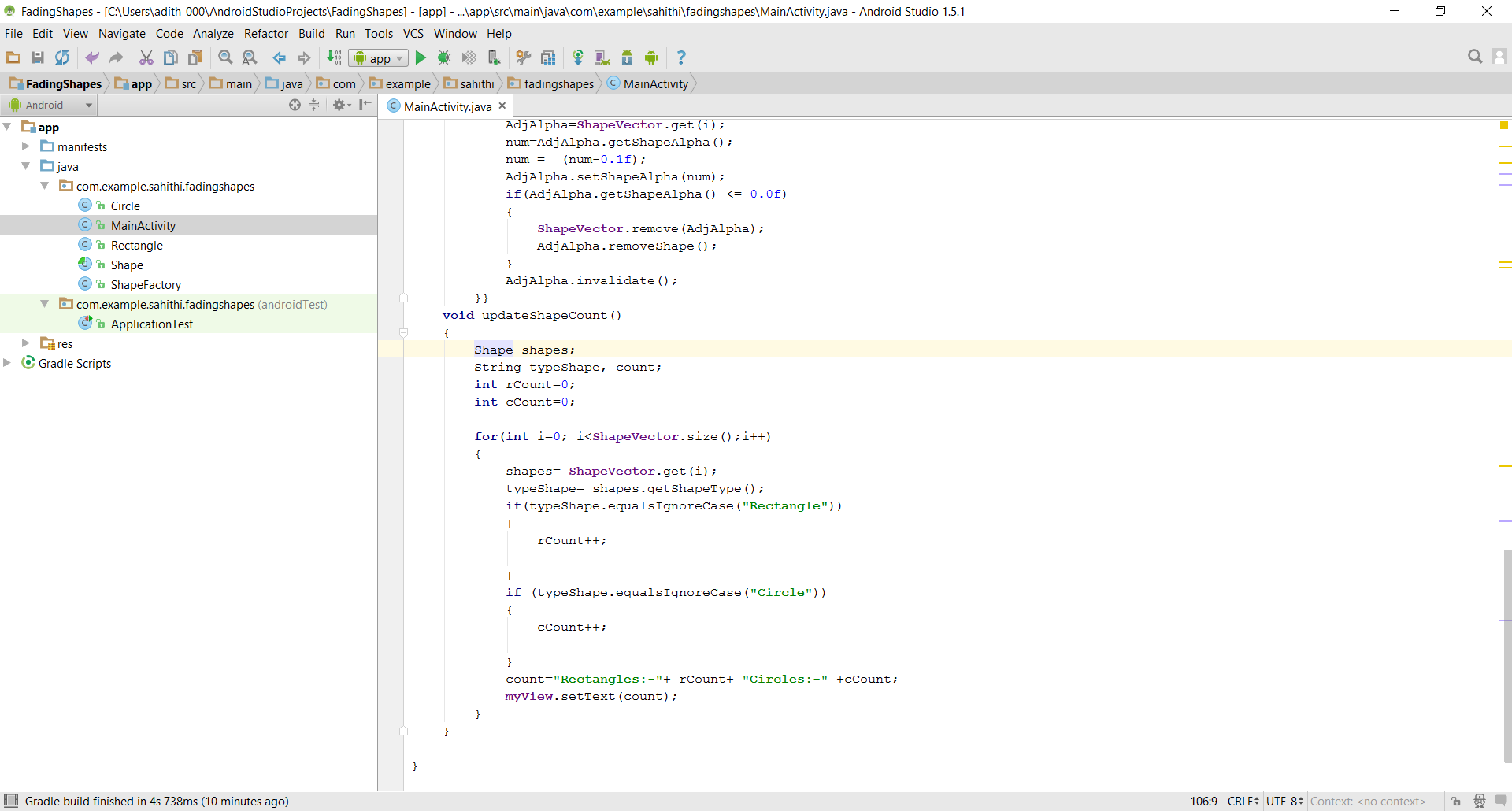
**MainActivity.Java**

Our main activity java file is where we knit everything together. We trigger our actions for our shapes using buttons. We use vectors to keep track of our shapes being draw on our view. If the shapes are not visible we remove the shapes from the vector to make sure we do not run out of memory. In addition to the other classes we have to create two extra classes to adjust the shape of the alpha and to count the number of rectangles and circles on our view. We use the below methods for our functionality

* **void adjustShapeAlpha()**- This reduces our alpha of the shapes that is currently in our shape vector by some amount. When the shapes are below alpha level that is transparent, the shape is removed from our view by using our removeShape() method in our shape class. Code for this class can be found in figure 6.
* **void updateShapeCount()**- This method is used for counting the number of rectangles and circles which are in our current shape vector, that is the count of our visible shapes. This method uses getShapeType method to count the number of shapes. The count is display on top of our text view. Code for this class can be referred in figure 7.

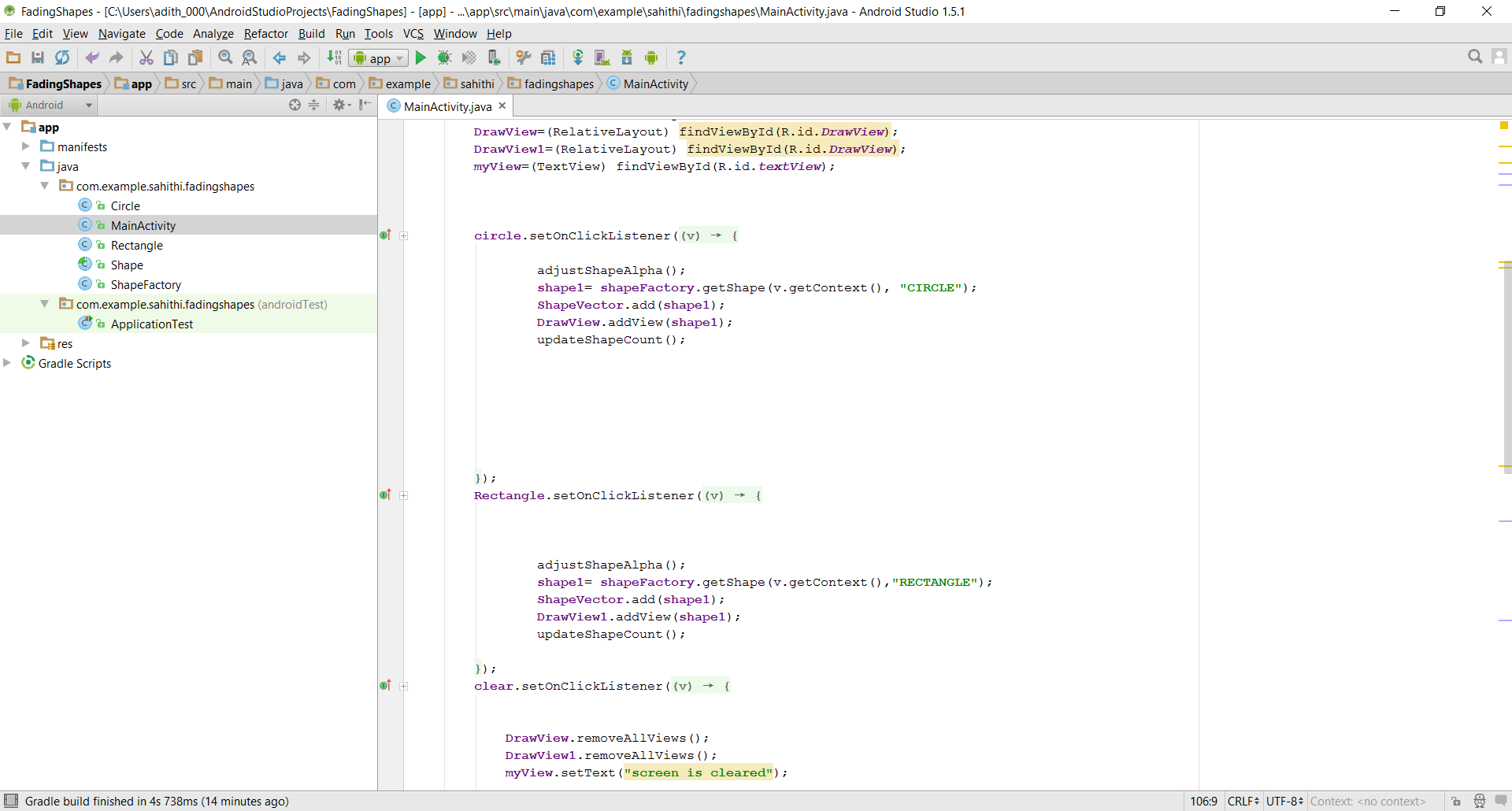


**Figure 6- A snippet of our adjustShapeAlpha() function**

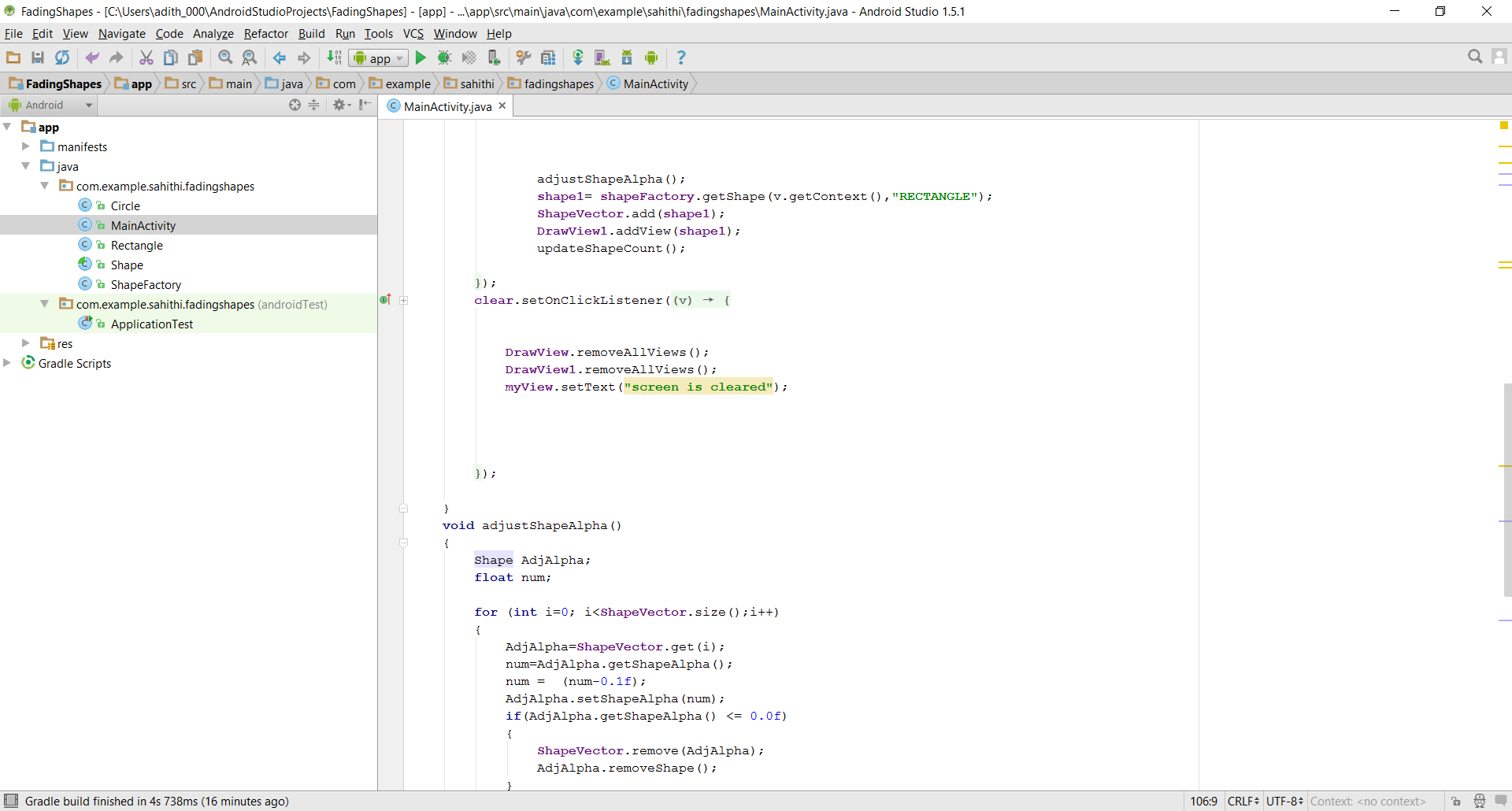


**Figure 7- a snippet of our updatShapeCount() function**

In our on click listeners we will provide the functionality and we draw the shapes on our view. When we click our clear button we clear the contents on our screen. Figure 8 explains how the functionality is implemented in our onClick listeners and figure 9 explains what is inside the clear button.



**Figure 8- A snippet of our onClick listeners**



**Figure 9 – A snippet of clear button**

**Conclusion**

By doing this project we get familiarized by factory design pattern and how they are implemented. Factory pattern is very simple and very easy to use after we get the idea about the design pattern. It provides a good structure to provide functionalities for our shapes without getting confused. With simple idea we can create many shapes on to our existing views. Conceptually even though is simple, I had to do a lot of research about how drawing functionality works. With thorough research and good examples used from the web I was able to achieve the goals of the project. By finishing the project I was successful in implementing and learning about factory patterns. With this simple app I was able to learn a lot of new information.

**Sources**

1. **http://www.tutorialspoint.com/design\_pattern/factory\_pattern.htm-** how to use factory pattern, referred this tutorial especially for ShapeFactory.
2. **http://courses.coreservlets.com/Course-Materials/pdf/android/Android-Drawing-1.pdf- used** for learning about onDraw and how 2d graphics are used in android.
3. **http://developer.android.com/reference/android/graphics/Rect.html#left-** referred for rectangle class.
4. **http://developer.android.com/reference/android/graphics/Canvas.html#drawOval(android.graphics.RectF, android.graphics.Paint)-** used for drawing objects.