Sahithi visvanathan

CSC-413

912549149

Project-4 Styled Fading shapes App

**Project-4 Styled Fading shapes App**

**Project Description**

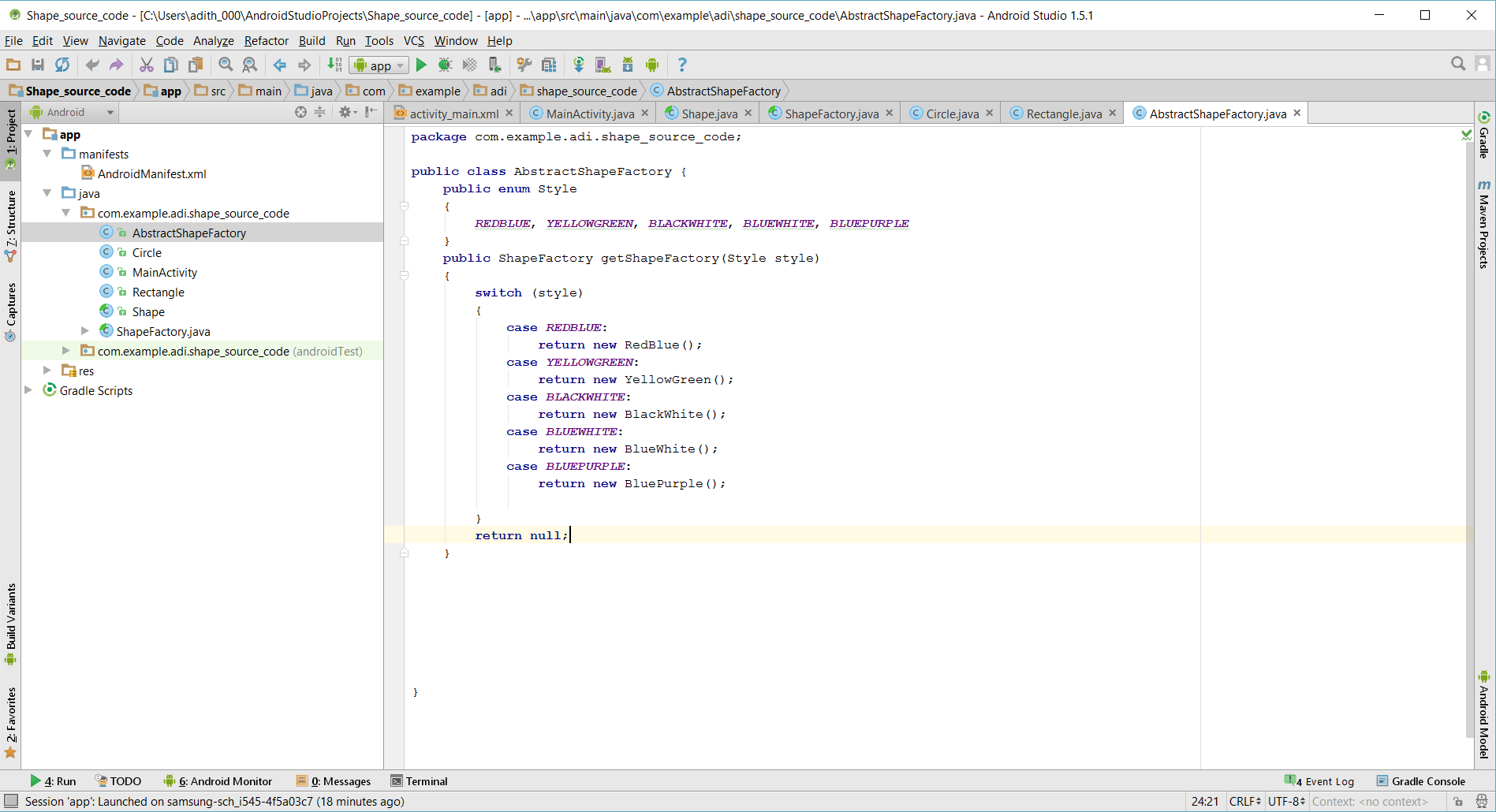
In this project we will implement abstract factory design pattern. We will extend our code in project 3 to accomplish our goals described in this project. In this project our buttons are triggered to draw shapes like circles and rectangles on our view like in project 3. As new shapes are added the previously drawn shapes are faded away on our view. In this project we will extend our code in project 3 which styles our circles and rectangles with a particular fill and border color. All shapes with a particular border and fill color are considered to have the same style. This project draws randomly sized circles and rectangles like in project 3 but now the shapes are drawn in our current style. The style button is added so that it is helpful in changing styles our shapes displayed on the view. Our style button is also used to loop back around to the start of the list of styles. Clear button is used to clear our view and also as in project 3 we display the count of rectangles and circles on the view, and also the name of the next style on our text view.

**Differences between Project 3 and Project 4**

Project 3 was based on factory design pattern and project 4 is an extension of project 3 but we introduce abstract factory design pattern to style our circle and rectangles. To understand the differences I first read about how abstract factory design works. Some of the differences between factory patterns and abstract factory patterns are:

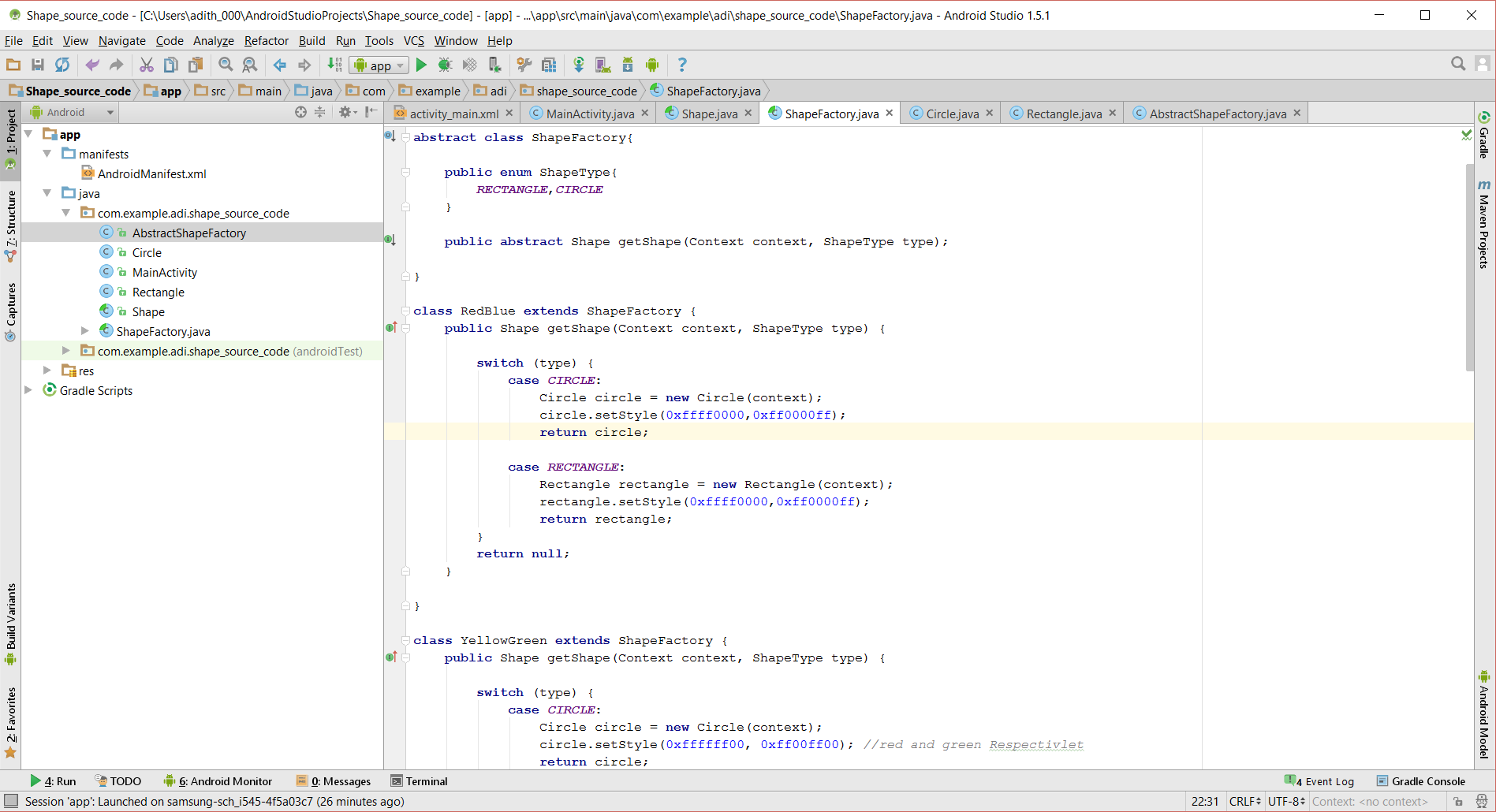
Abstract factory pattern are used to create families of related or dependent products of our factory class whereas Factory method is used to create one product only. Abstract Factory pattern expose a family of related objects which may consist of factory methods whereas factory method pattern exposes a method to the client for creating the object. Abstract factory pattern extends our factory methods to provide more functionality to our objects in our factory pattern. Abstract factory pattern also uses composition to delegate responsibility of creating object to another class while factory design pattern uses inheritances and relies on derived class or sub class to create object. Abstract factories are useful when we have to create multiple families of products without exposing the implementation details.

In our project we took use of the abstract factory design pattern to further extend our functionality to our shapes. In our previous project we had shapes like circle and rectangle derived from our shape class. We created only shapes. In project 4 we introduced to the concept of styling of our shapes which uses abstract factory pattern. Styling refers to the color of the border and fill of our shapes. In our project we create an abstract shape factory class called AbstractShapeFactory. As in our project 3, project 4 will produce circles and rectangles by every shape factory that is produced by the abstract shape factory. To achieve this I created an enum for style in AbstractShapeFactory. The enum has different border and fill colors labeled that are derived from our shape factory. We create various classes for our styles in our ShapeFactory. We can also use int for our style instead of enum. Our AbstractShapeFactory has an another class called **public ShapeFactory getShapeFactory (Style style)** which uses the enum style values, and with these values our classes related to various style will be called from our ShapeFactoryClass. The AbstractShapeFactory class can be viewed in figure 1 below.



**Figure1- A snippet from our AbstractShapeFactory class**

In project 3 we did not make our ShapeFactory class abstract. But in project 4 we have to make our ShapeFactory abstract because we are inheriting the styles from our AbstractShapeFactory method. ShapeFactory needs to be extended in project 4. This is where we add all our styling classes such as RedBlue, YellowGreen, BluePurple which provides style to our objects circle and rectangle. And all the classes relating to our styles needs to be extend from our ShapeFactory class. We can see how the ShapeFactory is defined in figure 2 below

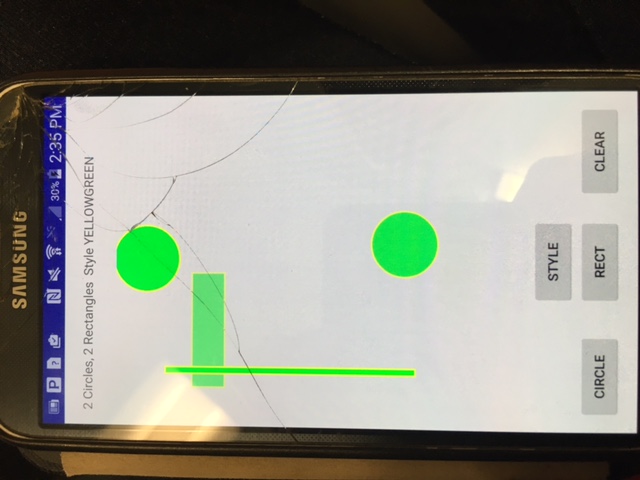


**Figure 2- A snippet of our ShapeFactory Class**

A few little things have to be changed in our Shape class, Rectangle, and class because we have to set paint styles such as stroke and fill for our circles and rectangles. A new class called void SetStyle is added in our shape class that sets the color of our border and fill.

In our main activity we just add a new button called style. A few things have to be changed in the main activity also like displaying the current shape style on our text view. Shape button changes the style, it loops back to the first style when all styles are done.

Figure 3 shows the image of our complete application running in my device. It shows the current style of our shapes that are drawn.

****

**Figure 3- A snapshot of the application**

**Sources Used**

1. [**http://www.tutorialspoint.com//design\_pattern/abstract\_factory\_pattern.htm**](http://www.tutorialspoint.com//design_pattern/abstract_factory_pattern.htm) **(talks about abstract factory pattern)**
2. [**http://developer.android.com/intl/zh-tw/reference/android/graphics/Paint.html**](http://developer.android.com/intl/zh-tw/reference/android/graphics/Paint.html) **(for paint.style)**