HW2 Basic Android Practice

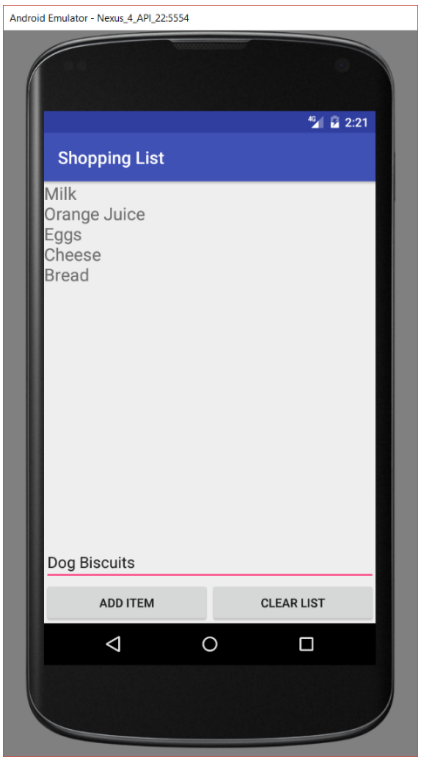
For our first assignment we're going to develop three simple Android programs. This will give you a chance to experiment with Android layout and basic wiring together of some Android Views. **This assignment is due at 11:59pm at the end of Wednesday January 24.**

Develop these as three independent projects named: "Shopping List", "Shipping Calculator", and "Color Picker". Your company domain is cs108.stanford.edu. Target SDK for all three is API15 Android 4.03. Each will involve a single activity which you may leave with the default "Main Activity" name.

Zip all three project folders into a single file zip file which you will submit on Canvas. Note that there are no starter files. You are developing everything from scratch.

Shopping List

For our first problem we create a simple Shopping List application. The application contains four Views, a TextView showing the current shopping list, an EditText allowing entry of new items for our shopping list, and "Add Item" and "Clear List" Buttons. Here's a screenshot showing our application in action:



As shown the "Add Item" and "Clear List" buttons should appear at the bottom of the application with each taking up half of the screen width. An EditText should be directly above the two buttons and should fill the screen width. The remainder of the screen should be filled with the TextView for the actual Shopping List. The default font size for TextView is actually rather small and thin, so set textSize on the

TextView to a more robust "20sp". Both the TextView and EditText should start out empty when the application starts up.

When the user clicks on “Add Item”, whatever is currently in the EditText should be added to the bottom of your Shopping List. In addition, the current text in the EditText should be cleared making it easy for the user to add another item. You may assume that the EditText always has text when “Add Item” is clicked. You also do not have to worry about what happens when the user has added so many items that they no longer fit in the TextView.

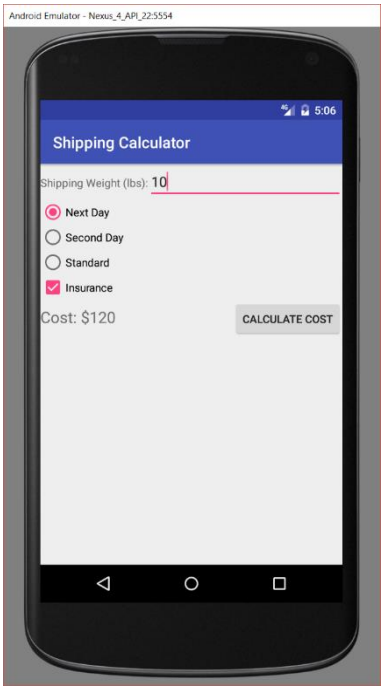
When the user clicks on “Clear List” you should clear the contents of the TextView (you can do this by setting the text to the empty string "").

Here’s a rough outline on how to build this application:

1) Edit the activity_main.xml file and layout your four Views. There’s a variety of different ways you can create the same layout. Either using a nested LinearLayout or a GridLayout should work well.
2) Setup your two buttons to call methods in MainActivity.java.
3) Write your button handlers.
4) Test your application.
5) Go shopping.

Shipping Calculator

In this problem we create a simple calculator telling a user how much it will cost to ship their package with our shipping company. Here’s what the application looks like when it’s up and running:



As you can see the application includes a TextEdit allowing the user to enter in the weight of their package in lbs. They then select a shipping speed. They can choose to insure their package. Finally they click on “Calculate Cost” and we display how much it will cost to ship with our company.

When the application loads, the Shipping Weight should initially be left empty. The “Next Day” option should be selected and the “Insurance” checkbox should be checked. The initial cost should be displayed as “Cost: TBD”.

Set the inputType on the weight EditText to "numberDecimal" so a numeric keypad with a decimal point shows up when they try to enter in a cost.

When the user clicks on the “Calculate Cost” button calculate the cost as follows:

- Next Day Air costs $10/lb.
- Second Day Air costs $5/lb.
- Standard Shipping costs $3/lb.

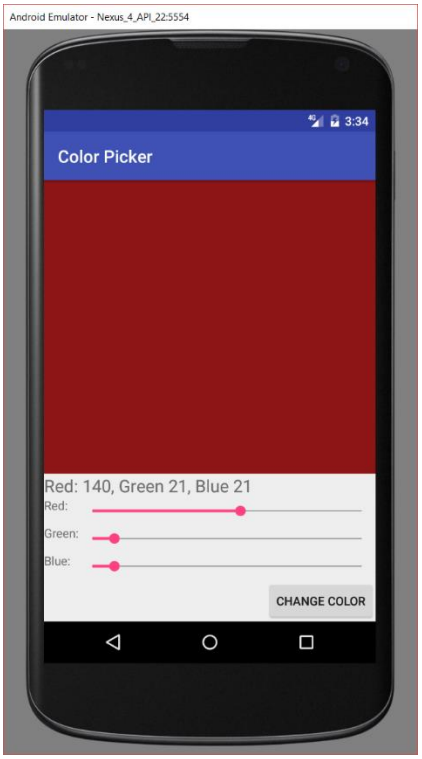
If the Insurance checkbox is checked, increase their shipping cost by 20%. You may assume that the user has entered in a legal decimal number in the EditText when the “Calculate Cost” button is clicked.

To keep output display simple (and to earn our company a bit more profit!) round the cost up to the nearest dollar.

If you’re planning to ship something any time soon and the costs displayed in this application alarm you, I pulled the costs out of thin air, so don’t take these costs as definitive!

Color Picker

For this problem we create a simple Color Picker. This will allow a user to select an amount of Red Green and Blue using SeekerBars and see what the corresponding color looks like. Here’s what this application looks like in action (showing the official values for Stanford Cardinal Red).



As you can see this application contains a TextView listing the current Red, Green, and Blue settings (in decimal); three SeekerBars, plus TextViews labelling the SeekerBars as Red, Green, and Blue; a button in the lower-right-hand corner to “Change Color”; and the remainder of the screenspace is taken up with a View which serves to show the current color.

The user will slide the Red, Green, and Blue sliders and then click on the “Change Color” button. Your application will change the color on the top View and will also change the text of the middle TextView to show the new Red, Green, and Blue values.

Please note that the colors and text will not change until the “Change Color” button is clicked. We’ll show you later in the quarter how to get everything to update immediately as the user slides the SeekerBars. For now though, sliding the SeekerBars will move the thumb dots, but will not update the color or the text.

Your SeekerBars should be setup to go from 0 to 255, and will initially start at 0 when the application starts up. Initially the Color Setting text field should display “Red: 0, Green: 0, Blue: 0” and the top square should be black (set the android:background to "#000000"). Also note that the labels associated with the SeekerBars are all left aligned and the SeekerBars themselves are all aligned together.

You may set the color of the View using the following formula:

```
colorView.setBackgroundColor(Color.rgb(red,green,blue));
```

where colorView is the View at the top of your application, and you’ve retrieved the red, green, and blue values from your SeekerBars. Note: you’ll need to explicitly import android.graphics.Color to get this to work.

Warning: if you use a GridLayout for the application and do not set an explicit height and width on the top view and instead rely on layout_gravity="fill", it will not show up unless all 5 rows below it are filled with Views. If any of the rows below are empty, they will end up taking the remaining vertical space available, instead of letting your top View take the vertical space. This technique will work, just don’t try doing it with just a single slider included for testing purposes – all rows must be filled.