Lab 02 Android Interface Layout

In CS301, you’ve been learning how to use a layout view (i.e., widgets or controls) to create a graphical user interface for your app. You’ll use the modified Birthday Cake project you created in the previous lab.

**Logistical Note:** Completing the first two checkpoints takes many students up to two hours. Completing the remaining four checkpoints generally takes about half an hour. It is important that you finish this entire lab before next week because the next lab continues to build upon the work in this lab.

# Initial Layout

Begin with the BirthdayCake application that you modified in the previous lab.

Modify the layout so that it also includes:

* a second Switch view labeled “Candles”
* a SeekBar
* a TextView above the SeekBar that says “How many candles?”

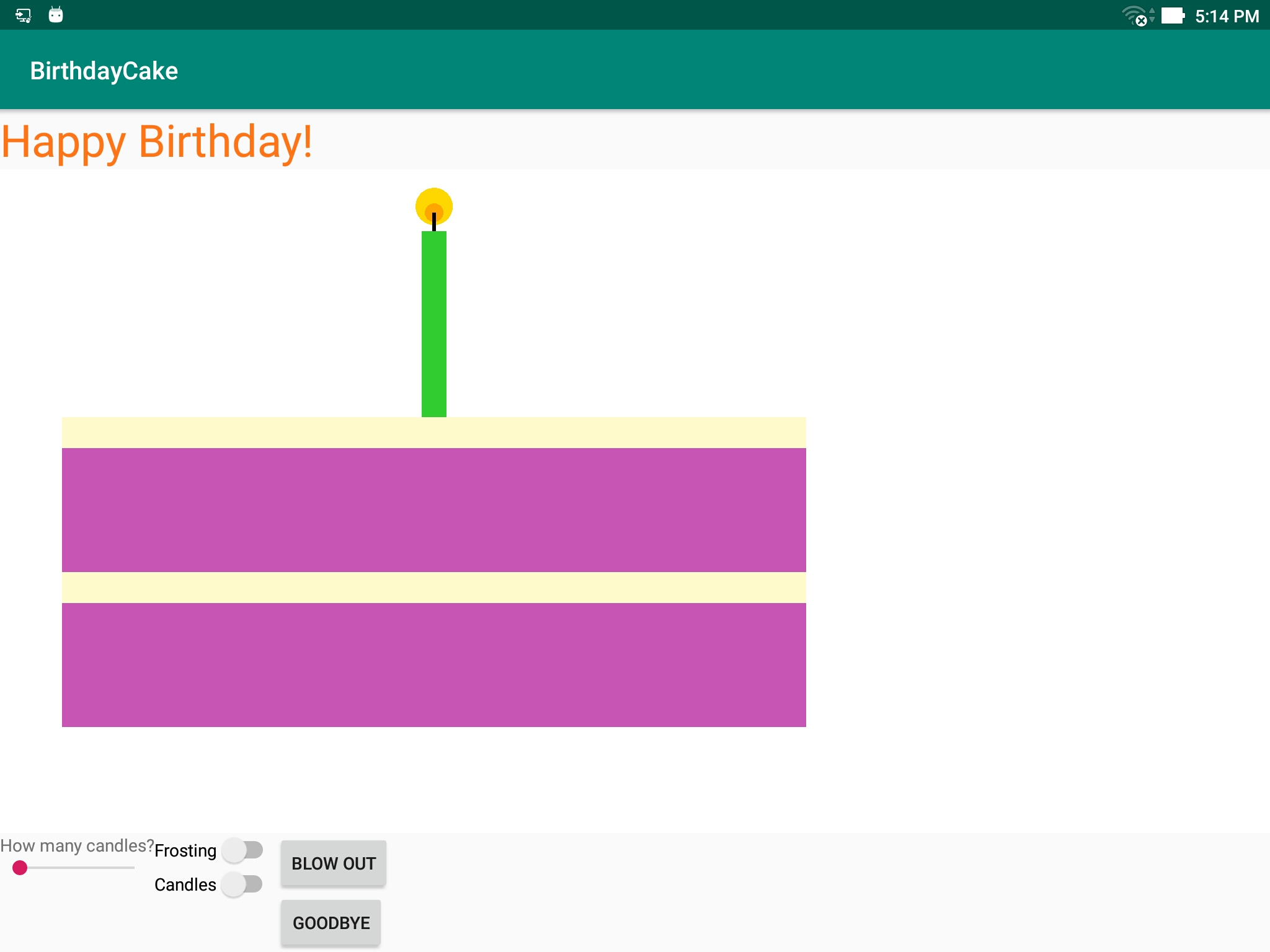
Change the text on the buttons to say “Blow Out” and “Goodbye” respectively.

Use Linear Layout objects to arrange the views as follows:

* The “Happy Birthday” text should remain by itself at the top of the display
* The birthday cake drawing should be just below the “Happy Birthday” text. It’s okay (for now) if the cake is partly obscured as long as some of it is visible. All other views must be fully visible.
* The remaining views should be below the cake in three columns:
  + column 1 will contain the TextView and SeekBar views
  + column 2 will contain the Switch views
  + column 3 will contain the Button views

Your layout does not have to be pretty. It’s only required that all views be visible and arranged as described above. The screenshot below shows one possible solution (except your cake should have two candles on it already).

**Hints:**

* What you see in the “Design” tab on Android Studio is not necessarily what you will actually see when you run the app. Be wary.
* You will need a combination of horizontal and vertical LinearLayouts to accomplish this.
* Remove any “layout\_weight” parameters from everything (for now) to guarantee nothing gets pushed off the screen
* Similarly, use “wrap\_content” rather than “match\_parent” for everything unless it is obviously okay.

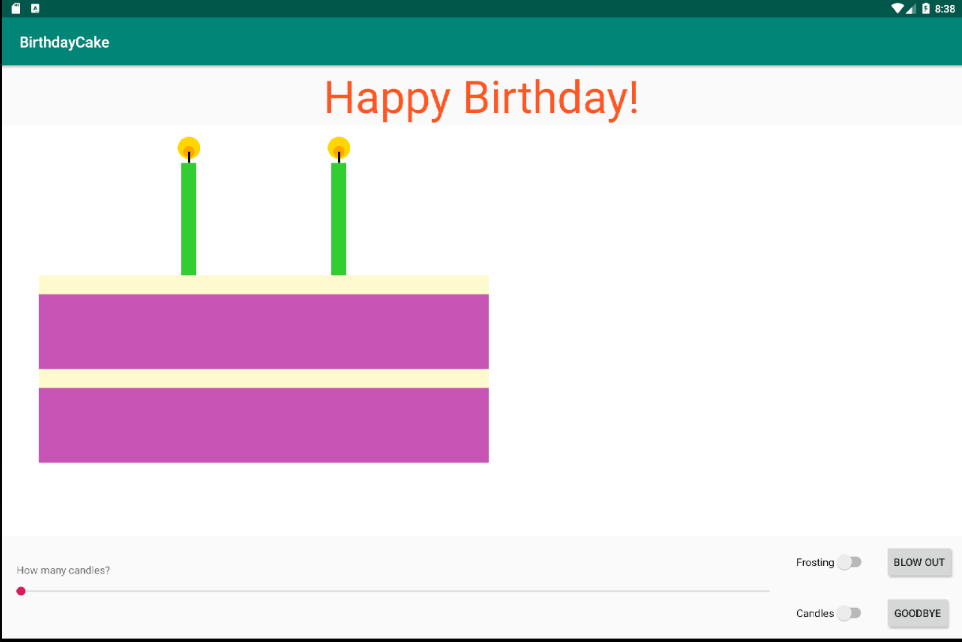
**Checkpoint 1 (25 points):** Show your instructor or lab assistant your initial layout running on your tablet and the changes you made in xml.

# Arranging Views

Your next goal is to arrange the views on the screen in a more pleasing way. The specific requirements are as follows:

* All views, except the cake, must be fully visible. It’s okay (for now) if the cake is partly obscured as long as some of it is visible. All other views must be fully visible.
* The “Happy Birthday!” text should be centered
* You may not use hardcoded values for x,y positions or layout\_width/height.
* Do not drag items in the layout view to change their size because that hardcodes the dimensions.
* The SeekBar should be at least three times wider than the Button or Switch views.
* There should be reasonable spacing between the views. None should be pushed up against each other.
* The buttons should be as far to the right as possible while still having a small gap so they are not pushed directly against it.
* The SeekBar and TextView should not be pushed directly against the left side of the screen.

The screenshot below shows one possible solution.



**Hints:**

* Avoid using the Space object to arrange your views. It won’t work well as that’s not how it’s intended to be used.
* You might have some success by applying layout\_weight to one or more of your linear layouts.
* You can apply layout\_margin specifications to controls or padding to layouts to space them out
* The layout\_gravity property may also be helpful.

**Checkpoint 2 (30 points):** Show the instructor or lab assistant your app running on your tablet with an appropriate layout and the changes you made in xml.

# Miscellaneous Tweaks

Make the following adjustments to your layout:

* The Switch views should both begin in the “on” position.
* The maximum value of the seekbar should be 5
* The starting progress of the seekbar should be 2 (matching the initial number of candles on your cake)

**Checkpoint 3 (15 points):** Show your instructor or lab assistant the changes you made to your .xml source code to cause the above changes to occur.

# First Taste of Events

When the user presses a button, it generates a “click” event. A method can register itself to list to such events if you want it to and then react in some way.

Locate the *onClick* property of your GoodBye button and give it the value *goodbye* (no quotes, as it is not a string).

Now locate the MainActivity.java file in your application. This class currently only contains a single method named onCreate. Add a second method to the class that looks like this:

**public void** goodbye(View button) {

System.***out***.println(**"Goodbye"**);

}

You will also have to import android.view.View; to support your use of that type.

**Logcat**

If you’ve done everything correctly, your program should print “Goodbye” when you click the button. But *where* is the text printing to?? At the bottom of your Android Studio window locate and click the button labeled “6: Logcat”. The Logcat pane should appear at the bottom of your window. Logcat is where status messages, error message and also console output appears when you run your app. (Note: If you can’t find the LogCat pane, try selecting View→Tool Windows→LogCat from the menu.)

If your app is running right now, you’ll see a stream of messages going by. To stem this tide, locate the dropdown box at the top of the Logcat pane is a selection box (aka, combo box or spinner) that is currently set to “Verbose.” Set this “Info” instead.

Now run your app and press the Goodbye button. The GoodBye message should appear in LogCat.

Using System.out.println() is not the recommended way to print console messages in Android (though it is supported). A better way is to use the Log object. Replace the System.out.println(“Goodbye”) line in your code with Log.i(“button”, “Goodbye”) instead. The first argument is a tag and the second is the message itself. This allows you to filter messages that have a certain tag to make monitoring an application easier. You will also have to import android.util.Log; to support your use of that type.

Notably the log option also has Log.d(), Log.e() and Log.v() methods which become visible when LogCat is filtering “Debug”, “Error” or “Verbose” messages, respectively.

Re-run your program and verify that the message prints.

**Checkpoint 4 (10 points)** Show your instructor or lab assistant the “Goodbye” messages in your Logcat.

# Diagnosing a Crash

Inevitably, the apps you write in CS301 will crash from time to time. It’s good to know how to respond when that happens.

To practice, we’re going to deliberately introduce a bug into your program that will cause it to crash. Specifically, rename your goodbye() method to something else so that the name does not match the onClick property on the button.

Now run your program again. When you hit the Goodbye button, the app should disappear and the tablet will display a message that “Unfortunately, BirthdayCake has stopped.”

Now look at your Logcat output. You should see lots of red text. In particular, a stack trace has been printed. Locate the exact line in this that correctly diagnoses the bug.

**Checkpoint 5 (10 points)** Show your instructor or lab assistant the line you found. Also fix your code so it won’t crash anymore.

# Android Debugger

While logcat is useful, serious debugging requires a debugger. Place a breakpoint on the Log.i() line in your goodbye() method by clicking on the gray border next to the line number. A small red circle should appear there.

For breakpoints to work, you must run your app in the debugger. From the menu select Run→Debug ‘app’. (You may need to terminate your currently running app first.) Wait for your program to run and then press the Goodbye button in your app. The app should stop the Log.i() line will be highlighted indicating that you’ve reached that breakpoint. Furthermore, the debug pane will appear in the Android window at the bottom (likely replacing Logcat).

In the debug pane, locate the buttons for the following actions:

* step to the next line in this file (called Step Over)
* resume program execution

In the debug pane, locate the value of the following variable:

this.mThemeId.

**Checkpoint 6 (10 points)** Show your instructor or lab assistant the items listed above.

**Optional:** If you’d like your birthday cake app to actually close when someone hits the Goodbye button, you can add a call to finishAffinity(); below the Log.i() call in your goodbye() method.

Congratulations! You’ve completed this lab. Be prepared to learn a lot more about event handling next week.