Lab 03 Android Event Handling

In this lab, we will continue to work with the Birthday Cake app. Load the latest version of this app to begin.

In software, an event is something that occurs externally to the app that the software needs to react to. The simplest example is a key press or a mouse click.

When writing code to handle events in Android, you generally follow three steps:

1. Identify the specific type of event you want to handle and the Java interface you must implement to handle it.
2. Implement the interface in a particular class within your program and implement the required methods in that class.
3. Register an instance of the class with the source of the events. For Android apps, the most common source of events is a View (e.g., a Button).

**Fair warning:** If you are still shaky in your understanding of classes, objects, inheritance, constructors, etc. then this lab will be a challenge for you, and you may fall behind. Be prepared to spend extra time outside of lab to catch up.

# Model-View-Controller

Before you can begin, you need to identify or create some additional classes for your program. A common design pattern for a GUI application is called Model-View-Controller. Typically, you would have a separate class for each of these three roles:

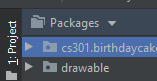
* The Model contains information about the state of the app.
* The View consults the model and renders the app to reflect the information therein.
* The Controller responds to user interactions (events) and changes the model to reflect those commands.

For an app as small as Birthday Cake, you could easily combine the model, view, and controller functionality into a single class. However, to help you learn about this distinction we will implement them in different classes in this lab.

In the following steps, do not use the static reserved word anywhere in your code. If you feel inclined to do so, it is likely because you have a fundamental misunderstanding about the relationship between classes and objects.

For the Birthday Cake app, your View is the CakeView class that has already been created for you. Add two new classes for the Model and Controller to your application by following these steps:

1. To create a new class, you can do either of the following:
   1. Select the Packages view from the Project tab in the top left.



Then right-click on the cs301.birthdaycake package in the project pane and select New→Java Class.

* 1. For other views, navigate to the java directory and then the cs301.birthdaycake directory. Then right-click on the cs301.birthdaycake directory and select New→Java Class.

Add two classes to your project in this way. Name one class CakeModel and the other class CakeController. Be sure to add these new files to your git repository.

1. The CakeModel class is a strictly data-containing class. It will contain information about the current state of the cake. Specifically add public instance variables that contain the following information:
   1. a boolean indicating whether the candles are lit or not. This should begin as true.
   2. an integer indicating how many candles are on the cake. This should initially be 2.
   3. a boolean indicating whether the cake has frosting or not. This should begin as true.
   4. a boolean indicating whether the cake has candles or not. This should begin as true.
2. Declare a private instance variable in the CakeView class of type CakeModel. Then initialize this variable with a new CakeModel object when CakeView object is created (aka in the CakeView constructor).
3. Add an accessor (getter) method to CakeView that allows other objects to retrieve a reference to the CakeModel object.
4. Your controller class should have a private instance variable that contains a reference to a CakeView object. This must be the CakeView that was created by Android when it inflated the GUI specification in your activity\_main.xml file. *So, do not create a new instance of the CakeView class to initialize that variable.* Instead, add a constructor to the CakeController class that expects to receive a reference to a CakeView object. Use this reference to initialize the private instance variable.
5. Add a second private instance variable to your CakeController class that is a reference to a CakeModel object. The CakeController and CakeView must share the same CakeModel object. *So, do not create a new instance of the CakeModel class to initialize that variable.* Instead, call the getter method you wrote to get a reference to the CakeModel object that was created in CakeView. This should also occur in the CakeController constructor.
6. In the onCreate() method in your MainActivity class, the GUI is being created via a call to a method named setContentView(). When that method is called, the app examines your activity\_main.xml file and creates the GUI you’ve defined there. In particular, it creates the Button, Seekbar, TextView, CakeView, Switch, etc. objects and arranges them as you’ve described.

When writing Android apps, you will frequently find yourself wishing to retrieve a reference to the objects in the GUI. The method for doing so is called findViewById(). Locate the Android SDK documentation for this method, then use it to retrieve a reference to the CakeView object that has been created in the GUI and place it in a local variable. Your new code should appear in an appropriate place in the onCreate() method. To accomplish this, you will need to pass in the unique ID number of the object. To find out how do to so, you will have to read the documentation. (Gasp!)

1. The next line of code in onCreate() should create a new CakeController object. Presuming you have implemented the constructor correctly, you should need to pass in the CakeView reference you retrieved in the previous step.
2. Run your app to verify that all your code is working. Visibly, your app should not have changed. Use the debugger to verify that the model, view and controller are all being created. All you have to do is show that those three objects are being initialized in the Variables pane of the debugger.

**Checkpoint 1 (25 points):** Show your instructor or lab assistant how you used the debugger to verify that your model, view and controller classes are all being instantiated.

# Button Event

In the previous lab, you used the onClick property to cause a function in your app to be called whenever the GoodBye button was pressed. This property is handy for buttons but generally not useful for most other types of events. In this section you will write a “proper” event handler for the other button without using this shortcut.

Above we defined three steps for handling an event. These steps apply to this situation as follows:

1. Locate the View.OnClickListener interface in the Android SDK documentation. Implement this interface in the CakeController class. This will require you to create an onClick() method in your class. Put a Log.d() call in the method so you can verify that you are receiving the clicks once all the steps are complete.
2. Add code to the bottom of your onCreate() method in the MainActivity class that performs the following:
   1. retrieve a reference to the “Blow Out” button view on the user interface
   2. call the setOnClickListener() method on the button to register your CakeController object as a listener for clicks on this button

If you implemented this correctly, your Log.d() method should appear when you run the app and press the Blow Out button. If you’re not seeing the log statement (and especially if you’re not seeing any log statements), be sure the dropdown menus in the Logcat window are set properly. The first dropdown should show the name of the device you’re testing on (emulator or tablet). The other dropdowns should look as follows. Make sure the search box is empty.





Now, let’s have the BlowOut button actually blow out the candles! Follow these steps:

1. Your onClick() method should modify the model to indicate that the candles are not lit. The candles and the wick should remain though.
2. Call the invalidate() method on the CakeView object to let it know that it needs to redraw itself.
3. Modify the drawCandle() method in the CakeView class to so that it will not draw the candle flame if the model indicates they should not be lit.

**Hints:**

* Your app is not behaving as you wish it to? Before you ask for help, use the debugger to verify that the code is executing correctly.

**Checkpoint 2 (25 points):** Show the instructor or lab assistant your app running on your tablet and behaving as prescribed.

# Candles Switch

Modify your app so that the Candles switch allows the user to decide whether or not the candles (including the wick) will be drawn on the cake. You may not use View.OnClickListener for this event. Instead, implement the CompoundButton.OnCheckedChangeListener in your controller class and use the corresponding method to create the behavior that you want.

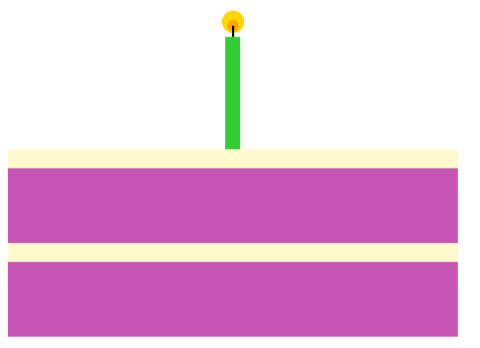
Otherwise, you should be able to follow the same general steps as above to get the desired behavior.

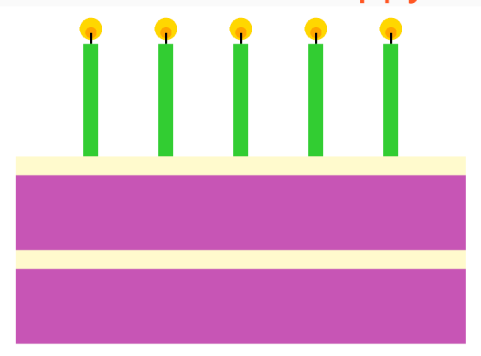
**Checkpoint 3 (25 points):** Show the instructor or lab assistant your app running on your tablet and behaving as prescribed and that no previous functionality has been lost. Demonstrate that your candles can be enabled and disabled before they are blown out and after they have been blown out.

# SeekBar

The SeekBar should change how many candles are on the cake. There can be a minimum of 0 and a maximum of 5. You will have to find the appropriate interface for this on your own. When you find it, you will discover there are three methods you have to implement. However, you only need one of these. The remaining two can be “do nothing” (empty) methods, but they still must be defined.

All of the candles need to be entirely on the cake (no partially hanging candles) and evenly spaced out without hardcoding values or using if/switch statements. See if you can center the candles on the cake, like the examples below.





**Checkpoint 4 (25 points):** Show the instructor or lab assistant your app running on your tablet and behaving as prescribed and that no previous functionality has been lost.

# Challenge!

Congratulations. You’ve completed this lab. However, you are encouraged to give yourself a more robust understanding by tackling one or more of the following:

* When the Blow Out button is pressed, the text on the button changes to “Re-Light.” Pressing that button relights the candles and changes button to “Blow Out” again.
* Have the Frosting switch do something fun. To handle this, you’ll need to modify your onCheckedChanged() method to detect which switch has been hit before it will know what to do. How might you accomplish this?