Module: Mobile Application development (Android)

Session 15: User Interface Library (Practice)

### Intro

### 

### In this lab we will be learning how to use and extend the Android user interface library. In a number of ways it is very similar to the Java Swing library, and in perhaps just as many ways it is different. While being familiar with Swing may help in some situations, it is not necessary. It is important to note that this lab is meant to be done in order, from start to finish. Each activity builds on the previous one, so skipping over earlier activities in the lab may cause you to miss an important lesson that you should be using in later activities.

### Objectives

### At the end of this lab you will be expected to know:

### What Views, View Groups, Layouts, and Widgets are and how they relate to each other.

### How to declare layouts dynamically at runtime.

### How to reference resources in code and from other resource layout files.

### How to use Events and Event Listeners.

### Activities

### For this lab we will be creating a "Joke List" application. It is a simple app that allows a user to view and edit a list of jokes. All tasks for this lab will be based off of this application. Over the course of the lab you will be iteratively refining and adding functionality to the Joke List app. With each iteration you will be either improving upon the previous iteration's functionality, or you will be implementing the same functionality in a different way.

### IMPORTANT:

### You will be given a Skeleton Project to work with. This project contains all of the java and resource files you will need to complete the lab. Some method stubs, member variables, and resource values and ids have been added as well. It is important that you not change the names of these methods, variables, and resource values and ids. These are given to you because there are unit tests included in this project as well that depend on these items being declared exactly as they are. These unit test will be used to evaluate the correctness of your lab. You have complete access to these test cases during development, which gives you the ability to run these tests yourself.

### 1. Setting Up...

### 1.1 Creating the Project

### To begin, you will need to download and extract the skeleton project for the JokeList application.

### Click [Here](https://sites.google.com/site/androidappcourse/resources/lab-skeletons/Lab2_v7_stub.zip?attredirects=0) to download the skeleton project.

### Extract the project, making sure to preserve the folder structure.

### *Take note of the path to the root folder of the skeleton project.*

### *You may prefer to extract it to your Eclipse workspace directory.*

### Next you will need to setup a "Joke List" Android project for this app. Since the skeleton project was created in Eclipse, the easiest thing is to import this project into Eclipse.

### Select File -> Import.

### In the Import Wizard, expand General and select Existing Projects into Workspace. Click Next.

### In the Import Project wizard, click select root directory and click Browse. Select the root directory of the skeleton project that you extracted. Click Open and then Finish.

### Click on the project name in the Package Explorer. Select File -> Rename and change the name of your project to lab2<userid> where <userid> is your user id (e.g. jsmith).

### 1.2 Fill in the Joke Class

### Throughout the lab you will be working with the Joke object class. This class will be used to encapsulate two items pertaining to jokes.

### m\_strJoke: This represents the actual text of the joke.

### m\_nRating: This represents a rating that can be assigned to a joke. There are three possible values that the rating can take:

### UNRATED: indicates no rating has been assigned to this joke by the user.

### LIKE: indicates the user liked this joke.

### DISLIKE: indicates the user did not like this joke.

### Begin by filling in this class:

### Open the Joke.java file

### Under the src/ folder in the edu.calpoly.android.lab2 package.

### Fill in all methods marked //TODO.

### *Read the comments if you are confused as to the purpose of any of the methods.*

### Run the JokeTest.java Unit Tests to ensure that you have properly filled in this class.

### Locate the JokeTest.java file under the test/ folder in the edu.calpoly.android.lab2 package.

### Right click the file, select Run as -> Android JUnit Test.

### This should open up a JUnit Tab and if you see the Green bar, you've passed all the tests and you can continue on to the next section:

### https://sites.google.com/site/androidappcourse/_/rsrc/1263273339077/labs/lab-2-1/TestSuccess.png

### If you see a red bar, this means that you've failed one or more tests. The tab should show how many tests failed, and which tests you failed (*circled in yellow*):

### https://sites.google.com/site/androidappcourse/_/rsrc/1263273363241/labs/lab-2-1/TestFailure.png

### Feel free to open up the JokeTest.java file to look at what exactly is being tested. Make the appropriate corrections and rerun the tests until you've passed all the tests.

### There are additional tests in the SimpleJokeListTest.java file, but don't run those until later (see section 3.2.3).

### 1.3 Retrieving the Joke Resources Strings

### The skeleton project has been pre-populated with an array of three different String resources you can use as sample jokes. For a complete background on Resources and how to properly use them you can see the [Android Developer Guide on Resources](http://developer.android.com/guide/topics/resources/resources-i18n.html). For an overview:

### Open up res/values/strings.xml in the XML editor to view the joke resources.

### Notice the <string-array name="jokeList"> element.

### This is how you declare an array of strings.

### Notice the name attribute.

### When you add an element to a resource file, the Android Development Toolkit will automatically add a static constant with this variable name to the R.java file.

### You can then use this constant as an identifier for retrieving the resource element through a Resource Object.

### Open up the R.java file under the gen/ folder in the edu.calpoly.android.lab2 package.

### Notice that R is a static class with a static subclass for each type of resource element.

### If you add a string resource element and give it a name attribute, a static constant with this name gets added to the R.string class. Arrays get added to the R.array class, drawables get added to the R.drawable class, etc.

### The constant has the same name as the name attribute. Layouts are slightly different in that you don't have to specify a name attribute; the name of the constant will be the name of the XML layout file.

### The constant contains the resource id that you can use to retrieve the resource.

### You need to display these jokes when the application starts up. When an Activity first starts up, its "public void onCreate(Bundle savedInstance)" method is always called. This method allows you to initialize the Activity. Right now you will only be initializing local variables to hold the jokes that you just entered:

### Open SimpleJokeList.java.

### In the public void onCreate(Bundle savedInstance) method:

### Notice the super.onCreate(savedInstance) call. This is crucial, never forget to make this call. If you don't your Activity won't work.

### Make a call to this.getResources() , which will return a Resources object.

### The Resources class provides an interface for retrieving resources by their resourceID.

### resourceID's can be found in the static R class, under their respective resource subclasses (which are named after their resource type), under the name given to them in the resource file: R.array.jokeList

### Create an instance of the Joke ArrayList.

### Retrieve the array of joke strings by calling getStringArray(R.array.jokeList) on the resource object.

### For each of these strings make a call to addJoke(...), which will initialize Joke objects and place them in m\_arrJokeList.

### Fill in the addJoke(...) method.

### When creating a new Activity you will almost always override the onCreate method. The onCreate method, as you will see later, is the place where you will be creating your User Interfaces, binding data to different UI controls, and starting helper threads (this will not be covered in this lab). Also, take note of the "Bundle savedInstance" parameter that gets passed in. This variable contains information on the previous state of the UI. As you may have assumed, this means that you are able to save the state of the UI before it closes so that it can be initialized to the same state the next time it is created.

### 2. Brief Background on View Classes

### In Android, a user interface is a hierarchy composed of different View objects. The View class serves as the base class for all graphical elements, of which there are two main types:

### Widgets: Can either be individual, or groups of UI elements. These are things like buttons, text fields, and labels. Widgets directly extend the View class.

### Layouts: Provide a means of arranging UI elements on the screen. These are things like a table layout or a linear layout. Layouts extend the ViewGroup class, which in turn extends the View class.

### Layouts are all subclasses of the ViewGroup class and their main purpose is to control the position of all the child views they contain. Layouts can be nested within other layouts as well, to create complex user interfaces. Some of the common layout objects you will use are:

### FrameLayout: Takes a single view object and simply pins it to the upper left hand corner of the screen. Each child view that is added is drawn on top of the previous one, causing it to be completely or partially obscured.

### LinearLayout: Has a list of child views and draws them sequentially in a single direction, either horizontally or vertically. You have the option of assigning a weight value for each child view which determines how much it is allowed to grow if there is extra space.

### TableLayout: Positions it's child views in a grid of rows and columns. A row is a child view specified by the TableRow class. TableRows can have zero or more cells and can contain empty cells. However, a cell cannot span multiple columns. Each cell is itself another view, like a Button, and can be set to shrink or grow.

### RelativeLayout: Positions it's child views relative to other child views or the parent view. For example, two child views can be left-justified in the parent, or one child view can be made to be below another.

### AbsoluteLayout: Has an absolute coordinate position for each child view. This is a rigid layout that pins all child views down exactly where they are specified. Using this does not allow the user interface to adjust for different screen sizes and resolutions.

### *See* [*Common Layout Objects*](http://developer.android.com/guide/topics/ui/layout-objects.html) *for more information...*

### Declaring the layouts for your user interface can be done dynamically (in code), statically (via an XML resource file), or any combination of the two. In the following subsection you will build a set of user interfaces in code. In a future lab, you will build a set of user interfaces in XML.

### 3. SimpleJokeList

### Work done in this section will be limited to the SimpleJokeList.java file. SimpleJokeList is an Activity class which displays a vertical scrollable list of all the Jokes in m\_arrJokeList. Additionally, it has the ability to add jokes to m\_arrJokeList via a text field and add a button that floats at the top of the screen.

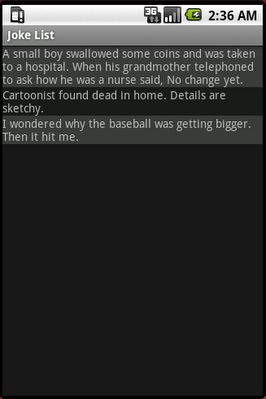
### 

### 3.1 Declaring Dynamic Layouts in Code

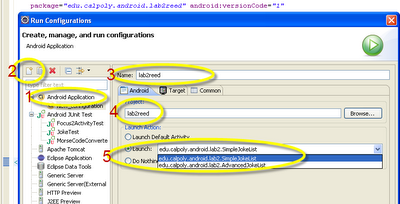
### If you've ever built UIs in Java using Swing, this should be somewhat familiar. You generally want to try to define the layouts for your interface using XML when possible, but there are plenty of instances where it is still necessary to do it at run-time in code. For instance, you may want to dynamically change the layout based on some type of user input. Never the less, working with the actual code is good practice for understanding how the different Layout classes work and what sorts of interfaces all the various controls offer.

### 3.1.1 Display a Scrollable Vertical List

Your first Task is to simply display each of the jokes in the **strings.xml** resource file in a scrollable vertical list. When finished your application should look something like this:



* Fill in the **initLayout()** method in SimpleJokeList.java:
  + Initialize the **m\_vwJokeLayout** LinearLayout member variable.
    - *When constructing View objects, you generally have to pass a* ***Context*** *object into the constructor. A Context object provides the functionality for accessing resources (like our jokeList), databases, and preferences. The Activity class inherits from Context which is how we were able to retrieve our jokeList by calling this.getResources().*
  + The LinearLayout class displays its child views horizontally by default. Make sure to change this to vertical using the setOrientation() method, passing in the LinearLayout.VERTICAL constant.
    - *LinearLayout:* [*[Docs]*](http://developer.android.com/reference/android/widget/LinearLayout.html)
  + Create a local ScrollView object.
    - *A ScrollView is merely a FrameLayout that becomes scrollable when its child views are larger than the screen area.*
    - *It generally has a single child view, which in our case will be another layout manager*
  + Add the LinearLayout to the ScrollView by calling its addView(...) method.
  + Make a call setContentView(...), passing in your ScrollView.
    - *setContentView provides the content which the Activity is supposed to display. Our UI is a hierarchical structure of nested components. We want to pass in the top-level, or root, element to this method. In our case this is the ScrollView.*
* Make a call to **initLayout()** in your **onCreate(...)** method and place it above your retrieval and initialization of the jokes in the **strings.xml** resource file.
* Update the **addJoke(...)** method.
  + Add the joke text to a new TextView object by calling its setText(...) method.
    - *A TextView is used for displaying text on the screen.*
  + Add the TextView to **m\_vwJokeLayout**.
* Your **AndroidManifest.xml** is currently setup so that SimpleJokeList Activity will launch on startup. Create a Run Configuration to specifically launch the SimpleJokeListActivity:
  + Select **Run -> Run Configurations** from the menu.
  + Select the **Android Application** item from the list.
  + Click the **New Launch Configuration** icon
  + Give the configuration a name and the name of your android project.
  + Select the **Launch** radio button and select the **SimpleJokeList** activity from the dropdown list.



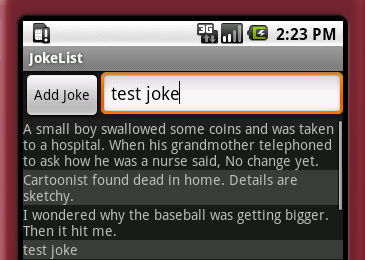
* Now run your application to verify that your list of jokes shows up.
* The text is a little small, so use **setTextSize(...)** to increase the size of all jokes to 16 font.
* It's somewhat hard to distinguish where one joke ends and another begins, so let's alternate the background colors of each joke. We'll store the color values as resources:
  + Create a new XML Resource file under **/res/values** like you did in Lab 1 and call it **colors.xml**
    - *Hint:* ***File->New->Android XML File***
  + You can use either the Resource Editor or you can directly add the following XML to the file:

**<color name="dark">#3D3D3D</color>**   
 **<color name="light">#1F1F1F</color>**

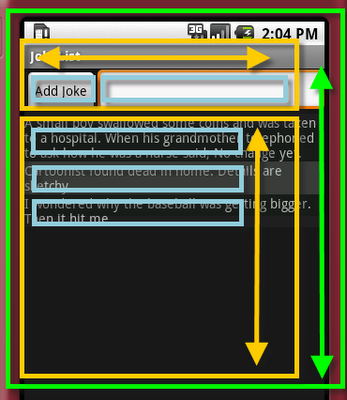
* + - *If adding the XML yourself, make sure to nest the color tags inside the <resource> tags.*
  + Modify the background color of the TextViews by calling **setBackgroundColor** as you create them in your **addJoke(...)** method.
    - Alternate between setting the background color to **dark** and **light**.
    - *Hint: Retrieve the colors in the same way you retrieved the joke strings.*

### 3.1.2 Adding Your Own Jokes

In this next task we will give the user the ability to enter their own jokes by adding a text field and a button to the UI. The text field and button should float at the top of the screen, above the list of jokes. When you scroll through the jokes, the text field and button should not scroll, but remain at the top of the screen.



In order to accomplish this we will walk through a process of nesting LinearLayouts within each other to achieve the desired effect. However, feel free to experiment and use whatever type of ViewGroup Layout you want, so long as the functionality and appearance remains the same. In the figure shown below, you can see how a ***root*** *vertically-oriented-LinearLayout* containing a *horizontally-oriented-LinearLayout* and the *ScrollView* of jokes can be combined to achieve the effect we want.



In this step you will only be updating your Layout. The new "Add Joke" button and text field won't work yet. You will add code to hook-up those controls in section *3.2 Simple Event Listeners.*

* *All of the work here, will be done in the* ***initLayout()*** *method. Additionally, all but a few lines of that code should be written above the code you wrote in 3.1.1.*
* Declare a local vertically-oriented LinearLayout object, this will be your root ViewGroup *(displayed in green in the figure above)*.
* Declare a local horizontally-oriented LinearLayout object, this will be the ViewGroup containing the "Add Joke" button and text field.
  + Initialize your **m\_vwJokeButton** member variable, setting its text to "**Add Joke**".
  + Add the Button to the horizontal LinearLayout.
  + Initialize your **m\_vwJokeEditText** member variable.
  + call **m\_vwJokeEditText.setLayoutParams(...)**, passing in an appropriately initialized LinearLayout.LayoutParams object so that the EditText will take up all of the extra available width and height.
  + *For more information see the Android Developer Guide and Documentation on:*
    - [Layout Parameters](http://developer.android.com/guide/topics/ui/declaring-layout.html#layout-params)
    - [How Android Draws Views](http://developer.android.com/guide/topics/ui/how-android-draws.html)
    - [LinearLayout.LayoutParams](http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html)
  + Add the EditText to the horizontal LinearLayout.
* Add your horizontal LinearLayout to your root ViewGroup.
* After your declaration and initialization of your ScrollView ViewGroup, add the ScrollView to your root ViewGroup.
* Change your call to **setContentView(...)** to pass in your new root ViewGroup.
* Run your application to test that you have properly setup your layout.

### 3.2 Simple UI Event Listeners

### Now that your UI has the components necessary to allow users to enter new jokes, you need to hook these components up. In this section you will define and register event listeners to handle adding new jokes.

### Read the Android Developers Guide on [Handling UI Events](http://developer.android.com/guide/topics/ui/ui-events.html) to get a detailed background on event handlers and listeners. In short, you use event listeners to respond to user interaction with your UI. There are two general steps for doing this. The first is to define an object that implements an interface for the type of interaction you want to respond. The second step is to register that object with the UI control you want to respond to. For example, to react to a particular Button being clicked, you have to register an object that implements the OnClickListener interface with the Button you want to listen to.

### 3.2.1 Adding a Button Listener

Setup the **onClickListener** for the "Add Joke" Button. When the user hits the "Add Joke" Button a new Joke object should be initialized with the text from the EditText field and added to your joke list. If the EditText is empty, then the Button should do nothing. In **initAddJokeListeners()**:

* Call **setOnClickListener** method for you **m\_vwJokeButton** member variable. Pass in a reference to what's called an *Anonymous Inner Class* that implements the OnClickListener interface.
  + *If you are unfamiliar with anonymous inner classes that's alright. Essentially, its an inline way of creating a one-time-use class that implements some interface. You declare the class and instantiate it in one motion. You can read more about them from Sun's Java Tutorials on* [Anonymous Inner Classes](http://www.google.com/url?q=http%3A%2F%2Fjava.sun.com%2Fdocs%2Fbooks%2Ftutorial%2Fuiswing%2Fevents%2Fgeneralrules.html%23innerClasses&sa=D&sntz=1&usg=AFrqEzdr8GEfHhhdijtDoaRBLwKJnq5wSg)*.*
  + Copy the following code:

**m\_vwJokeButton.setOnClickListener(new OnClickListener() {**

**@Override**

**public void onClick(View view) {**  
  
**//Implement code to add a new joke here...**  
  
**}**   
**});**

* + Fill in the **onClick** method in the anonymous inner class that you created to add a new joke to the list.
    - Retrieve the text entered by the user into the EditText by calling **getText().toString()** (*Check for empty strings here).*
    - Clear the EditText.
    - Call your **addJoke** method
    - Add the following two lines of code to hide the Soft Keyboard that appears when the EditText has focus:
  + **InputMethodManager imm = (InputMethodManager)  
     getSystemService(Context.INPUT\_METHOD\_SERVICE);  
      
    imm.hideSoftInputFromWindow(m\_vwJokeEditText.getWindowToken(), 0);**
  + Call initAddJokeListeners from onCreate.
* Run your application to ensure that the "Add Joke" Button functions properly now.



* *In this situation you had two options for where you could implement the code to handle the "Add Joke" button's onClick event. Each has its drawbacks:*
  + *Use an* ***Anonymous Inner Class****:* 
    - *This option is a little cleaner and more readable since the event handling logic for different elements is self contained and referenced where it is used.*
    - *Since this event listener is only being used for one UI control, you are free to make certain assumptions when implementing it. For instance, you know exactly which View generated the event.*
    - *However, this will consume more resources because this instantiates a new object, and class information for the object will need to be stored as well.*
  + *Make* ***SimpleJokeList*** *implement the OnClickListener interface:* 
    - *In this option you would be able to register SimpleJokeList to respond to Click events from many different UI Controls. However, you would have to add conditional logic for determining which view fired the onClick event.*
    - *This has a clear performance benefit in that you don't have to load extra classes, which reduces your memory footprint and load time.*
    - *This is actually very common in Android applications due to the need to design for performance imposed by limited resources.*

### 3.2.2 Adding Key Listeners

Your next task is to add event listeners that respond to certain keys being pressed on your own. When the user is entering their joke into the EditText and presses either the "Enter/Return" key or the "DPad-Center/Track-Ball" key, the application should respond exactly the same as it does when the "Add Joke" Button is pressed.  
  
Add this code to the **initAddJokeListeners()** method. Remember that after the joke gets added, the soft-keyboard should disappear.  
 *Hints:*

* *Android* [OnKeyListener Interface](http://developer.android.com/reference/android/view/View.OnKeyListener.html) *Documentation*
* *Android* [KeyEvent Constants](http://developer.android.com/reference/android/view/KeyEvent.html#constants) *Documentation: ACTION\_DOWN, ACTION\_UP, & KEYCODE'*

### 3.2.3 Run SimpleJokeList Tests

### Run the SimpleJokeListTest.java Unit Tests to ensure that you have properly filled in this class.

### Start the emulator or connect your device.

### Turn off the keyguard by pressing the menu button. *This is necessary for some of the UI tests to work.*

### Locate the file under the test/ folder in the edu.calpoly.android.lab2 package.

### Right click the file, select Run as -> Android JUnit Test.

### This should open up a JUnit Tab and if you see the Green bar, you've passed all the tests. It is always a good idea to run all your other tests (e.g. JokeTest.java) as well.

### 4. Deliverables

### Put your entire project directory into a .zip file and submit the archive. This effectively provides time-stamped evidence that you submitted the lab on time. The name of your archive should be lab2<yourname>.zip. So if your username is RAM and you created a zip file, then your file would be named lab2RAM.zip.