**Lab 3**

|  |
| --- |
| Go to <http://sites.google.com/site/androidappcourse/> for the most up-to-date versions of these labs.    **Intro**    This lab will be a continuation of Lab 2. You will expand on your knowledge of the Android user interface library. 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:   * How to declare layouts statically as an xml resource. * How to create custom Views from scratch to suit a specific need. * How to create Options and Context Menus. * How to use Adapters and AdapterViews to bind a View class to data. * How to establish Http connections. * How to create Dialogs and Notifications.   **Activities**  For this lab we will be extending the "Joke List" application that you created in Lab2. This version of the app will be more advanced. It will allow the user to give ratings to Jokes, delete Jokes, upload Jokes to a server, and download Jokes from a server. 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 units 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. In fact, you are encouraged to run these tests to ensure that your application is functioning properly.*  **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/androidcoursearchive/resources/lab-skeletons/Lab3_v1_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.*   Next you will need to setup a "Joke List" Android project for this app. You will do this in a manner similar to the way you did in Lab 2:   * In the "New Android Project" wizard, select the **Create project from existing source** option. * Point the **Location** field at the root directory of the skeleton project that you extracted.   Use the settings listed below for the remaining fields:   * **Project Name:**  lab3<userid> * **Build Target:**Android 1.6 * **Application Name:**Joke List   **1.2 Fill in the Joke Class**  You may fill in the Joke Class using the functionality that you implemented for this class in Lab2. However, there is one key difference. A member variable named **m\_strAuthorName** has been added to the class which will contain the name of the Joke's Author. In particular:   * You must update the constructors. You are required to pass in an Author name for all the Constructors except for the default constructor. * The **equals(...)** method now requires that the names of the Authors of the two Jokes being compared must match as well, in addition to their text. * There is a getter and setter that needs to be filled in.   Run the JokeTest.java Unit Tests to ensure that you have properly filled in this class.  **1.3 Use SimpleJokeList as a Starting Point**  You may fill in the AdvancedJokeList class using some of the code that you implemented for SimpleJokelist in Lab2:   * Fill in the **addJoke(...)** method using the code from **SimpleJokeList.addJoke(...)**.   + Note that the signature on the method has changed to accept a Joke object instead of a string. You will have to modify the **SimpleJokeList.addJoke(...)** code to use this new interface. * Fill in the **initAddJokeListeners(...)** method using the code from **SimpleJokeList.initAddJokeListeners(...)**.   + Remember that the signature on the **addJoke(...)** method has changed to accept a Joke object instead of a string. You will have to modify some of the code here to use this new interface. * Fill in the **onCreate(...)** method using the code from **SimpleJokeList.onCreate(...)**.   + Remember that the signature on the **addJoke(...)** method has changed to accept a Joke object instead of a string. You will have to modify some of the code here to use this new interface. * Run your application to ensure that it performs the way it did in Lab2.   **2 Declaring Static Layouts in XML**  Read the Android Developer Guide on [Declaring Layout](http://developer.android.com/guide/topics/ui/declaring-layout.html) for complete background on declaring layouts. Declaring your user interface in XML is the preferred method of implementation. By declaring your UI in an XML resource file it gives you better separation between the presentation layer of your application and the code controlling things underneath. One benefit of this is that modifications to your UI can be made without having to change any source code or recompile. This allows you to define different views for different screen sizes, resolutions, and scenarios while using the same code to control everything.  **2.1 Porting Your Dynamic Layout Into Static XML**  In order to get some practice with setting up layouts in XML, you will begin by converting the layout you setup dynamically in SimpleJokeList to an XML layout file. You will then inflate this layout in AdvancedJokeList and set it as your ContentView.   * Fill in the **/res/layout/advanced.xml** layout file:   + Make your **advanced.xml** layout file produce the *exact*same UI as the one you declared dynamically in SimpleJokeList.   + **advanced.xml** has been stubbed out for you already. It contains a FrameLayout as the root ViewGroup to prevent compilation errors, you will need to replace this with an appropriate root ViewGroup element.   + **IMPORTANT:** You must use the following resource id's for each of the UI Components listed. These UI Components are defined as member variables in AdvancedJokeList.java the same way they were defined in SimpleJokeList.java:     - EditText m\_vwJokeEditText: "newJokeEditText"     - Button m\_vwJokeButton: "addJokeButton"     - LinearLayout m\_vwJokeLayout: "jokeListViewGroup" * Edit your **initLayout()**method to use the **advanced.xml** layout file:   + Remove all the code from this method. It should be an empty method when you start.   + You must make your call to **setContentView** be the first thing that you do.   + Instead of passing in a view, pass in the Layout Resource Id for **advanced.xml**.     - *Hint: You did this in Lab1, you can find it in the Static****R****class.*     - *You won't be able to retrieve references to your UI Controls until the layout has been inflated from the XML file.*   + Initialize your view class member variables by retrieving a references to them, instead of constructing new ones:     - **m\_vwJokeLayout**     - **m\_vwNewJokeEditText**     - **m\_vwAddJokeButton**     - *Hint: You did this in Lab1, remember the****findViewById****method?* * Try running your application. The UI should appear and function exactly as it did for SimpleJokeList.   **2.2 Building Custom UI Components**  Sometimes the standard View library will not supply the functionality that you need. In situations like this it is completely acceptable to define your own UI Components. There are three general approaches to creating custom UI components:   1. Creating a custom component from scratch. 2. Modifying an existing component to serve your needs. 3. Combining existing components to create a compound component.   In this section you will be using the third approach to develop a custom component. You will combine a number of different existing View classes to create a coherent Widget for displaying Jokes. For a complete background on this approach, as well as the other two approaches, read the Android Developer Guide on [Compound Controls](http://developer.android.com/guide/topics/ui/custom-components.html#compound).  The custom component that you are going to implement will have two states, an expanded and a collapsed state. It will look something like this:  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033548990/labs/lab-3/CustomJokeView.png?height=104&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/CustomJokeView.png?attredirects=0)  In the collapsed state there is an expand/collapse Button that displays a "+" and a joke TextView displaying the first two lines of the joke. If the joke is longer than two lines, it will only display two lines and append to the displayed text an ellipsis.  In the expanded state there exist the exact same components that existed in the collapsed state. Additionally, there will exist a RadioGroup containing two RadioButtons, that will appear horizontally centered underneath the expand/collapse Button and joke EditText. In the expanded state, the expand/collapse Button will display a "-" and the joke EditText will no longer display an ellipsis, but rather display the entire text of the joke (no matter how long it is). The expand/collapse button should remain anchored to the top left corner; it should not be centered vertically.  **2.2.1 Declare a Custom JokeView XML Layout**  The first step is to create the XML layout file that the custom component will use. You have to implement this custom component using a single XML layout file. Switching between the collapsed and expanded states is only a matter of hiding the RadioGroup, changing the Button text, and changing settings on the TextView.   * Fill in the **res/layouts/jokeview.xml**layout file:   + When writing the layout, write the file as though you were writing it for the expanded state. Don't worry about the collapsed state until the next section.   + **jokeview.xml** has been stubbed out for you already. It contains a FrameLayout as the root ViewGroup to prevent compilation errors.  You will need to replace this with an appropriate root ViewGroup element.   + **IMPORTANT:** You must use the following id's for this list of UI Components defined in AdvancedJokeList.java     - Button m\_vwExpandButton: "expandButton"     - RadioButton m\_vwLikeButton: "likeButton"     - RadioButton m\_vwDislikeButton: "dislikeButton"     - RadioGroup m\_vwLikeGroup: "ratingRadioGroup"     - TextView m\_vwJokeText: "jokeTextView" * Hints:   + You can test your UI without having to run your application by switching back and forth between the LayoutEditor and the XML Editor.     - Make changes in the XML Editor.     - The LayoutEditor will render your UI.     - Test your layout by setting the text of your TextView in the XML Editor to a really long string.   + If TextView keeps cutting the bottom off of really long text, try adding padding to the top and bottom.     - See the Android Documentation on the [paddingTop/paddingBottom](http://developer.android.com/reference/android/view/View.html#attr_android:paddingTop) XML attributes.   + Experiment with Relative Layout as your root ViewGroup, you can then nest LinearLayouts as you see fit.   + Empty LinearLayouts can also be used as spacers between components to help center things or provide adequate spacing.     - See the Android Developer Guide on [LinearLayout](http://developer.android.com/guide/topics/ui/layout-objects.html#linearlayout)for a discussion of weight.     - Look into the [layout\_weight](http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html#attr_android:layout_weight)XML attribute in the Android Documentation   **2.2.2 Create a Custom JokeView Widget**  The next step is to implement your custom component class. This class will be the JokeView class. It is your task to fill in JokeView.java that has been stubbed out for you. In general when creating a compound component, after you have established your layout, you want your component class to extend the class of the root ViewGroup in your layout. Your component class then becomes a special subclass of that ViewGroup.   Open up **JokeView.java**:   * Make the JokeView class extend the root ViewGroup of your layout. * Fill in the **JokeView(Context context, Joke joke)**constructor:   + Un-comment the call to **super(context)**. This was only commented to prevent compilation errors that would appear without having JokeView extend a View class.   + Inflate **joke\_view.xml**:     - This will be done differently than the way you've been doing it. In this particular context, after the layout is inflated, we want this JokeView object to be the root ViewGroup of the inflated layout.     - Copy the following code:   LayoutInflater inflater = (LayoutInflater)context.getSystemService(   Context.LAYOUT\_INFLATER\_SERVICE);  inflater.inflate(R.layout.joke\_view, this, true);   * + - Instead of returning an inflated hierarchy of Views, this JokeView object will become the root of that hierarchy.   + Initialize all the View component member variables by retrieving references to them as you would normally do for a layout declared in XML:     - **m\_vwLikeButton**     - **m\_vwDislikeButton**     - **m\_vwLikeGroup**     - **m\_vwJokeText**     - **m\_vwExpandButton** * Fill in the **setJoke(...)**method:   + Update your **m\_joke**reference with the joke was passed.   + Update **m\_vwJokeText** with the text for the new joke.   + Set the checked state to true on the appropriate RadioButton to reflect the rating for the new joke. If the joke is unrated then neither RadioButton should be checked.     - *Hint: You can use the RadioGroup to clear the checked state of all RadioButtons in the group.*   + Make sure to call **setJoke(...)**from the constructor. * Fill in the **collapseJokeView()** method:   + Set the **m\_vwJokeText** to ellipsize the text of the joke. If the joke is too long to fit in the TextView, the text should be truncated to fit, and an ellipsis should be appended to the end of the joke.     - *Hint: There is a single method call to handle this. See the Android Documentation on*[*TextView*](http://developer.android.com/reference/android/widget/TextView.html#setEllipsize%28android.text.TextUtils.TruncateAt%29)*.*   + Set the text on **m\_vwExpandButton**to display the **JokeView.EXPAND** string constant.   + Set the visibility on **m\_vwLikeGroup** so that it disappears, and does not take up any space in the layout.   + Make a call to **requestLayout()**.     - *By ellipsizing the joke TextView and making the rating RadioGroup disappear we have changed the size of the JokeView. This has caused the JokeView to become****invalidated****. Whenever a view becomes invalidated it should request to be laid out again.*     - *Failing to make this call will result in the view not being updated properly.*   + Make sure to call **collapseJokeView()**from the constructor. * Fill in the**expandJokeView()** method so that it performs the inverse functionality of **collapseView()**. * Setup the **m\_vwExpandButton** to respond to OnClick events:   + Make the JokeView class implement the OnClickListener interface.     - fill in the **onClick(...)** method so that if the JokeView is in its Expanded state, it calls collapseJokeView(). If the JokeView is in its CollapsedState, it calls expandJokeView().     - In the constructor, set the OnClickListener for **m\_vwExpandButton** to **this**JokeView object. * Setup the **m\_vwLikeGroup** to respond to OnCheckedChange events:   + Make the JokeView class implement the RadioGroup.OnCheckedChangeListener interface.     - You can read the details for this interface method in the Android Documentation for [RadioGroup.OnCheckedChangeListener](http://developer.android.com/reference/android/widget/RadioGroup.OnCheckedChangeListener.html).     - Fill in the **onCheckedChanged(...)** method so that when the state of the rating changes in the UI, the internal state of the joke is updated to properly reflect this change as well.   + In the constructor, set the OnCheckedChangeListener for **m\_vwLikeGroup** to **this**JokeView object.   **2.2.3 Make AdvancedJokeList use the JokeView class**  The last step is to update AdvancedJokeList to make use of your new JokeView custom component.   * Edit the **AdvancedJokeList.addJoke(...)**:   + Remove the code that initializes and adds a new TextView to **m\_vwJokeLayout**.   + Add code to initialize and add a new JokeView to **m\_vwJokeLayout**. * Run your Application to ensure that your changes to AdvancedJokeList and your new JokeView custom component are functioning properly.   + Your application should load with all of the jokes in the collapsed state.   + Test that your jokes expand properly by clicking one of the expand Buttons:   [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033594314/labs/lab-3/CustomJokeViewActivity.png?height=284&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/CustomJokeViewActivity.png?attredirects=0)  **3. Adapters & AdapterViews**  The purpose of this next section is to introduce you to the concept of AdapterViews. An AdapterView is a View class that allows us to bind it to a dataset. This binding then takes care of responding to user selections as well as populating the AdapterView with data. The binding is performed by a third intermediate class, called an Adapter. It is the Adapter that is responsible for keeping track of the selection and supplying the AdapterView with a View object representation of each item in the dataset. Read the Android Developer Guide on [Binding to Data with AdapterViews](http://developer.android.com/guide/topics/ui/binding.html) for a complete background on the topic.  In the context of this section, the AdapterView is a scrollable vertical ViewGroup called a ListView. The dataset is then our ArrayList of Joke objects. The Adapter class is the JokeListAdapter, which follows the standard ***Object*** ***Adapter Design Pattern****;* read the wiki on [Object Adapter](http://www.google.com/url?q=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FAdapter_pattern%23Object_pattern&sa=D&sntz=1&usg=AFrqEzdTc2Aqb-mG_RsJITpDMhblFhA6uQ) for more information. JokeListAdapter contains a reference to our list of Joke objects and supplies ListView with a JokeView for each them.  **3.1 Implement JokeListAdapter.java**  Begin by filling in the **constructor** and the stubbed **getSelection()** method:   * Selection should be initialized to the Adapter.NO\_SELECTION static constant. * *NOTE: The selection functionality won't be used until later on when you add the ability to "Remove" a joke.*   Make the JokeListAdapter class extend the BaseAdapter class. Check the Android Documentation on [BaseAdapter](http://developer.android.com/reference/android/widget/BaseAdapter.html) for details. You will have to add and implement the following abstract methods:   * **public int getCount()**   + Returns the number of items in the dataset. * **public Object getItem(int position)**   + Returns the Joke object from the dataset at the specified position. * **public long getItemId(int position)**   + If a Joke had a unique Id this would return it. However, you can use the Joke's position as its unique Id. * **public View getView(int position, View convertView, ViewGroup parent)**   + This method returns a JokeView object for the Joke object at the position in the dataset specified by **position**.   + The **convertView** object allows you to re-use a previously constructed view for better performance. Since **convertView**is a View object that was previously returned by **JokeListAdapter.getView(...)**, then you can safely assume it is JokeView.     - Check to see if this value is null.     - If it is null, then create a new JokeView object for the Joke at **position**.     - If it is not null, then change this JokeView to use the Joke at **position**.   + The **parent** parameter represents the container the returned JokeView will get added to. You won't need to use this, but in some cases it can provide useful information.   **3.2 Make AdvancedJokeList Activity Use ListView**  You will now make the AdvancedJokeList Activity class use the ListView and the JokeListAdapter classes to maintain your list of Jokes. You can read the Android Documentation on [ListView](http://developer.android.com/reference/android/widget/ListView.html) for details on the class.   * Update **advanced.xml** to use a ListView instead of a vertically oriented LinearLayout nested inside a ScrollView. The ListView will replace both the ScrollView and the LinearLayout.   + IMPORTANT: You must set the id attribute for the **ListView**element to be "**jokeListViewGroup**". * Update**AdvancedJokeList.java**:   + Change the type on **m\_vwJokeLayout** to **ListView** and update its initialization in **initLayout**.   + Initialize your **m\_jokeAdapter**member variable with your ArrayList of jokes.     - Do this in the**onCreate** method immediately after **m\_arrJokeList**has been initialized, but before you populate it with the joke's string resource values.   + Set **m\_vwJokeLayout's**adapter to be **m\_jokeAdapter**.     - Do this in the**onCreate** method immediately after **m\_jokeAdapter**has been initialized.   + Update the **addJoke(...)** method to notify **m\_jokeAdapter** that the dataset has changed.     - Make a call to JokeListAdapter's **notifyDataSetChanged()** method after adding the joke to **m\_arrJokeList**.     - If you don't make this call after changing the dataset, the ListView will not be updated to reflect the new state of your list of Jokes. You can read the Android Documentation on [BaseAdapter.notifyDataSetChanged()](http://developer.android.com/reference/android/widget/BaseAdapter.html#notifyDataSetChanged%28%29) for a complete description of the method.     - Remove the lines of code that explicitly initialize a new JokeView and add it to **m\_vwJokeLayout**. You don't need these anymore.     - Remove the lines of code that alternate the background colors. You don't need these anymore; ListView provides separators for the View objects it contains. * Run your application. AdvancedJokeList should function exactly as it did before. It should look slightly different. Instead of having alternating colors for rows, there should be line separators that automatically get added by ListView for you. The list should also scroll automatically. Here is an example with some sample jokes added and one of the JokeViews expanded:   [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033644175/labs/lab-3/AdvancedJokeList_Adapters.png?height=400&width=269](https://sites.google.com/site/androidcoursearchive/labs/lab-3/AdvancedJokeList_Adapters.png?attredirects=0)  **3.3 Enable ChoiceMode on ListView**  The ListView class provides an API for maintaining a selected, or chosen view. It does this in much the same way that a RadioGroup maintains a chosen RadioButton. When you select a RadioButton, the previously chosen RadioButton becomes un-chosen. ListView has the ability to do this as well for the view objects it contains.  In this section, you will enable the ChoiceMode on ListView, so that only one JokeView can be in the expanded state at a time. The JokeView that is in the expanded state will be the chosen View. When you click on another JokeView, the previously chosen JokeView will collapse, and the clicked JokeView will expand. In order for this to work, the Views that the ListView contain must implement the Checkable interface. Read the Documentation on the [Checkable](http://developer.android.com/reference/android/widget/Checkable.html) interface to understand how to implement it.  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033687799/labs/lab-3/ChoiceMode.png?height=167&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/ChoiceMode.png?attredirects=0)   * Begin by setting the choice mode on **m\_vwJokeLayout** after its initialization in **AdvancedJokeList**.   + Make sure to limit the number of chosen views allowed to just one. See the Android Documentation on [ListView.setChoiceMode()](http://developer.android.com/reference/android/widget/ListView.html#setChoiceMode%28int%29) for details. * Make JokeView implement the Checkable interface.   + When a JokeView is *checked*, it should be in the EXPANDED state. If the JokeView is *not checked*, it should be in the COLLAPSED state. * Read the following Blog on the Android Developer site explaining [Touch Mode](http://android-developers.blogspot.com/2008/12/touch-mode.html).   + In order for ListView to intercept the touch events and process your choice, it must not contain anything that receives focus.   + Update **joke\_view.xml** and set the following attribute on all of your UI Controls that normally receive focus:   android:focusable="false"   * + - Set this attribute on your expand/collapse Button and each of your like/dislike RadioButtons. * Run your application to make sure that only one JokeView can be in the expanded state at any given time and that your layout is unchanged.   + NOTE:     - You should be able to click anywhere on the JokeView to change it to the expanded state, not just the expand/collapse Button.     - The expand/collapse Button will no longer function as it does not receive focus anymore; this is the expected behavior.     - Consequently, once you choose a JokeView, at least one JokeView will always be in expanded mode. You can't clear a choice and set every JokeView back to the collapsed state.   **4 Menus**  This section is devoted to working with Menus. Read the Android Developer Guide on[Menus](http://developer.android.com/guide/topics/ui/menus.html) to get a good overview on the Android Menu system. In short, there are two different types of Menus. The first type of Menu is the Options Menu that can be brought up when the user hits the Menu button. Each Activity has the ability to declare its own Options Menu.  The second type of Menu is the Context Menu. Context Menus can be assigned to different Views and thus change depending on which View has focus. Hence the Menu depends on the *Context* in which it was triggered. Context Menus are brought up when the user holds the center D-pad button, clicks the trackball, or long-touches the screen for a few seconds when a particular View has focus.  In the two subsections that follow you will use both types of Menus. First, you will create a Context Menu for the ListView that will allow you to delete a JokeView. Then you will add an Options Menu to the AdvancedJokeList Activity that will allow users to filter the Jokes that are displayed by their rating.  **4.1 Adding a "Delete Joke" Context Menu**  When a user long-touches a JokeView, a Context Menu displaying a single MenuItem should be displayed. The MenuItem should simply display the text "Remove". When the Remove MenuItem is selected, the JokeView that was long-touched should be removed from the screen and from the list of Jokes.  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033733878/labs/lab-3/RemoveContextMenu.png?height=162&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/RemoveContextMenu.png?attredirects=0)  There are two ways to create Context Menus. The first is to have a particular View, like our ListView, be responsible for creating the Context Menu. The second and more common way is to have the Activity create the Context Menu, and then register the ListView to use the Context Menu. In this section, you will be taking the latter approach.   * Begin by overriding the following method in AdvancedJokeList:   public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)   * + This method only gets called the first time that a Context Menu gets brought up. Each subsequent opening of the ContextMenu only calls the onPrepareContextMenu(...) method. You can use this method to dynamically change how the Context Menu looks.   + Make sure to call back to the super class version of onCreateContextMenu so that the ContextMenu will be properly initialized.   + Initialize a new MenuItem to display "Remove" and add it to the ContextMenu.   + set the [OnMenuItemClickListener](http://developer.android.com/reference/android/view/MenuItem.OnMenuItemClickListener.html) of the MenuItem:     - Feel free to use either an anonymous inner class or make AdvancedJokeList implement the OnMenuItemClickListener interface.     - You should retrieve the selectionPosition from **m\_jokeAdapter**and remove that element from **m\_arrJokeList**.     - *Don't forget to notify****m\_jokeAdapter****that you have just changed the dataset.* * Next, register **m\_vwJokeLayout** to receive the context menu:   + Make a call to **registerForContextMenu(...)**   + This should be done in AdvancedJokeList.onCreate(...) after m\_vwJokeLayout has been initialized. * Make JokeListAdapter implement the [OnItemLongClickListener](http://developer.android.com/reference/android/widget/AdapterView.OnItemLongClickListener.html)interface:   + Use the interface method to set your **m\_nSelectedPosition**member variable. This will allow you to identify which item was long-clicked/touched.   + It is important to note that this event was generated from a LongClick event. The Activity.onCreateContextMenu(...) method is also listening to this event as well. If you consume this event, then it will not receive this event. If you haven't already, you should read the Android Developer Guide on[Event Listeners](http://developer.android.com/guide/topics/ui/ui-events.html#EventListeners), in particular, the section at the bottom that talks about consuming events.   + Make sure to correctly indicate whether this event was consumed. * Lastly, set **m\_jokeAdapter**to be **m\_vwJokeLayout's** OnItemLongClickListener in the AdvancedJokeList.onCreate(...) method.   **4.2 Adding a "Filter Jokes" Options Menu**  Your final task is to implement an Options Menu, that contains only a Submenu entitled "Filter". The Submenu will have four radio button options that allow the user to choose which Jokes to display. The radio button options are:   * **Like:**Displays only Jokes with a rating of Joke.LIKE. * **Dislike:** Displays only Jokes with a rating of Joke.DISLIKE. * **Unrated:**Displays only Jokes with a rating of Joke.UNRATED. * **Show All:**Displays all jokes.   On startup, the activity should show all jokes by default. To simplify things, the Filter functionality need only filter the jokes that are currently displayed. For example, let's say the user has selected the **Like** filter option so that only jokes with a rating of Joke.LIKE are displayed. Then the user adds a new Joke which will have a rating of Joke.UNRATED by default. This Joke will still be displayed. Similarly, if the user changed one of the Joke's ratings from Joke.LIKE to Joke.DISLIKE, this Joke will still be displayed. Thus, the filter does not need to monitor the addition of new Jokes, or changes in Joke ratings. (Click on the image below for an example story board Use Case)  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033796648/labs/lab-3/FilterMenuOption.png?height=400&width=270](https://sites.google.com/site/androidcoursearchive/labs/lab-3/FilterMenuOption.png?attredirects=0)  You are tasked to do this on your own  *Hints:*   * *I was able to do this by only adding code to AdvancedJokeList.java file.* * *You are free to add extra member variables if you would like, perhaps an extra list of Jokes...* * *Android Developer Guide on*[*Options Menus*](http://developer.android.com/guide/topics/ui/menus.html#options-menu)*,*[*Submenus*](http://developer.android.com/guide/topics/ui/menus.html#submenu)*,*[*Menu Groups*](http://developer.android.com/guide/topics/ui/menus.html#groups) * *Android Documentation on*[*Activity.onCreateOptionsMenu*](http://developer.android.com/reference/android/app/Activity.html#onCreateOptionsMenu%28android.view.Menu%29)*,*[*Activity.onOptionsItemSelected*](http://developer.android.com/reference/android/app/Activity.html#onOptionsItemSelected%28android.view.MenuItem%29)   **5. Establishing Http Connections**  Http connections are immensely useful for transferring data between a mobile device and a server. Such connections can be used by mobile devices for retrieving information as well as sending data back to a server. For the last section of the lab, you will be implementing functionality to share your jokes with the rest of the class. By making Http Requests, you will connect with a server to upload your own jokes and download other people's jokes.  The Android framework doesn't supply its own API for establishing connections to the internet. Instead, it includes both of the java.net and org.apache.http packages. The java.net.URL class provides a very clean and simple interface for creating an Http connection and retrieving a response. The apache.http package provides a more robust collection of classes and is not as straightforward to use.  **5.1 Uploading Jokes to the Server**  You will begin by adding an "Upload Joke" MenuItem to the Context Menu that gets displayed when a user long-presses/long-touches a joke. When the user selects the "Upload Joke" MenuItem, your application should send the text of the joke and the name of the Author to a server. The server will then send back a response indicating whether the joke was recieved. Your application should then notify the user that the upload succeeded or failed via a *Toast Notification*.  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033849162/labs/lab-3/UploadJoke.png?height=188&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/UploadJoke.png?attredirects=0)  **NOTE**: The method comment block for uploadJokeToServer() and getJokesFromServer() in AdvancedJokeList.java say the methods will show a progress dialog for the duration of the method. **You do not have to do this**. Ignore this comment.  **5.1.1 Update Your Manifest**  Accessing the Internet requires your application to have permission to use the internet. Your application must declare in its manifest that it uses the internet. When the application gets installed, the user is presented with a list of permissions that the application says it will use. At this point, the user will have the option to grant the application these permissions, or deny these permissions and cancel the installation.  Add the following line to your AndroidManifest.xml file:  <uses-permission android:name="android.permission.INTERNET"></uses-permission>  It should be nested between the root **<manifest></manifest>** tags, one level deep, at the same level as your **<application></application>**tags. It should look something like this:  <manifest ... >      ...      <application ...>          ...      </application>      ...      <uses-permission android:name="android.permission.INTERNET">      </uses-permission>      ...  </manifest>  **5.1.2 Add "Upload Joke" Context MenuItem**  In your **onCreateContextMenu(...)** method:   * Add a new MenuItem to the context menu, Initializing its text to "Upload Joke to Server". * Set the MenuItem's OnClickListener:   + The OnClickListener should call the **uploadJokeToServer(...)** method, passing in the Joke that was long-pressed.   + Feel free to use either an anonymous inner class or make AdvancedJokeList implement the OnMenuItemClickListener interface.   **5.1.3 Fill in "uploadJokeToServer(Joke joke)" Method**  You will have to test for and catch any Exceptions that are thrown by using classes from the java.net package. You don't have to do anything special should an exception be thrown, just exit the method gracefully. Feel free to print out whatever exception information you feel may be of use.  Begin by constructing a string that will contain the complete URL you will use to submit your Joke to the server:   * The base URL to the addJoke script is: **http://simexusa.com/aac/addJoke.php?** * Append to this a **UTF-8** encoded parameter named **joke**, containing the text of your Joke.   + Use the static method **java.net.URLEncoder.encode(...)**to perform the encoding.     - This method takes in a string containing the text it should encode, and a string containing the type of encoding it should use.     - It will then return a "UTF-8" encoded string containing the text you passed in. For instance, this method will replace spaces with "+" characters.   + Use **"UTF-8"** as the encoding string. * Append to this another **UTF-8** encoded parameter named **author**, containing the Author's name.   + Separate this parameter from the previous one with an **"&"** character. * Your URL should look something like this when finished:   **http://simexusa.com/aac/addJoke.php?joke=***This+is+my+joke***&author=***reed*  Make your Http connection and read the response from the server:   * Create a new java.net.URL Object by passing in your complete URL string into the constructor.   + This will establish the connection to the server and submit your joke.   + An example of using the URL class can be found [here...](http://www.google.com/url?q=http%3A%2F%2Fwww.exampledepot.com%2Fegs%2Fjava.net%2FReadFromURL.html&sa=D&sntz=1&usg=AFrqEzfHcTryZFNMYhfL7FirGueM8yfqfA) * call **openStream()** on your URL Object to read the response from the server.   + This will return an InputStream from which you will have to parse the server response.   + Read the entire response from the server into a string object. Do this however you like. As a suggestion, the java.util.Scanner class can be constructed from an InputStream object and provides a nice interface for parsing.   Test the response and notify the user of success or failure:   * Compare the response from the server:   + If you succeeded the server will send back a message with the text of **"1 record added"**   + Any other response is considered a failure. * Create an android.widget.Toast notification:   + If your upload was successful, display the text **"Upload Succeeded!"**   + If your upload failed, display the text **"Upload Failed!"**   + Do this by using the static **Toast.makeText(...)** method.     - Read the Android Developer Guide on [Creating Toast Notifications](http://developer.android.com/guide/topics/ui/notifiers/toasts.html) for more information.     - Use Toast.LENGTH\_SHORT as the duration.   + Call the **show()**method on the returned Toast object to display the notification.   Run your application and ensure that you can successfully upload jokes to the server.  **5.2 Downloading Jokes from the Server**  Your final task is to add a "Download Jokes" Options MenuItem. When the MenuItem is clicked your application will download all the jokes from a server and display them in the list of jokes. While the application is downloading the jokes and adding them to the joke list, it will display a progress indicator dialog box.  [https://sites.google.com/site/androidcoursearchive/_/rsrc/1264033883092/labs/lab-3/DownLoadJokes.png?height=187&width=400](https://sites.google.com/site/androidcoursearchive/labs/lab-3/DownLoadJokes.png?attredirects=0)  **5.2.1 Add "Download Jokes" Options MenuItem**  In your **onCreateOptionsMenu(...)** method:   * Add a new MenuItem to the context menu, initializing its text to "Download Jokes". * Set the MenuItem's OnClickListener:   + The OnClickListener should call the **getJokesFromServer()** method.   + Feel free to use either an anonymous inner class or make AdvancedJokeList implement the OnMenuItemClickListener interface.   **5.2.2 Fill in "getJokesFromServer()" Method**  Construct a new URL object:   * The URL string to download the jokes is  **"http://simexusa.com/aac/getJokes.php"** * This URL takes an optional **UTF-8** encoded parameter named **author**, containing the Author's name.   + When this parameter is omitted, all the jokes on the server will be retrieved. This could be a lot of jokes by the time you perform this exercise.   + By supplying the author parameter you will only download jokes whose author matches the value supplied.   Parse the response:   * The jokes returned to you by the server come in the form of a single string with each joke separated by a new-line character, '\n'. * Parse the jokes text from the InputStream   + I would recommend the Scanner class. In particular the useDelimeter function is quite helpful. * Add the jokes to your joke list and display them.   Run your application and ensure that you can successfully download jokes from the server.  **6. Deliverables**  To complete this lab you will be required to:   1. Provide a demonstration of your functional application to Dr. Janzen or the Lab Assistant during Lab Hours no later than the due date. Make sure to plan for this accordingly.  Come ready to demonstrate at the beginning of lab on the lab due date at the latest and you should be fine. 2. Put your entire project directory into a .zip or .tar file, similar to the stub you were given. Submit the archive to the Digital Dropbox on Blackboard (don't forget to both "Add" and "Send"). This effectively provides time-stamped evidence that you submitted the lab on time should there be any discrepancy later on in the quarter. The name of your archive should be **lab3<cal-poly-username>.zip|tar**. So if your username is **jsmith** and you created a zip file, then your file would be named **lab3jsmith.zip**. 3. Complete the following survey for Lab 3: <http://www.surveymonkey.com/s/5QNDC32>   Primary Author: James Reed Adviser: [Dr. David Janzen](http://www.google.com/url?q=http%3A%2F%2Fwww.csc.calpoly.edu%2F%7Edjanzen&sa=D&sntz=1&usg=AFrqEzc6vv4qEokQc8b1hJMCHSneIGc7Gw) |