0

Java Essentials for Android Course

Labs >

Lab 3: Joke List 2.0

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. It is also important to note that this lab builds on concepts started in Lab 2; finishing Lab 2 first is highly recommended even though you are given a skeleton to work with in this lab as well.

Objectives

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

- *How to declare layouts statically as an XML resource (briefly covered in Lab 1).*
- *How to create custom Views using existing components and show them in a scrollable ListView.*
- *How to add custom icons to a component and apply State Lists.*
- *How to use Adapters and AdapterViews to bind a front-end View class to its corresponding back-end data.*

- *How to create Toast Notifications.*
- *How to add backwards-compatible Android menus and nested menus using ActionBarSherlock (Action Bar).*
- *How to add Contextual Menus for individual Views (Contextual Action Bar).*

Activities

For this lab we will be extending the "Joke List" application that you created in Lab 2. This version of the app will be more advanced. It will allow the user to give ratings to Jokes, delete Jokes and filter existing jokes by rating. All tasks for this lab will be based off of this application. Over the course of the lab you will iteratively refine and add functionality to the Joke List app. With each iteration you will be either improving upon the previous iteration's functionality, implementing the same functionality in a different way or adding new functionality. You will be using code from Lab 2, so make sure to complete Lab 2 first.

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 tests 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 at any time to ensure that your application is functioning properly.

Contents

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 to download the skeleton project, then make sure to extract the project and 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, choose *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 lab3<userid> where **<userid>** is your user id (e.g. jsmith).
- *Note: If you receive a message in the Console in Eclipse about requiring specific Android compiler compliance, perform one of the following:*
    - *Right-click on the project in the Project Explorer and choose **Android Tools** > **Fix Project Properties**.*
    - *Right-click on the project in the Project Explorer and choose **Properties**, then under the **Java Compiler** tab change **Compiler compliance level** to 1.5 or 1.6.*

Finally, make sure that your project is targeting the latest version of Android but supporting API 10 and higher.

- In the Manifest file, find the **uses-sdk** XML component and change **android:targetSdkVersion** to the latest API version (17 at the time of writing this lab). If **android:minSdkVersion** is not set to 10, do so.

You may run the application right now to make sure it installs and appears on your development device of choice, although the app will have nothing in it yet.

1.2 Fill in the Joke Class

You may fill in the Joke class using the functionality that you implemented for this class in Lab 2. However, there are several key differences. In particular:

- A member variable named **m_strAuthorName** has been added to the class which will contain the name of the Joke's author.
- You must update the code in the constructors. You are required to pass in an Author name for all the Constructors except for the default constructor. Set the author name in **res/values/strings.xml. Retrieve this author name in onCreate()**, and use this author name when creating a Joke object.
- 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 one getter and setter method apiece that must be filled in.

In Lab 2, we covered how to create an Android Test Project specifically for running tests on a single application. In this lab and the remainder of the labs, you will just run Unit Tests from within the same project. Currently this technique works, but it goes against the testing fundamentals provided by Google. For the sake of speeding up development, the Android Unit Tests can be run from the tests folder in the skeleton project you downloaded.

Run the **JokeTest.java** Unit Tests in the **test** folder to ensure that you have properly filled in this class before continuing to the next step. For more information on how to run them, please see this section in lab 2.

You will run more tests later to guarantee proper functionality up to certain points as you progress further in this lab.

Note: There are several test files that are excluded from the build path and thus appear differently in the Project Explorer. You will add them back to the build path when the time is right.

1.3 Fill in the AdvancedJokeList Class

You will first fill in the AdvancedJokeList class using the code that you implemented for SimpleJokelist in Lab 2:

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.
    - You may have to add a new class member variable **m_nTextColor** to account for the variable you used in Lab 2 for the joke text color. You'll have to update **colors.xml** to include this new color as before, and initialize all three color variables in **onCreate()**.
    - If you have a Logcat debug statement in the lab 2 code, you can modify it for lab 3 or remove it entirely, up to you.
    - Keep in mind that the size of the text needs to be 16 raw pixels (**COMPLEX_UNIT_PX**) otherwise one of the tests you will run soon will fail. You can also modify the test (in **AdvancedJokeListAcceptanceTest.java**) to check for a larger text size if 16 raw pixels is too small for your device.
    - Remember to set the joke author variable before adding any Jokes. Do this in **onCreate()**.

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 rest of 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. You will also probably need to add an entry into your resources for the white joke text color (R.color.text).

Fill in the **initLayout()** method using the code from **SimpleJokeList.initLayout()**. It should be exactly the same as the code from Lab 2.

Run the **AdvancedJokeListAcceptanceTest.java** Unit Tests to ensure that you have properly filled in this class, then run your application on the emulator or a physical Android device to ensure that it performs the way it did in Lab 2.

- If you fail a unit test due to text size but wish to have your application display larger text, change the **TEXT_SIZE** variable in **AdvancedJokeListAcceptanceTest.java** to match your desired text size. This will set up the other unit test classes to adjust to this text size automatically as well.

Note: *You may have noticed that changing the orientation of the Android device screen when running the applications you've programmed thus far causes new components and information to disappear. We will fix this in the next lab!*

2. Declaring Static Layouts in XML

Read the Android Developer Guide on Declaring Layout 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 set up dynamically in SimpleJokeList in **initLayout()** to a static one in 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:

- **advanced.xml** already contains a LinearLayout as the root ViewGroup to prevent compilation errors. Feel free to use this as your root ViewGroup for the main application layout.
    - **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:
        - **Button m_vwJokeButton**: use **addJokeButton** as the resource id.
            - For the text attribute, instead of putting the raw String "Add Joke", follow good Android practice and reference a String resource. Add a String value "Add Joke" to **/res/values/strings.xml** and reference it in this XML layout file for the button's text.

        - **EditText m_vwJokeEditText**: use **newJokeEditText** as the resource id.
            - Add a hint to the text field (with **android:hint**) that says "Enter a joke". Instead of putting the raw String "Enter a joke", follow good Android practice and reference a String resource. Add a String resource with the value "Enter a joke" to **/res/values/strings.xml** and reference it in this XML layout file for the text field's hint.
        - **LinearLayout m_vwJokeLayout**: use **jokeListViewGroup** as the resource id.
            - *This id seems type-inconsistent, but it will become obvious later in the lab.*
    - *Hint: Be sure that your layout_height is wrap_content for the LinearLayout that contains the Button and EditText. Otherwise, you might never see your list of jokes that follows.*

Edit your **initLayout()** method to use the **advanced.xml** layout file:

- Remove all the code from this method. It should now be an empty method.

- You must make your call to **setContentView()** be the first thing that you do. This is called "inflating" the layout. Layout inflation happens when Android takes a static layout XML file and interprets it in a special way, turning the raw XML into part of the application's actual UI. Remember to pass in the resource for **advanced.xml** from **R.java**.

    - *Hint: You did this in Lab 1, it requires the type and name of the resource as well as the resource id itself.*

    - **Note:** You won't be able to retrieve references to your UI Controls until the layout has been inflated from the XML file. That's why the call to **setContentView()** is being done first!

- Initialize your view class member variables by retrieving references to them, instead of constructing new ones:

- **m_vwJokeLayout**
- **m_vwJokeEditText**
- **m_vwJokeButton**
- *Hint: You did this in Lab1, remember the **findViewById** method?*

Try running the **AdvancedJokeListPreTest.java** Unit Tests. They should all pass.

Try running your application. The UI should appear and function exactly as it did for SimpleJokeList in Lab 2.

2.2 Building Custom UI Components

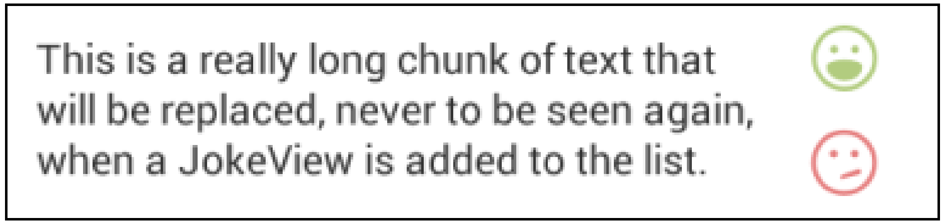
Sometimes the standard View library will not supply the functionality that you need. In situations like this it is completely acceptable to define your own UI Components. There are three general approaches to creating custom UI components:

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. We can do better than displaying a TextView with a background color; it lacks aesthetics and is limited to only showing text. We want to create a graphical representation of a Joke that looks clean and functions appropriately for the application. We shall call this representation a **JokeView**.

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.

The custom component that you are going to implement will look like this (minus the black outline):



This is a really long chunk of text that will be replaced, never to be seen again, when a JokeView is added to the list.

This is how a single Joke will show up visually on-screen.

This is what a single **JokeView** will look like in visual form. Each JokeView will be composed of a **TextView** and a **RadioGroup** with two **RadioButton** children. The RadioButtons will show custom icons instead of the default button icons to better convey the intention (rating; do you like or dislike the joke?). The icons will toggle between looking more colored or discolored to indicate which rating is selected. The custom icons are provided in the lab stub under the ***res/drawable-*dpi*** folders. You will first create the State List selectors to enable use of multiple custom icons on each RadioButton component.

Note: this lab is supportive of mdpi, hdpi and xhdpi screen densities. For more information on supporting multiple screens, see this page.

2.2.1 Create State Lists for Like and Dislike

State Lists act as state machines: they provide logic for the appearance of a set of images (one at a time) on a single component. We will use them to make each RadioButton display a fully colored emote when it is selected, and a more faded emote when not selected. In other words, selecting the green emote in the above JokeView image will cause it to become fully colored and the red emote to become faded, and selecting the red emote will cause it to become fully colored and the green emote to become faded. To do this, we will use the **selector** XML object type.

Define the **res/drawable/like.xml** State List file:

- Right-click the drawable folder in the Package Explorer and choose **New** > **Other...** so the wizard pops up. Beneath the Android folder, choose **Android XML file** and then click **Next**.

- Make sure the Resource Type is set to **Drawable**, choose **selector** as the Root Element and call the file **like.xml**. Click **Finish**.

- Add two items to the selector: one that uses the **ic_action_emo_laugh** drawable resource if the object is checked, and a second one that uses **ic_action_emo_laugh_deselected** if the object is not checked.

    - Hint: The item properties documentation for selectors can be found here.

    - *Note: The order of the items in a State List is important. Android uses the last selector item as the default drawable resource shown if none of the other items' conditions are met. In this lab the order doesn't matter too much, just keep this in mind for future usage.*

Define the **res/drawable/dislike.xml** State List file:

- Repeat the above steps, but use the **ic_action_emo_err** and **ic_action_emo_err_deselected** resources instead.

2.2.2 Declare a Custom JokeView XML Layout

The next step is to create the XML layout file that the custom component will use.

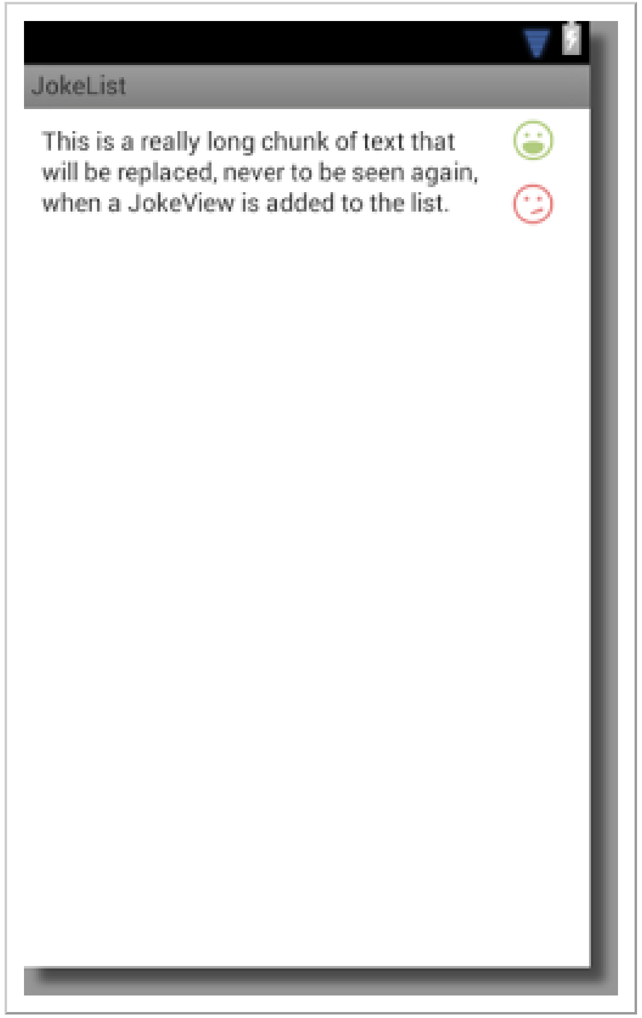
Fill in the **res/layouts/joke_view.xml** layout file:

- **jokeview.xml** has been stubbed out (very minimally) for you already. It contains a LinearLayout as the root ViewGroup to prevent compilation errors.

- **IMPORTANT:** You must use the following id values for this list of UI Components defined in AdvancedJokeList.java:

- **RadioButton m_vwLikeButton**: use **likeButton** as the resource id.
- **RadioButton m_vwDislikeButton**: use **dislikeButton** as the resource id.
- **RadioGroup m_vwLikeGroup**: use **ratingRadioGroup** as the resource id.
- **TextView m_vwJokeText**: use **jokeTextView** as the resource id.

This is roughly what you want to see as the final result for **joke_view.xml** in the Graphical Layout tab (click image for full view):



This is a really long chunk of text that will be replaced, never to be seen again, when a JokeView is added to the list.

By default we want no rating, so it makes sense that both rating buttons are faded to start.

Hints:

- You can test your UI without having to run your application by switching back and forth between the LayoutEditor tab (**Graphical Layout**) and the XML Editor tab (**joke_view.xml**).

    - Make changes in the textual XML Editor tab. The Graphical Layout tab will render your UI.

    - Test your layout by setting the text of your TextView in the XML Editor to a long string, to see how all components appear.
        - The buttons should appear centered relative to the size the TextView containing the joke string. You may wish to also provide space between the TextView and RadioGroup.

- Try using **android:weightsum** and **android:layout_weight** properties to designate varying screen space for each top-level component. Information on LinearLayout and Weight can be found here.
    - For all sakes and purposes, RadioGroup is a LinearLayout meant to contain RadioButtons.

- To get the RadioButton components to hide the default button icons and show yours, you must edit the following properties:
    - **android:button** - Set this to **@null**. This will remove the default button icon set completely.
    - **android:background** - Set this to the respective State List drawable resource.

- If TextView keeps cutting the bottom off of really long text, try adding padding to the top and bottom.

- To position the RadioButtons with precision, try adding layout margins.
    - What's the difference between padding and margins? This question is addressed here.

2.2.3 Create the JokeView data class

The next step is to implement your custom component class. This class will be called (unsurprisingly) **JokeView**. 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 (likely LinearLayout). 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.

- *It currently extends the View base class, you will have to change this to the class your root ViewGroup is. (i.e. what is the root ViewGroup in joke_view.xml, this is the class you should extend in JokeView.java).*

Fill in the **JokeView(Context context, Joke joke)** constructor:

- 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**
        - *Note: Ignore the background color of each JokeView for now, this is because the next section will address differentiation between JokeViews in a different way.*

Fill in the **setJoke(...)** method:

- Update your **m_joke** reference with the joke that was passed in.
- 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.

- 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.*

Set up 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.

    - 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. This means changing the rating of the Joke object inside the JokeView.

- In the constructor, set the OnCheckedChangeListener for **m_vwLikeGroup** to **this** JokeView object.

- *Note that without the OnCheckedChangeListener, joke ratings would be lost when doing other things on your Android device such as summoning and dismissing the onscreen keyboard.*

2.2.4 Make AdvancedJokeList use the JokeView class

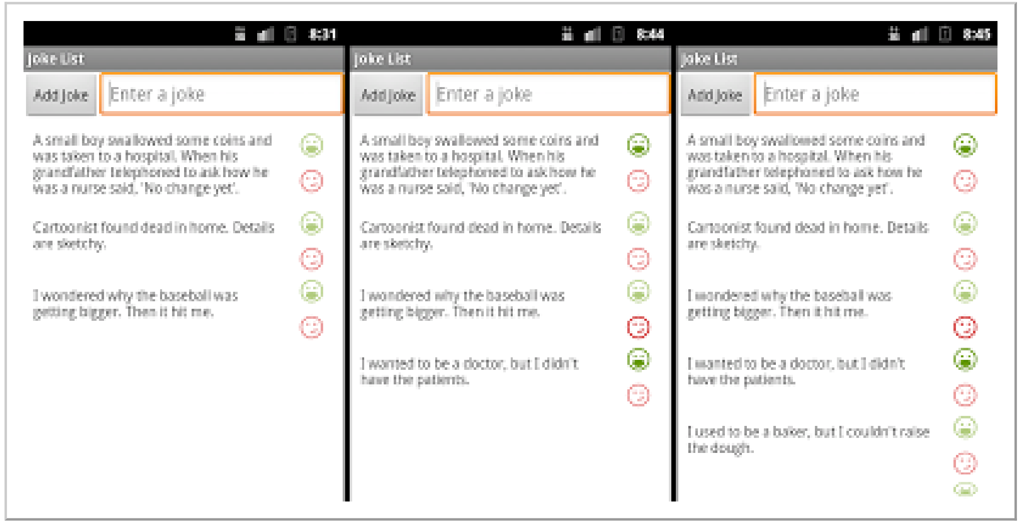
The last step is to update AdvancedJokeList to make use of your new JokeView custom component. Now we can finally see those jokes in the list! Or at least, we hope so.

Edit the **AdvancedJokeList.addJoke(...)** method:

- Remove the code that initializes and creates, customizes and adds a new TextView to **m_vwJokeLayout**.
- Add code to initialize and add a new JokeView to **m_vwJokeLayout**.

Run the **AdvancedJokeListTest.java** Unit Tests. They should all pass. If a unit test fails due to JokeView's joke text size, change the size in the TextView component in **joke_view.xml** to best fit your viewing needs (you removed the dynamic text size change when you removed the code to create the TextView in **addJoke(...)**).

If you run your application now, you should see something like the following where you can change each joke's rating by touching the smiley radio button icons (click image for full size):



A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

I wanted to be a doctor, but I didn't have the patients.

A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

I wanted to be a doctor, but I didn't have the patients.

I used to be a baker, but I couldn't raise the dough.

Adding several new jokes should work.

Note: *Sometimes running an application multiple times from Eclipse will merely cause the application to be brought to the center of attention on your Android device (in Eclipse, you may have seen the text* ***ActivityManager: Warning: Activity not started, its current task has been brought to the front*** *in the Console window). If you want to repeatedly test a fresh "just launched" version of an Android application while making little to no changes to the code base, you may wish to demolish the old version of it and create a brand new version of it each time you run it. To do this, open a terminal window (Command Line or PowerShell in Windows, Terminal in Mac/Linux) and type* ***adb uninstall <packagename>*** *before every new run. For this lab, this would be* ***adb uninstall edu.calpoly.android.lab3****.*

The application functions reasonably well, but right now there are a few noticeable problems. There is no clear distinction between Jokes; with the previous version, each joke would alternate background colors making them easy to separate. Furthermore, LinearLayout does not provide scrolling functionality (in lab 2 you placed it inside of a ScrollView to enable scrolling), causing JokeViews that would go off-screen to squish the icons. However, we will be replacing the LinearLayout with a much cleaner ViewGroup in the next section that solves the separation and scrolling issues--**ListView**.

Additionally, you will notice that we have two separate calls in **addJoke()**: One that adds a joke to **m_arrJokeList**, and another that adds a View to **m_vwJokeLayout**. Since we are now introducing additional data (i.e. actions we are able to perform on each Joke, namely applying a rating), it would be nice if we could easily apply changes in **m_arrJokeList** to the views in **m_vwJokeLayout**, and vice-versa. Right now if a user of this application were to change the Rating of one Joke object, the change would not carry over to the ArrayList of Joke objects. We are missing the 'glue' that binds them together, the Adapter (**m_jokeAdapter**), and we will tackle this next.

3. Adapters & AdapterViews

The purpose of this section is to introduce you to the concept of AdapterViews. An **AdapterView** is a View class that allows us to bind one or more Views 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 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 for more information). JokeListAdapter contains a reference to our list of Joke objects and supplies ListView with a JokeView for each of them.

3.1 Implement JokeListAdapter.java

Begin by filling in the **constructor**:

- Set **m_context** and **m_jokelist** appropriately.

- Make the JokeListAdapter class extend the **BaseAdapter** class. Check the Android Documentation on BaseAdapter for details. You will have to add and implement the following abstract methods:

  public int getCount()

- Returns the number of items in the dataset (**m_jokeList**).

public Object getItem(int position)

- Returns the Joke object from the dataset at the specified position.

public long getItemId(int position)

- Usually returns the id of the item at the position in the list. However, you can use the Joke's position as its unique Id for this lab.

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 use JokeListAdapter and ListView

You will now make the AdvancedJokeList Activity class use the **JokeListAdapter** and **ListView** classes to maintain your list of Jokes, thus adding the 'glue' to keep the visual and data halves binded.

You can read the Android Documentation on ListView for details on the class. LinearLayout is an excellent ViewGroup for displaying components in a row or column fashion, but lacks functionality otherwise compared to other Views like ListView. In a nutshell, ListView offers functionality beyond what LinearLayout provides, including built-in scrolling and child View management options (in particular, it allows control over child view selection behavior).

Update **advanced.xml** to use ListView instead of a LinearLayout.

- IMPORTANT: You must keep the id attribute for the **ListView** element as **jokeListViewGroup**.

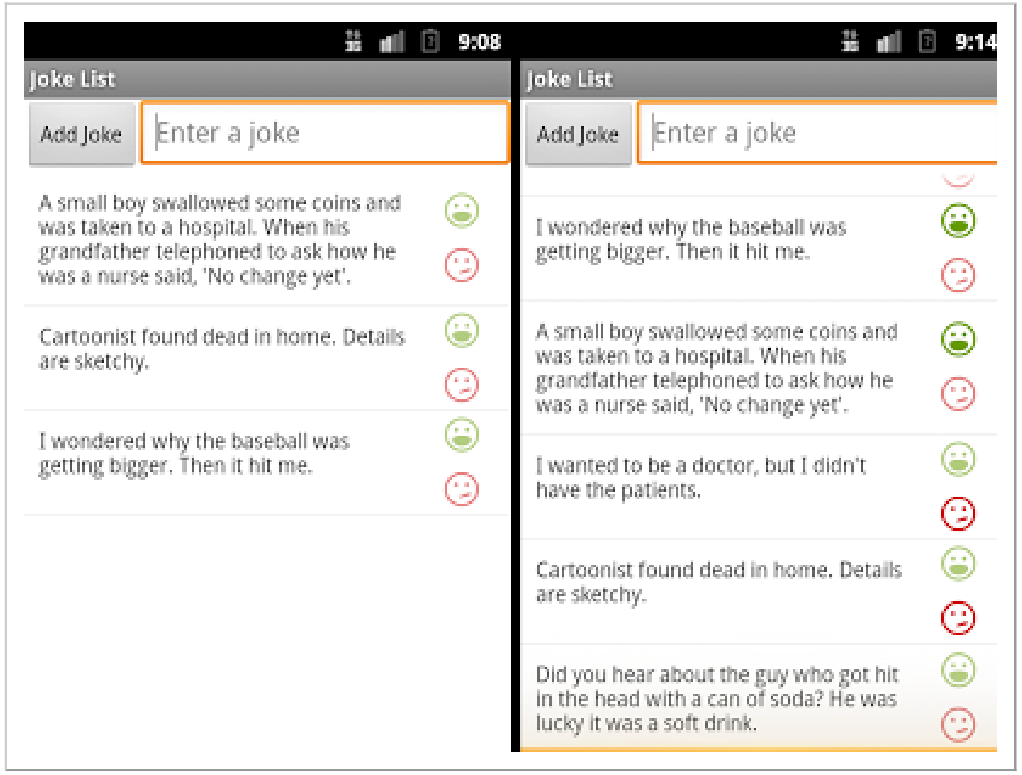
Update **AdvancedJokeList.java**:

- 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 **initLayout()** method immediately after **m_vwJokeLayout** 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**. This is the single method call that will allow changes in the Joke list to affect the physical JokeViews in the app.

        - 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() for a complete description of the method.

    - Remove the lines of code that explicitly initialize a new JokeView and adds it to **m_vwJokeLayout**. You don't need these anymore since the Adapter now handles it for you.

Run your application. AdvancedJokeList should function exactly as it did before, but it should look slightly different. Instead of having no row indication, there should be line separators that automatically get added by ListView for you. The list should also scroll automatically. Here is an example with the default jokes added, then a few sample jokes added (click image for full size):



A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

I wondered why the baseball was getting bigger. Then it hit me.

A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

I wanted to be a doctor, but I didn't have the patients.

Cartoonist found dead in home. Details are sketchy.

Did you hear about the guy who got hit in the head with a can of soda? He was lucky it was a soft drink.

Scrolling down or up reveals all jokes.

Add the **AdvancedJokeListTest2.java** file in the **test** folder back to the Build Path (Right-click on file > **Build Path** > **Include**) and run its Unit Tests. Make sure they all pass before proceeding to the final section. Note that the previous tests, **AdvancedJokeListTest.java**, will fail to run successfully since you changed the structure of the components.

4. Menus

This section is devoted to working with Menus. Read the Android Developer Guide on Menus to get a good overview on the Android Menu system. There are two different types of menus you will work with: the Action Bar (which replaces the Options Menu) and Context Menus (which replace floating Context Menus with Contextual Action Mode). These will be described in greater detail as you prepare to implement them.

More on backwards-compatibility

Due to the evolution of the Android API and Android physical devices (e.g. tablets), there have been numerous changes made to menus. Starting with Honeycomb (API 11, Android 3.0), devices are no longer required to have a physical Menu button. Instead, devices running Honeycomb or higher use the Action Bar to display menu options using a combination of on-screen actions and overflow actions.

You will notice that the minimum supported API version in this lab is 10. Why concern ourselves with API 10 (Gingerbread, Android 2.3)? At the time of writing this lab, Android device usage information indicates that a significantly large percentage of Android users (~90%) have devices that run Gingerbread or higher, with ~40% running Gingerbread itself. Gingerbread devices (largely if not entirely made up of smart phone devices) were created with a physical Options menu button and thus did not need the Action Bar. Since we are developing for a minimum of API 10, we must consider ***backwards-compatibility***: The ActionBar class is not available in API 10 and lower, but we wish to create an application that looks the same across multiple versions of Android.

How to implement backwards-compatible menus

Fortunately, there are several ways to provide an Action Bar and its functionality for older versions of Android that do not normally contain it. The first is to use ActionBarCompat, a solution provided in Google's Android support library. More information, including reasons to consider ActionBarCompat are available in this article.

The second is ActionBarSherlock (ABS), an extension of the Google compatibility solution and provides an Action Bar and its features across all Android versions. We will use ABS due to its ease-of-use and its flexibility in providing a custom implementation of the Action Bar if the native version of Android does not provide it.

In the two subsections that follow you will use both types of Menus. First, you will instantiate the Action Bar and add a menu item enabling joke filtering. Then you will create a Contextual Menu for removing jokes from the current list of jokes being displayed.

4.1 The Action Bar

4.1.1 Download and Set Up ActionBarSherlock

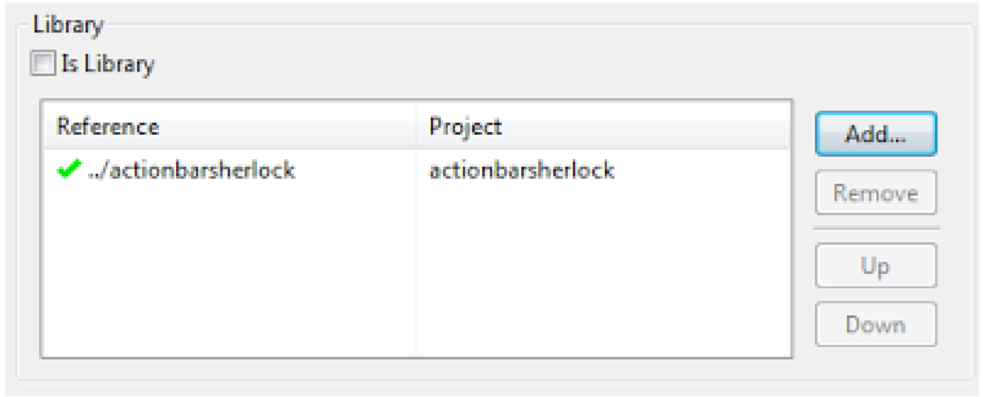
You will now download and set up ABS as an Eclipse library project, which will be referenced in your Eclipse Android application project.

- Download ABS here and extract its contents. You will likely want to download the ZIP project, but depending on your operating system you may wish to download a different version.
- Follow the instructions on the ABS Usage page to create the library project in Eclipse using the extracted **library** folder. Hint: Use Import an existing Android project to add the actionbarsherlock project in Eclipse.
    - ABS requires a support JAR file called **android-support-v4.jar**. Information about how to obtain and include this support JAR in your project can be found here. Make sure to copy it into your project's **libs** folder.

    - If there is a JAR mismatch error, you can go to the android SDK folder on your computer, navigate to the **android-support-v4.jar** file (typically located in **extras/android/support/v4**) and copy the file into both directories indicated by the error.

- Reference the newly-created ABS library project in your own project.

    - Right-click on your project in the Package Explorer, then select **Properties...** and click on the **Android** tab. At the very bottom of the displayed options is a place to include Library targets. Add the ABS project and click **OK**.



Reference	Project
✓ ../actionbarsherlock	actionbarsherlock

You may consider renaming the ABS library project to something more intuitive such as 'actionbarsherlock'.

- Make AdvancedJokeList.java extend **SherlockActivity** instead of Android's **Activity** class. You will need to import **com.actionbarsherlock.app.SherlockActivity**.

- Add an appropriate ABS theme to your project's Manifest file (**Theme.Sherlock**, **Theme.Sherlock.Light**, **Theme.Sherlock.Light.DarkActionBar** or a child of any of these), since the Action Bar's inclusion is dependent on themes introduced in ICS and higher.

    - In AndroidManifest.xml, change the application's theme from AppTheme to one of the above themes (make sure to still precede it with **@style/**). We won't create any themes for this lab so you will use one of the already existing ABS ones.

- You may *possibly* encounter a huge slew of errors in the Console upon finishing the above step, claiming multiple cases of Error retrieving parent for item. This is caused by an improper API target, as ABS requires at least Android 4.0 or higher to function. To fix these issues, perform the following:

    - Return to the Android Properties tab and switch the Project Build Target to the latest version of Android (17 at the time of writing this lab). Make sure it's not the Google APIs version (select the one that has **Android** in its name, not **Google APIs**). Remember that you already changed the targeted version and minimum supported version in the Manifest file.

Run the application and make sure that it still looks and functions similarly to before. If the background is completely black (likely due to choosing **Theme.Sherlock**), try **Theme.Sherlock.Light** instead.

4.1.2 Adding the Action Bar and a Menu Item to AdvancedJokeList

Now ABS is ready to be used. ActionBarSherlock uses the same steps to create the Action Bar as is detailed in the Android Menus walkthrough, only using different imports.

First, create the menu XML file:

- Use the Eclipse Android XML file wizard to create a new XML Menu file in

res/menu called **mainmenu.xml**.

- Add an item with id **menu_filter** that uses a String resource with the value "Filter" as its title (it should already exist in **strings.xml**).
    - Set **android:showAsAction** to make the menu item show up only if there is room in the menu, and to provide the title text for this item in the Action Bar.
        - See the XML menu item documentation for more details.
        - *What this will do is prevent the Action Bar from trying to fit too many menu items inside itself; if there is no room in the Action Bar for a menu item, it will be placed either in the overflow menu (Android 3.0 and higher) or in the options menu displayed when pressing the physical Menu button (Android 2.3 and lower). See Android's page on Backwards Compatibility for more details.*

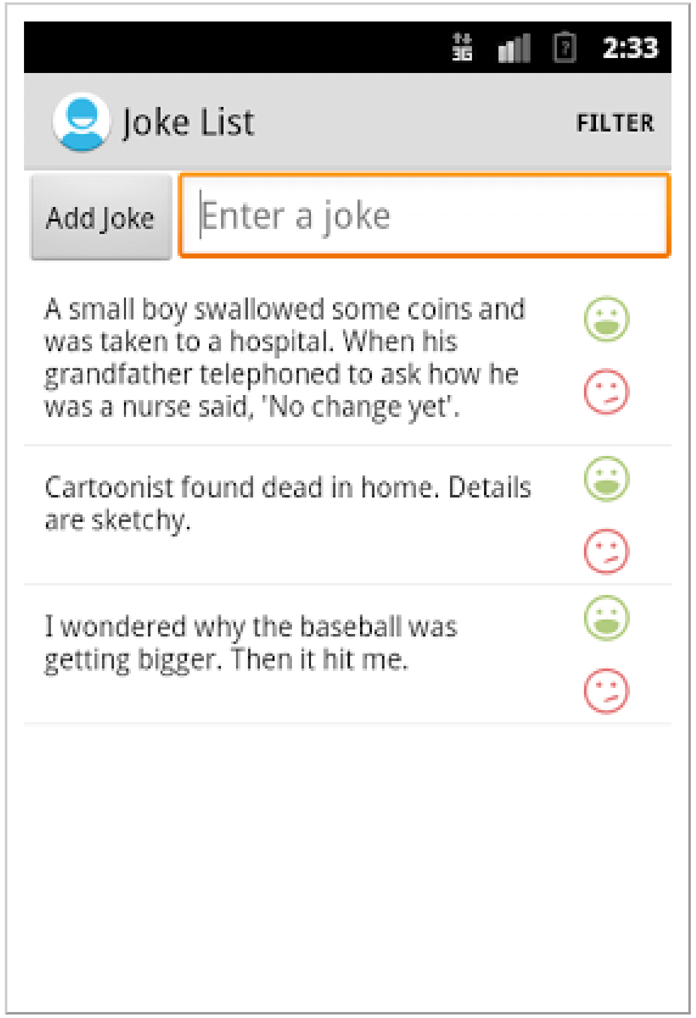
Edit **AdvancedJokeList.java** to use the menu resource:

- Fill in the **onCreateOptionsMenu()** method using a **MenuInflater**.
    - You will need to make a call to **getSupportMenuInflater()** to use ABS's compatibility package instead of Android's functionality. To do this, import **com.actionbarsherlock.view.MenuInflater**.
        - **IMPORTANT:** Make sure that you are using the ActionBarSherlock packages for Menu, MenuItem and MenuInflater. Remove any native Android imports such as **android.view.Menu** and replace them with

the ABS versions, otherwise there will be compile errors due to clashing imports.

- Initialize **m_vwMenu** after inflating the menu. This is essential for the upcoming Unit Tests.
- Documentation for filling in this method can be found here.

If you run your application now, you should see an Action Bar across the top of the application (click image for full size):


2:33

Joke List FILTER

Add Joke | Enter a joke

A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

Action Bar Sherlock works on all Android versions, even those completely lacking an Action Bar. Above is a Nexus One (API 10) emulator screenshot.

If you click or press the Filter Action Bar menu item, nothing will happen. You will now be briefly introduced to an Android feature that lets you provide minimal visual behavior feedback before adding the full menu functionality.

4.1.3 Toast Notifications

In the previous lab you were introduced to LogCat, which allows you to simulate debug printing on an Android device. Adding debug **Log.d()** statements allows for quick textual feedback in the LogCat tab in Eclipse and lets you double-check whether or not expected behavior or proper method invocation is happening without changing up your code base.

You will now use a second tool for that same kind of feedback on your Android device, only visual: Toast notifications. Toasts create a condensed text popup that appears briefly on screen before disappearing. You can create and modify Toast variables, but creating and displaying a toast all at once is as simple as injecting one line at the desired area:

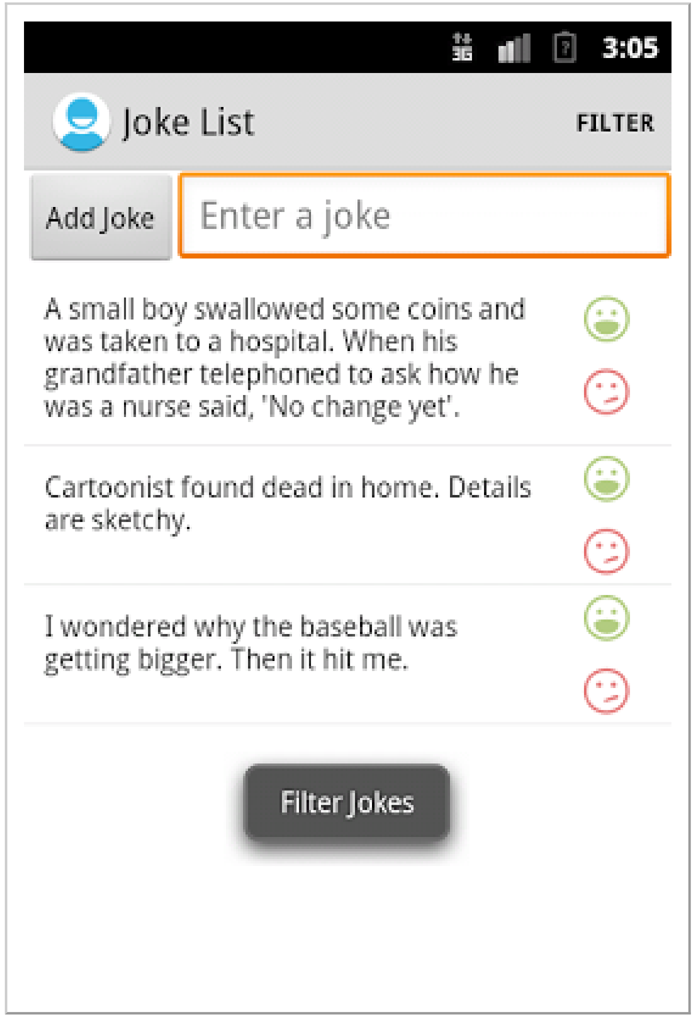
```
Toast.makeText(context, text, duration).show();
```

context - Almost always **this**, or context obtained via **getContext()**, etc.
text - A String to display on the screen.
duration - How long the Toast is displayed on the screen. One of **Toast.LENGTH_LONG**, **TOAST.LENGTH_SHORT**.

Use a Toast notification to show minimal menu feedback as a sanity check to make sure the menu item responds correctly. Make AdvancedJokeList respond to selected menu items:

- Override the **onOptionsItemSelected()** method.
  - Documentation for filling in this method can be found here.
  - Inside of the switch case for Filter, create and display a Toast notification. Feel free to experiment with the duration parameter to see roughly how long each duration is.

Run the application. Clicking or pressing on Filter will show a Toast notification on the screen (click image to view full size):



A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

This proves that the Filter menu item responds as intended.

We now see minimal visual feedback response from the menu. Now to replace the Toast notification with the actual menu functionality. You may or may not find Toasts useful later for further minimal testing similar to what you just did.

4.1.4 Enable Joke Filtering

You will now add a nested menu to the Filter menu item in the Action Bar. This will allow for JokeView sorting in the ListView based on the current rating of all jokes.

There will be four options: sorting by Like, Dislike, Unrated and showing all Jokes. Choosing ***Like*** will cause only jokes with the rating **Like** to show, ***Dislike*** will cause only jokes with the rating **Dislike** to show, **Unrated** will cause only ***Unrated*** jokes (no RadioButton chosen) to show, and ***Show All*** will show all jokes regardless of rating.

Edit **mainmenu.xml**:

- Add four nested menu items to the Filter menu item. Do this by adding a nested menu component inside the Filter item component.
    - Use the following **ids** and **titles** respectively for each item:
        - **Like** - **submenu_like** and **like_menuitem**
        - **Dislike** - **submenu_dislike** and **dislike_menuitem**
        - **Unrated** - **submenu_unrated** and **unrated_menuitem**
        - **Show All** - **submenu_show_all** and **show_all_menuitem**

If you run the application now, you will see a dropdown menu on the Filter menu item. However, they currently have no functionality. To give them functionality, add them to the **onOptionsItemSelected()** method.

- Remove the Toast display when pressing Filter if you haven't already, as it is now intrusive and unwanted.

Now you will add the Filter functionality in **AdvancedJokeList.java** yourself. Hints follow:

- You are free to add new class member variables and methods to handle this functionality.
- You will need to use a second "filtered" ArrayList containing the list of Jokes and bind this to the Adapter instead. There is already a variable there for you: **m_arrFilteredJokeList**. Rebind the adapter to use this list instead of

m_arrJokeList.

  - The filtered ArrayList will be in charge of containing the appropriate jokes to display based on which filter is selected, and the base ArrayList (**m_arrJokeList**, the one you've been using so far) will keep track of the entire list of Jokes.

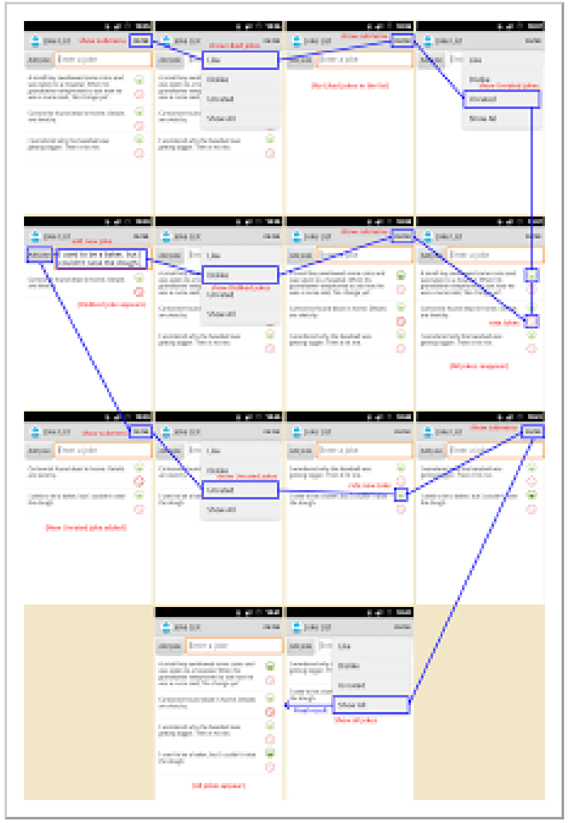
  - Whenever a filter option is chosen, you must take care of two issues:
    - Changes in the currently displayed filtered list of Jokes that have not been synced with the base list of Jokes
    - Populating the filtered list of Jokes with only the Jokes that match the appropriate filter

    Carefully consider all possible use cases. For example, what if the user starts the app up with the default three unrated Jokes in the list, rates the first Joke with "Like", chooses the "Like" Filter (causing the one Liked Joke to appear by itself in the list), changes the rating of that joke to "Dislike" and chooses the Dislike filter? Just that same Joke should appear again.

- Don't forget to call the adapter's **notifyDataSetChanged()** method after changing the filtered list's contents, otherwise changes will not appear in the application.

- There is an issue where the list of JokeViews may seemingly reverse at random, or show the wrong rating even though the data behind the JokeView is correct. This is likely caused by lazy View reuse in ListView--View objects are cached but their positions are not. To resolve this, explicitly set the JokeView properties in JokeListAdapter.java's **getView()** method if **convertView** is being reused. See a documented example of this inversion behavior here.

Below is an example use case 'storyboard' (click to show full size) that shows what happens when the following is done: Load a fresh instance of the app -> Filter by Like -> Filter by Unrated -> Rate two Jokes -> Filter by Dislike -> Add new Joke -> Filter by Unrated -> Rate new Joke -> Filter by Show All. Notice that if you add a new joke while the list of jokes is being filtered, the new joke will show up in the filter, even though it is unrated, until you filter jokes again. This is the desired functionality.


You will want to view this in full size. Trust us.

Include the **AdvancedJokeListAddFilterTest.java** Unit Test file in the build path, then run the tests. They should all pass.

Feel free to test your app further to make sure it works as expected before moving on to the next and final section.

4.2 Contextual Action Mode

Now that filtering is enabled, your final task is to implement a deletion feature that will enable removal of a selected Joke in the list. This removal should persist across both the filtered and unfiltered lists of Jokes as well.

It is not intuitive to add this feature as an Action Bar menu item because it is not clear which Joke is being selected. This would require implementing a way to display which JokeView in the list is currently selected, which is extra work. Instead of reinventing the wheel, you will create a contextual menu that appears when certain components (in this case, JokeViews) are "selected" by the user, with a single option to remove the selected Joke from the list.

There are two ways to provide Contextual Menus: Floating Context Menus and Contextual Action Mode (CAM). As you could probably guess from the title of this subsection, you will be working with the latter. In the above link you will not only find a great comparison image between the two Contextual Menu implementations, but you will also notice Google's warning that CAM is only available on Android 3.0 and higher. However, like typical adventurous developers would, we are going to ignore Google's warning and make it work on older devices anyway! With some help from ActionBarSherlock, that is.

4.2.1 Create Menu Resource

First you will create the static menu XML resource. This will be similar to the menu creation for the Action Bar menu items.

Add **actionmenu.xml** to **res/menu**:

- Add a single menu item with id **menu_remove** and title **remove_menuitem**.

4.2.2 Enable Contextual Action Mode

You will enable Contextual Action Mode for individual views. Ideally we would want batch contextual actions; this would allow for selecting multiple JokeViews inside the ListView and marking them for removal, but this is only enabled on API 11 and higher

(CHOICE_MODE_MULTIPLE_MODAL only works above API 10) and cannot be enabled through ActionBarSherlock.

You will now use Action Mode to enable a selection menu to appear whenever a JokeView is long-clicked. Note that long-click has been chosen over a normal click to allow for accidental missed rating button presses on each JokeView, giving the user room for error in touching the small rating buttons.

Implement ActionMode.Callback:

- Add two variables of type **com.actionbarsherlock.view.ActionMode** and **com.actionbarsherlock.view.ActionMode.Callback** to the class. It is essential that you use the ActionBarSherlock versions, otherwise the application will not function properly.

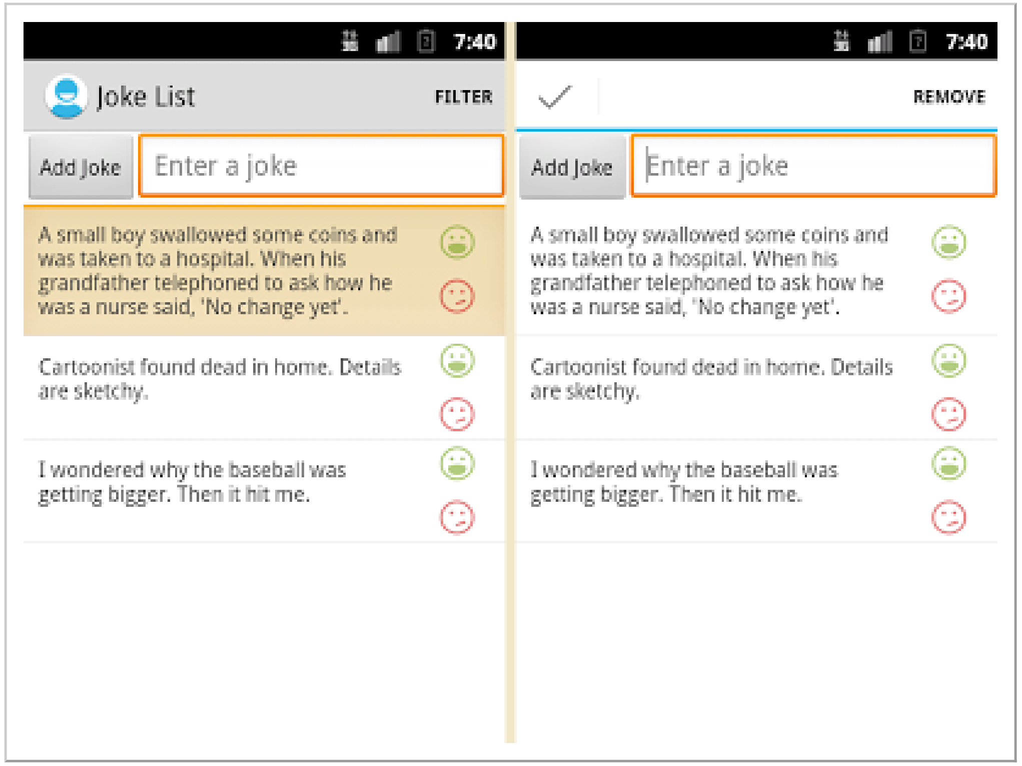
- Follow the instructions for CAB for individual views here to inflate the Action Menu you just created and set the Action Mode to properly terminate after the Remove item is selected. You will fill in the functionality for actually removing the joke in the next step. You will probably want to create the Callback as an anonymous inner class in initLayout(), and use it to start the contextual action mode in the onItemLongClick() method described next.

- Continuing with the instructions as a guide, set your ListView to have an **OnItemLongClickListener** to trigger the firing of the Action Mode Callback.

    - **getActivity()** is not available. See if you can figure out which method to call in **onItemLongClick()** to fetch the Activity instead. Hint: Remember that you are in a SherlockActivity.

    - To get each JokeView to respond to long-clicks, you will need to prevent JokeView from obtaining focus. This will allow JokeView to relinquish focus to the ListView that contains it, letting ListView handle focus events instead. Read more about handling focus here. Hint: consider using android:descendantFocusability to your root ViewGroup.

Run your application and gaze in wonder at the Contextual Action Menu that appears when long-clicking on a JokeView (click image for full size):



A small boy swallowed some coins and was taken to a hospital. When his grandfather telephoned to ask how he was a nurse said, 'No change yet'.

Cartoonist found dead in home. Details are sketchy.

I wondered why the baseball was getting bigger. Then it hit me.

You may want to add a temporary Toast to make sure that the Remove option responds when selected.

Clicking the checkmark in the left corner of the Contextual Action Menu will dismiss it and bring the Action Bar back into focus.

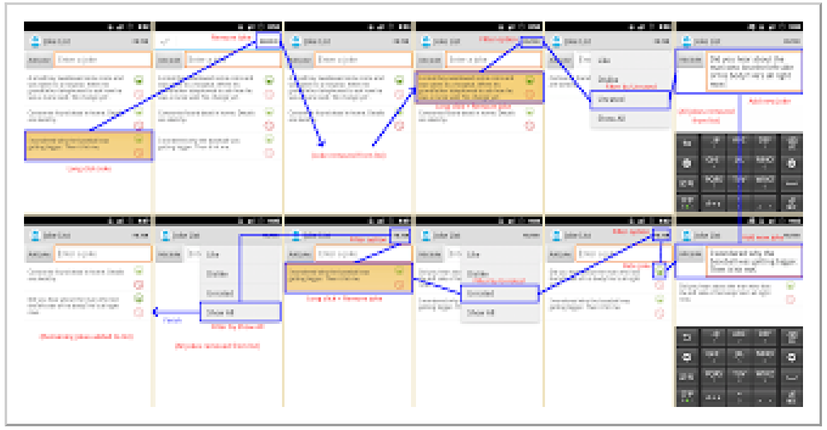
4.2.3 Implement Joke Deletion

The last step in this lab is to add the functionality for joke removal. You are tasked to do this on your own. Hints follow:

- When a joke is long-clicked, you must keep track of its position in the filtered list since ListView does not automatically handle child View selection status. This will allow you to remove the appropriate joke from your lists.

- You may make good *inherent* use of Joke.java's **equals()** method for deletion without even having to call it once. You've probably already made good use of it for filtering and changing ratings.

Run your application and make sure that both filtering and deletion work as anticipated. This is a sample use case storyboard showing several deletions, filters and insertions (click image for full view):



Contextual Menu only shown once for brevity's sake.

You didn't think the fun was over just yet, did you? Include **AdvancedJokeListAddFilterDeleteTest.java** in the build path and run those tests. They should all pass. Your project should also still pass the **AdvancedJokeListAddFilterTest.java** unit tests. If so, congratulations, you've completed this lab!

5. Deliverables

To complete this lab you will be required to:

1. Put your entire project directory into a .zip or .tar file, similar to the stub you were given. Submit the archive to PolyLearn. 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<userid>** with a **.zip** or **.tar** extension. So if your username is jsmithand you created a zip file, then your file would be named **lab3jsmith.zip**.
2. Load your app on a mobile device and bring it to class on the due date to demo for full credit.
3. Complete the survey for Lab3**:** https://www.surveymonkey.com/s/436F13Lab3

Primary Authors: James Reed and Kennedy Owen
Adviser: Dr. David Janzen

Comments

Commenting disabled due to a network error. Please reload the page.

You do not have permission to add comments.