**121`**

ITMD 455/555 *Intelligent Device Applications* Lab 3

#### Famous Quotes App- 50 points

**Introduction**. Alright enough of the hot/cold weather apps! This lab concentrates on the fundamentals of using Intents and multiple activities. You will create a ListView full of quotes, which will serve as the master view, then when the user clicks on a particular quote they will be taken to a detail view of the full quote!

Also included in this package is a Splash screen for grad credit that will be added!

Sample intial view of running app...

**A screenshot of a cell phone

Description automatically generated**

**STEP 1 Creating a New Android Project**

Create a new **Empty Views Activity** project with the Application Name called **Quotes**. Use **API 28 (“Pie”…)**, as your minimum SDK going through the configuration wizard.

Once your app builds, right click on MainActivity and choose **Refactor > Rename ... .** Rename your activity as **QuoteReaderActivity** and keep any defaults. Press Refactor when complete.Also in the same manner refactor your activity\_main.xml file in your **res/layout** folder to **activity\_quote\_reader** as your Layout Name. Press Refactor when complete.

**STEP 2 Crafting your activity for your project. Your first activity screen will contain a list with quotes and thumbnails. Lets start by creating the UI for this.**

Open up your **activity\_quote\_reader.xml** file and include the following code:

<ListView xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:id=*"@+id/quotes\_list"*

android:layout\_width=*"fill\_parent"*

android:layout\_height=*"fill\_parent"*>

</ListView>

This will create a list view with a width and height equal to the size of the screen.

Delete your defaulted TextView tag included in the file. Ignore any *constraint* warning.

**STEP 3 Add in rows to your list.**

In this step you will create rows and add them to your list. We know that a Listview row

needs to have a thumbnail and text.

Here is how the first row is going to look:

Graphical user interface, text, application

Description automatically generated

I know it’s Steve Jobs, but who knows without him maybe there would be no Android!?

Let’s start by creating the layout for the row. Create a new xml file in the **res/layout** directory. To add an XML, right click on your **layout** folder and choose   
**New > XML > Layout XML File**. Name your layout “**list\_item\_layout**”. Use a **RelativeLayout** as your Layout “Root Tag” option. Next, open it in the raw XML view and replace the contents starting at the top line, with the following object elements:

*<?***xml version="1.0" encoding="utf-8"***?>*<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent" android:layout\_height="100dip"**>  
 <**ImageView  
 android:id="@+id/thumb"  
 android:layout\_width="60dip"  
 android:layout\_height="60dip"  
 android:layout\_centerVertical="true"  
 android:layout\_alignParentLeft="true"**/>  
 <**TextView  
 android:id="@+id/text"  
 android:layout\_toRightOf="@+id/thumb"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:textSize="18dip"  
 android:layout\_marginLeft="10dip"  
 android:layout\_centerVertical="true"  
 android:singleLine="true"  
 android:ellipsize="end"  
 android:textStyle="bold"**/>  
</**RelativeLayout**>

Notice this creates an ImageView for the thumbnail plus a TextView for the quote. Notice too that the parent view used in this layout is a RelativeLayout. This layout allows you to position its children 'relative' to each other and itself. The ImageView needs to be placed to the left of the layout, and it needs to be centered vertically, hence those properties are set

to true. The TextView also needs to contain text in a single line and if the length increases it

should *ellipsize*, therefore the singleLine and ellipsize properties are set.

Okay now that we have the UI in place, lets add some code so that we can use it for each list item.

**STEP 4 Adding in quotes and a data source for your adapter.**

For this step you will add in a class to serve as your adapter to help paint your

layout on your screen.

First off, open up your **res/values/strings.xml** file and add in the following quotes to your syntax:

<string name=*"quote\_1"*>Innovation distinguished between a leader and a follower</string>

<string name=*"quote\_2"*>I want to put a ding in the universe!</string>

<string name=*"quote\_3"*>People don\'t know what they want until you show it to them.</string>

<string name=*"quote\_4"*>The only way to be truly satisfied is to do what you believe is great work.</string>

<string name=*"quote\_5"*>That\'s been one of my mantras -- focus and simplicity. Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But it\'s worth it in the end because once you get there, you can move mountains.</string>

<string name=*"quote\_6"*>Great things in business are never done by one person, they\'re done by a team of people.</string>

<string name=*"quote\_7"*>You can\'t connect the dots looking forward; you can only connect them looking backward.</string>

<string name=*"quote\_8"*>Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work.</string>

<string name=*"quote\_9"*>I\'m the only person I know that\'s lost a quarter of a billion dollars in one year. ... It\'s very character-building.</string>

<string name=*"quote\_10"*>It\'s more fun to be a pirate than join the Navy.</string>

Notice you have to precede any single quotes ( ‘ ) with a backslash ( \ ). You’ll be replacing

some quotes soon with your own, so not to worry Android lovers!

Next download the zip file I have provided in Canvas accompaning this lab entitled ‘**steve.zip**’ that will serve as some pics to now add (copy) into your drawable folder. Note the pics just serve as an example for now, as you can delete some of them later and add in your own! Snapshots that follow will guide you thru copy and paste flow.

A blue and white text

Description automatically generated

Choose **drawable** folder to copy to.

At the pop-up dialog box, press OK again to finish the Copy directive.

Graphical user interface, text, application

Description automatically generated

Now that you have all your data, create your data source class. Right click on your package ( ex.**com.example.quotes**),locateddirectly under your **java folder** and choose **New** then select **Java Class**. Name it **DataSource**, and press Enter to commit your entry.

Add the following lines of code within the class:

**private** ArrayList<Integer> mPhotoPool;

**private** ArrayList<Integer> mQuotePool;

**private** ArrayList<Integer> mPhotoHdPool;

**public** ArrayList<Integer> getmPhotoHdPool() {

**return** mPhotoHdPool;

}

**public** ArrayList<Integer> getmPhotoPool() {

**return** mPhotoPool;

}

**public** ArrayList<Integer> getmQuotePool() {

**return** mQuotePool;

}

**public** DataSource() {

mPhotoPool = **new** ArrayList();

mQuotePool = **new** ArrayList();

mPhotoHdPool = **new** ArrayList();

}

All that we have done here is create 3 variables, assigned their getter methods, and initialized the variables in the constructor.

You’ll notice you get warnings that it can’t find ArrayList with the supplied code, so to recitfy the situation simply by adding in the necessary import namely

**import** java.util.ArrayList;

towards the top of the file, and ALWAYS under your package name!

Next populate the array lists with data which are the pictures plus the quotes. Notice that we have defined the array list to contain integers (ex. *steve\_1,* *quote\_1,…)*. The reason for this is, when you drop in any resource into the resource directory the build system creates a unique identifier which is represented by an integer.

Okay now create 3 separate functions to populate the ArrayLists. Add the following functions above the constructor definition within your DataSource class.

**private** **void** setupPhotoPool() {

mPhotoPool.add(R.drawable.*steve\_1*);

mPhotoPool.add(R.drawable.*steve\_2*);

mPhotoPool.add(R.drawable.*steve\_3*);

mPhotoPool.add(R.drawable.*steve\_4*);

mPhotoPool.add(R.drawable.*steve\_5*);

mPhotoPool.add(R.drawable.*steve\_6*);

mPhotoPool.add(R.drawable.*steve\_7*);

mPhotoPool.add(R.drawable.*steve\_8*);

mPhotoPool.add(R.drawable.*steve\_9*);

mPhotoPool.add(R.drawable.*steve\_10*);

}

**private** **void** setupQuotePool() {

mQuotePool.add(R.string.*quote\_1*);

mQuotePool.add(R.string.*quote\_2*);

mQuotePool.add(R.string.*quote\_3*);

mQuotePool.add(R.string.*quote\_4*);

mQuotePool.add(R.string.*quote\_5*);

mQuotePool.add(R.string.*quote\_6*);

mQuotePool.add(R.string.*quote\_7*);

mQuotePool.add(R.string.*quote\_8*);

mQuotePool.add(R.string.*quote\_9*);

mQuotePool.add(R.string.*quote\_10*);

}

**private** **void** setupPhotoHDPool() {

mPhotoHdPool.add(R.drawable.*steve\_hd\_1*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_2*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_3*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_4*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_5*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_6*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_7*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_8*);

mPhotoHdPool.add(R.drawable.*steve\_hd\_9*);

mPhotoHdPool.add(R.drawable.*apple\_hd*);

}

Each of these functions code work is to fill in data into the ArrayLists. Call these functions   
 within your constructor at the bottom, by adding in the following three lines:

setupPhotoPool();

setupQuotePool();

setupPhotoHDPool();

Next add another function for the size of the data source. In our case the size of the   
 data source is 10 i.e. we have 10 images and 10 quotes hence we can simply return the size of any one of the ArrayLists.

Add the following method above your constructor:

**public** **int** getDataSourceLength() {

**return** mPhotoPool.size();

}

Do a Save All at this point. Next you need to use this data source to help populate your list.

**STEP 4 Creating your custom adapter.**

Open QuoteReaderActivity.java from the java folder and add in the following inner class named **QuoteAdapter** (code shown in **bold** below) directly within your QuoteReaderActivity class. This class should inherit from the BaseAdapter class and it should therefore implement all the abstract methods with minimal implementations.

Here’s what the class should look like now (no need for any other extraneous code):

public class QuoteReaderActivity extends AppCompatActivity {

**public class QuoteAdapter extends BaseAdapter {**

**@Override**

**public int getCount() {**

**return 0;**

**}**

**@Override**

**public Object getItem(int position) { return null; }**

**@Override**

**public long getItemId(int position) { return 0; }**

**@Override**

**public View getView(int position, View convertView,   
 ViewGroup parent) {**

**return null;**

**}**

**}**

/\*\* Called when the activity is first created. \*/

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_quote\_reader);

}

}

**Clean up your imports for your activity file to resemble the following at this   
point:**

**import** androidx.appcompat.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import android.view.View;  
import android.view.ViewGroup;  
import android.widget.BaseAdapter;**

Here what each method of the inner class does.

* **getCount()**: Returns the number of items present in the data set.
* **getItem(int position)**: Gets the data item associated with the specified position in the data set.
* **getItemId(int position)**: Gets the row id associated with the specified position.
* **getView(int position, View convertView, ViewGroup parent**): This is used to get a view that displays the data at the specified position in the data set.

Now that you have a basic understanding of all the functions, add a constructor to your adapter and some private variables. Add the following lines of code to the **QuoteAdapter** inner class:

**private** Context mContext;

**private** LayoutInflater mInflator;

**private** DataSource mDataSource;

**public** QuoteAdapter(Context c) {

mContext = c;

mInflator = (LayoutInflater)  
 mContext.getSystemService(Context.*LAYOUT\_INFLATER\_SERVICE*);

mDataSource = **new** DataSource();

}

Add any imports where you see highlighted text in red as a flag. Click on any red part that is flagged and just do Alt+ Enter to allow for the automatic addition of a needed import statement. In case you still need import statements, the following should definitely be included.

**import** android.view.LayoutInflater;  
**import** android.content.Context;

The new mInflator variable (shown above) will be used here to instantiate the list\_item\_layout “inflated” XML file into *corresponding* View objects, as will be shown below in the code.

Next change all your overridden functions EXACTLY as follows:

@Override

**public** **int** getCount() {

**return** mDataSource.getDataSourceLength();

}

@Override

**public** Object getItem(**int** position) { **return** position; }

@Override

**public** **long** getItemId(**int** position) { **return** position; }

@Override

**public** View getView(**int** position, View convertView, ViewGroup

parent)

{

ImageView thumbnail;

TextView quote;

**if**(convertView == **null**) {

convertView = mInflator.inflate(R.layout.*list\_item\_layout*,

parent,**false**);

}

thumbnail = convertView.findViewById(R.id.*thumb*); thumbnail.setImageResource(mDataSource.getmPhotoPool()

.get(position));

quote = convertView.findViewById(R.id.*text*);

quote.setText(mDataSource.getmQuotePool().get(position));

**return** convertView;

}

Again add in import statements where you’re red flagged. The main function here is the getView function, since it is called for every item in the list view. It is responsible for binding the data source with the list view item. In it we first check if the **convertView** passed into the function is **null**. The reason we do this is to check if the convertView returned is not a recycled view. The adapter does not create a new view for each and every item. Instead, it creates a set number of views and reuses them as the users scrolls through the list.

Once this is done, we get a reference to our image view and text view and fill it with data from our data source. The **position** variable returned is used as an index in your data source. Note needed imports for above code fill in case you still need them are

**import** android.widget.ImageView;  
**import** android.widget.TextView;

**STEP 5 Finalizing your Master View.**

You now have our adapter and data source set, so you just need to tell your list view to start using your adapter. Add the following ListView declarations in your **onCreate** method within your **QuoteReaderActivity** class (an import will be needed).

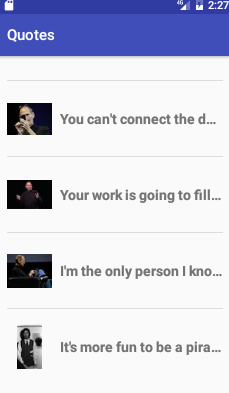
ListView mListView = findViewById(R.id.*quotes\_list*);

mListView.setAdapter(**new** QuoteAdapter(**this**));

Needed import will be the following (good idea to clean up any unused imports first)

**import** android.widget.ListView;

Now run the application with an acceptable emulator and you should see a screen (Master view) like the following:

****

**Sweeet! Make sure to scroll up/down to see all quotes!**

**STEP 6 Setting up your detail screen.**

Next extend your app features so the user can tap a quote to see it in full detail thusly showing a larger size picture and the full quote written at the bottom!

Creating a new UI. Create a layout for your detail view screen by creating a new xml file in your **res/layout** directory and name it “**quote\_detail**”. Use ScrollView as your “Root Tag” Layout option. Add the following lines ( shown in **bold** ) to the file:

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<ScrollView   
 xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

**<RelativeLayout**

**android:layout\_width=*"wrap\_content"***

**android:layout\_height=*"wrap\_content"*>**

**<ImageView**

**android:id=*"@+id/image"***

**android:layout\_alignParentTop=*"true"***

**android:layout\_width=*"fill\_parent"***

**android:layout\_height=*"350dip"***

**android:layout\_marginLeft=*"5dip"***

**android:src=*"@drawable/steve\_1"* />**

**<TextView**

**android:id=*"@+id/quote"***

**android:layout\_width=*"fill\_parent"***

**android:layout\_height=*"wrap\_content"***

**android:layout\_below=*"@+id/image"***

**android:textSize=*"24sp"* />**

**</RelativeLayout>**

</ScrollView>

Here notice that the root view is a scroll view and it has only one child, which is a relative layout. This relative layout contains your image view for the picture and text view for the quote. The reason why a scroll view is used is that since the quote could be long, there is a possibility that it might go off the screen. A scroll view is smart enough to understand when its child view is bigger than the screen size and will automatically add in a scroll bar for the user to scroll down!

**STEP 7 Add in the QuoteDetail activity.**

Lets add the third and last activity that is responsible for presenting the

quote\_detail.xml layout details to the user.

Create a new **class** and name it **QuoteDetail**, make *sure* that it **extends**   
 **AppCompatActivity**.

Add in the following code and the needed imports to your file:

**import** androidx.appcompat.app.AppCompatActivity;

**import** android.os.Bundle;

Add in the following **onCreate** method as a needed override to allow for a layout to render.

**public** **class** QuoteDetail **extends** AppCompatActivity{

**@Override**

**protected void onCreate(Bundle savedInstanceState) {**

**super.onCreate(savedInstanceState);**

**setContentView(R.layout.quote\_detail);**

**}**

}

**STEP 8 Connecting your two activities (like a Master/Detail view).**

An activity in Android is instantiated with an Intent. An Intent is used to start an activity.

In your **QuoteReaderActivity.java** file add the following at the end of your onCreate  
 function. Make sure to included needed imports, especially like for your intent!

mListView.setOnItemClickListener(**new** AdapterView.OnItemClickListener() {  
  
 @Override  
 **public void** onItemClick(AdapterView arg0, View arg1, **int**

position,**long** arg3) {  
 Intent i = **new** Intent(QuoteReaderActivity.**this**,

QuoteDetail.**class**);  
 i.putExtra(**"position"**, position);  
 startActivity(i);  
 }  
});

What we are doing here is setting an item click listener and every time an item is clicked we create an Intent! Notice that we pass the position variable to the intent, this is used to

represent the position in the data source for the item clicked.

Next add the following lines (appearing in **boldface**) in **QuoteDetail.java**:

**private ImageView mImageView;**

**private TextView mQuote;**

**private int mPosition;**

**private DataSource mDataSource;**

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.*quote\_detail*);

**Intent i = getIntent();**

**mPosition = i.getIntExtra("position", 0);**

**mDataSource = new DataSource();**

**mImageView = findViewById(R.id.*image*);**

**mQuote = findViewById(R.id.*quote*);**

**mImageView.setImageResource(mDataSource.getmPhotoHdPool().**

**get(mPosition));**

**mQuote.setText(getResources().getString(mDataSource.getmQuotePool()**

**.get(mPosition)));**

}

Include needed imports at this point and then save all your files at this point to update your package. As you can see, we retrieve the position sent by the QuoteReaderActivity from the intent and use that to fill in the image view and the text view.

A last thing you need to do before you can run the application is to add an entry for this activity explicitly in your manifest as a rule of thumb whenever we add a new activity

to the manifest. Add the activity tag line to your manifest (AndroidManifest.xml) as follows:

<activity android:name=*".QuoteDetail"* />

Now run your application and test it. You should have a complete working app with both a master and detail view! Double Sweeet!!! Pressing your back button should return you to the Master view!

**STEP 9 Adjustments to your view. Adding in your own quotes and images.**

Go back your strings.xml file and adjust the last 5 quotes to some other famous

list of quotes.

Hunt down some appropriate pictures and replace the steve\_5 thru steve\_10 png thumbnails with appropriate pics that match your quotes of some famous person. Replace also the steve\_hd\_5 thru steve\_hd\_10 png’s, which serve as somewhat larger versions of your non-hd files. Of course, as you can notably see, the files when you ran your app show that the master view serves up a thumbnail file, while the hd version serves as a larger file rendered in the detail view along with the appropriate detailed quote. Also notice that the smaller files do not necessarily have to match the larger files but should somehow represent the person you’re quoting! ☺

**STEP 10 Attention all Grads! Adding in a splash activity screen.**

As good practice for your final project, add in a splash screen to learn the process.

Your boss has complemented you on this great app! Now he or she wants a dynamic

splash screen added to the mix! As you may know a splash screen is the first screen scene

at start up.

1. To add in a splash screen, grab an appropriate image and add it into your drawable

folder.

1. Create a java Empty Views Activity file called Splash. Time the splash screen to appear at least for 5 seconds.

When the timer is complete, allow an intent to automatically load your QuoteReaderActivity class file.

Make sure your Splash activity file is logged in your **manifest** file with an intent to launch

first. Your file’s activity’s section should end up looking something like this:

<activity  
 android:name=".Splash"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
</activity>  
<activity  
 android:name=".QuoteReaderActivity"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
</activity>  
<activity android:name=".QuoteDetail" />



My Splash screen snapshot!

R.I.P. Stever!!!

**STEP 11 Submitting your assignment:**

For **full** credit turn in the following **pdf /zip** files filled with this:

* PDFs containing snapshots & all source code & XML files

1. Snapshot of your Master view of some quote other than the supplied Steve Jobs quotes

and your Detail view also of a corresponding quote you picked from your Master   
 view.

Submit your Splash screen snapshot as well if you are doing that for credit.

1. All your java files copied in plus code for both your XML layout files, your strings.xml file as well as your Manifest file.

* Zip file of all your project files.