

Lab 4.1 – Using ListView and ArrayAdapter

Overview:

In this lab you will build a ListView to display a scrollable list of items. You will start with a simple list of strings and then enhance the lab by creating a custom layout for each item in the list, binding the detail display to the Event domain object.

Setup:

You must have completed the prior lab.

Step	Description
1.	Modify the main layout to include a ListView element. This element will hold a list of the garage sales. The default behavior for the list view will allow scrolling. Populate the list from the Event Service.
a.	<p>Edit the file “res/layout/activity_main.xml”. Add the following code after the TextView element:</p> <pre><ListView android:id="@+id/eventlistview" android:layout_height="fill_parent" android:layout_width="fill_parent"> </ListView></pre> <p>Note: It is important to give the ListView an id so that it can be referenced from the code.</p>
b.	<p>Get data from the Event service and populate the ListView. Incorporate the new functionality into a separate method that can be called from the onCreate method.</p> <p>Add the following to the end of the “onCreate” method in “MainActivity.java”.</p> <pre>displayListView();</pre> <p>This will result in a compiler error because the method doesn’t yet exist. You will create the method in the next step.</p>
c.	<p>Create a new method called “displayListView” in “MainActivity.java”. The method should return “void”. Add the following code to the method to get the data from the Event service.</p> <pre>List<Event> events = EventService.getAllEvents(this);</pre> <p>The “getAllEvents” method was included in the lab setup files. You don’t need to code it.</p>
d.	<p>Convert the Event data to an array of Strings by adding the following code to the end of the “displayListView” method.</p> <pre>List<String> listData = new ArrayList<String>(); for (Event event : events) { listData.add(event.getStreet()); }</pre> <p>The String ArrayList contains a list of street addresses. This is all that will be displayed for the event right now.</p>
e.	<p>Create an array adapter to hold the data. The ArrayAdapter is a special Android object used to provide a wrapper around the real data that provides a standard interface that the ListView object can use.</p>

```
final ArrayAdapter<String> arrayAdapter =
    new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        listData);
```

Review the associated JavaDoc for ArrayAdapter to understand the constructor of the ArrayAdapter.

Note. `android.R.layout.simple_list_item_1` is a layout that comes in the Android API. Use it when you have a list of string items that don't require any special formatting.

Below is the contents of the file "simple_list_item_1.xml". **Note:** Do NOT code this, it is provided for reference only.

```
<TextView
    android:id="@android:id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:paddingLeft="6dip"
    android:minHeight="?android:attr/listPreferredItemHeight"/>
```

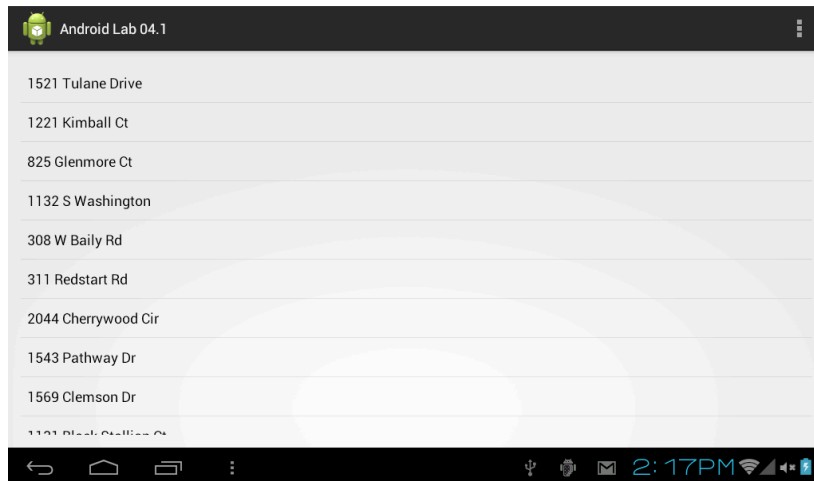
- f. The ArrayAdapter must be bound to the ListView. Add the following code to the end of the "displayListView" method.

```
ListView listView = (ListView) findViewById(R.id.eventlistview);
listView.setAdapter(arrayAdapter);
```

Note: Notice that the code is finding the ListView element in the "main.xml" layout by using the id it was assigned in the XML file.

- g. Remove the TextView for the count of events from "res/layout/activity_main.xml". Also remove the code that refers to it in "MainActivity.java".

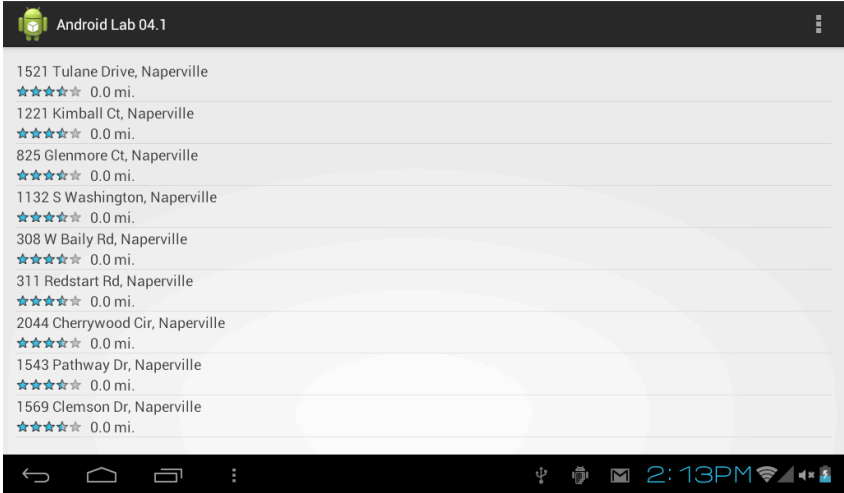
- h. Run the app to verify that a list of addresses is shown. You should see the following (your data may differ slightly):



Use up and down swiping gestures to scroll down the list. You may need to switch the screen to landscape mode so that scrolling is available.

2.	<p>The layout for each item can be more complex, including multiple fields of data and additional formatting. In this step you will replace the default list item layout provided by Android with your own custom version.</p>
a.	<p>Create a new layout file in the “res/layout” directory. The layout should have a meaningful name such as “event_list_item.xml”</p> <p>To create the new layout, highlight the “res” directory in the “AndroidLab” project and right click to get the context menu. Select “New > Other”.</p> <p>Then select “Android > Android XML Layout File” and click the “Next” button. You should see the “New Android Layout XML File” wizard. Enter the following values in the wizard.</p> <p style="padding-left: 40px;">Resource Type: Layout</p> <p style="padding-left: 40px;">File: event_list_item</p> <p style="padding-left: 40px;">Root Element: LinearLayout</p> <p>Click the “Next” button.</p> <p>Don't enter any qualifiers.</p> <p>Select "Finish"</p> <p>Notice the new file called “res/layout/event_list_item.xml”.</p> <p>Android XML files can be edited in a WYSIWYG view called “Graphical Layout” or in raw XML view. In this course we will use the raw XML view. So click on the tab at the bottom of the edit view labeled “event_list_item.xml”.</p>
b.	<p>For each item in the list display the following fields from the Event object:</p> <ul style="list-style-type: none"> • Address (street address and city) • Rating • Distance <p>Add elements to the layout for displaying each item. You can use the following XML snippet as an example or create your own. Be sure to put this code within the existing Layout element.</p> <pre> <TextView android:id="@+id/item_address" android:layout_width="wrap_content" android:layout_height="wrap_content" android:textSize="18dip" /> <LinearLayout android:layout_width="fill_parent" android:layout_height="fill_parent" android:orientation="horizontal" > <RatingBar android:id="@+id/item_rating" style="?android:attr/ratingBarStyleSmall" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_gravity="center" android:isIndicator="true" android:maxHeight="14dip" </pre>

	<pre> android:minHeight="14dip" android:numStars="5" android:stepSize=".5" /> <TextView android:id="@+id/item_distance" android:layout_width="wrap_content" android:layout_height="wrap_content" android:layout_marginLeft="10dip" android:textSize="18dip" /> <TextView android:layout_width="wrap_content" android:layout_height="wrap_content" android:text=" mi." android:textSize="18dip" /> </LinearLayout> </pre> <p>Notice the technique of including a “LinearLayout” within another “LinearLayout” element to group items together. In this case, it is used to get rating and distance on the same line.</p> <p>Note: Layout files can sometimes be quite large. To make them more readable you may want to change the formatting rules. Right click anywhere in the editor to get to the context menu. Select “Preferences” then “XML > XML Files > Editor” to adjust the formatting rules. Many developed like to see each attribute on its own line. Select “Split multiple attributes each on a new line”. Re-format your XML by typing “<ctrl><shft>F” in the editor.</p>
c.	<p>We will now have to create a new ArrayAdapter object where each element in the array is a full Event object (rather than just a string). Then we will have to tell the ArrayAdapter which layout to use and how to bind the fields in the Event with the elements in the layout.</p> <p>Start by creating a new class for the custom array adapter.</p> <p>In “src/com.garagze” create a new class called “EventArrayAdapter” which should take “ArrayAdapter” as a super class. This class uses typing so be sure to specify “Event” as the type. Also create a new constructor that takes 4 parameters as shown in the example below:</p> <pre> public class EventArrayAdapter extends ArrayAdapter<Event> { public EventArrayAdapter(Context context, int resource, List<Event> events) { super(context, resource, events); } } </pre> <p>You must pass a list of Events when creating this adapter. Read the javadoc for ArrayAdapter to understand the requirements for the constructor.</p>
d.	<p>You now need to provide the mapping between the fields in the Event object and the fields on the detail layout. This is done in ArrayAdapter by providing an over-ride to the “getView” method</p> <p>In “src/com.garagze/EventArrayAdapter.java” override the “getView” method. Find each view by looking it up by id. Then populate it with the value from the appropriate Event object.</p> <pre> @Override public View getView(int position, View convertView, ViewGroup parent) { </pre>

	<pre> Event event = getItem(position); if (convertView == null) { LayoutInflater vi = (LayoutInflater) getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE); convertView = vi.inflate(R.layout.event_list_item, null); } TextView address = (TextView) convertView.findViewById(R.id.item_address); address.setText(event.getStreet() + ", " + event.getCity()); RatingBar rating = (RatingBar) convertView.findViewById(R.id.item_rating); rating.setRating(event.getRating()); TextView distance = (TextView) convertView.findViewById(R.id.item_distance); distance.setText(Double.toString(event.getDistance())); return convertView; } </pre>
e.	<p>Replace your current array adapter with the new custom array adapter.</p> <p>Replace the code in "MainActivity.java" that creates the arrayAdapter with the following:</p> <pre> final ArrayAdapter<Event> arrayAdapter = new EventArrayAdapter(this, R.layout.event_list_item, events); </pre>
f.	<p>Run the app to verify that the new item format is shown. You should get the following results:</p> 
3.	<p>Improve the performance of the ListView by implementing the "ViewHolder" technique.</p>
a.	<p>Declare an object to hold the references to the view elements. Consider where this object should be declared (in a class of its own or as an inner class to the adapter?).</p> <p>The holder should contain a property for each view element that you've placed in the line item view.</p>

	<p>Following is the code for the suggested view. You can place with inside the "EventArrayAdapter" class. Consider other places it could be placed.</p> <pre> static class ViewHolder { public TextView text; public ImageView image; } </pre>
b.	<p>Create the holder object and assign values to the view element references.</p> <pre> ViewHolder viewHolder = new ViewHolder(); viewHolder.address = (TextView) convertView .findViewById(R.id.item_address); viewHolder.rating = (RatingBar) convertView .findViewById(R.id.item_rating); viewHolder.distance = (TextView) convertView .findViewById(R.id.item_distance); </pre>
c.	<p>Save the holder object along with the view element it is associated with.</p>