Android Workshop Labs

James Harmon

Version 2016-11-28

Table of Contents

Lab 2.1 – Install Android Studio 1
- Overview 1
- Setup 1
- Steps 1
- Additional Resources and Notes 3

Lab 2.2 – Create HelloWorld App 5
- Overview 5
- Setup 5
- Steps 5

Lab 2.3 – Modify a Simple Android App 10
- Overview: 10
- Setup: 10
- Steps 10

Lab 3.1 – Unit Testing an Android App 14
- Overview 14
- Setup 14
- Steps 14
- Resources 18

Lab 4.1.1 - Using ListView and ArrayAdaptor 20
- Overview 20
- Setup 20
- Steps 20
- Resources 23

Lab 4.1.2 - Building a Complex Adapter for ListView 24
- Overview 24
- Setup 24
- Steps 24

Lab 4.1.3 - Optimize ListView using View Holder Pattern 29
- Overview 29
- Setup 29
- Steps 29

Lab 4.1.4 - Replace ListView with RecylerView 32
- Overview 32
- Setup 32
- Steps 32
- Additional Resources and Notes 34

Lab 4.2 - Create a Menu Item . . . 35
Overview . . . 35
Steps . . . 35
Additional Notes . . . 38
Lab 4.3 - Add Styling . . . 40
Overview . . . 40
Steps . . . 40
Lab 5.1 – Create a Layout . . . 42
Overview: . . . 42
Steps . . . 42
Resources . . . 44
Lab 6.1 – Use Resource Qualifiers . . . 45
Overview: . . . 45
Steps . . . 45
Lab 7.1 - Create A New Activity . . . 46
Overview: . . . 46
Steps . . . 46
Lab 7.2 - Another New Activity . . . 52
Overview . . . 52
Steps . . . 52
Lab 7.3 - Use Espresso to Test App . . . 56
Overview . . . 56
Steps . . . 56
Advanced . . . 57
Resources . . . 57
Lab 7.4 - Running Spoon . . . 59
Overview . . . 59
Steps . . . 59
Resources . . . 60
Lab 8.1 - Fragments . . . 62
Overview: . . . 62
Setup . . . 62
Summary . . . 62
Steps . . . 64
Optional . . . 70
References . . . 70
Lab 9.1 - Local Storage . . . 71
Overview: . . . 71

Steps . . . 71
Lab 9.2.1 - Preferences . . . 73
Overview: . . . 73
Steps . . . 73
Lab 9.2.2 – Preference Fragments . . . 76
Overview: . . . 76
Setup: . . . 76
Steps . . . 76
Lab 9.2.3 - Testing Preferences . . . 77
Overview: . . . 77
Steps . . . 77
Resources . . . 79
Lab 9.3.1 - Database . . . 80
Overview: . . . 80
Steps: . . . 80
Lab 9.3.1 - Database (using Room) . . . 89
Overview: . . . 89
Steps: . . . 89
Optional . . . 92
Resources . . . 92
Lab 9.3.2 – Testing the database . . . 94
Overview . . . 94
Setup . . . 94
Steps . . . 94
Lab 10.1 – Creating a ContentProvider . . . 95
Overview . . . 95
Setup . . . 95
Steps . . . 95
Lab 10.2 – Testing a ContentProvider . . . 100
Overview . . . 100
Setup . . . 100
Steps . . . 100
Lab 11.1 - Asynchronous Tasks . . . 102
Overview: . . . 102
Steps . . . 102
Lab 12.1 - Making HTTP Calls and Parsing JSON . . . 106
Overview: . . . 106
Steps . . . 106

Lab 13.1 - Creating a Service . . . 111
Overview . . . 111
Setup . . . 111
Overview . . . 111
Steps . . . 111
Lab 14.1 - Permissions . . . 113
Overview . . . 113
Setup . . . 113
Steps . . . 113
Resources . . . 114
Lab 15.1 - Broadcast Receiver . . . 115
Overview . . . 115
Setup . . . 115
Overview . . . 115
Steps . . . 115
Lab 16.1 - System Services . . . 117
Overview: . . . 117
Steps . . . 117
Lab 17.1 - Using WebView . . . 121
Overview . . . 121
Setup . . . 121
Steps . . . 121
Lab 18.1 - Display A Map . . . 122
Overview: . . . 122
Steps . . . 122
Resources . . . 126
Lab 18.2 - Map Markers . . . 128
Overview: . . . 128
Setup: . . . 128
Steps . . . 128
18.3 - Location Using Google Mobile Services (GMS) . . . 133
Overview . . . 133
Setup . . . 133
Additional Instructions . . . 133
Definitions . . . 133
Steps . . . 133
Resources . . . 135
Lab 19.1 - Advanced UI . . . 136

Overview . . . 136
Setup . . . 136
Steps . . . 136
Additional Resources . . . 136
Lab 19.2 - Card View . . . 138
Overview . . . 138
Setup . . . 138
Steps . . . 138
Lab 19.3 - Activity Transition . . . 140
Overview . . . 140
Setup . . . 140
Steps . . . 140
Resources . . . 141
Lab 20.1 - Improving Gradle Build Times . . . 142
Overview . . . 142
Setup . . . 142
Steps . . . 142
Resources . . . 143
Other Resources . . . 144
Lab 20.2 - Using Proguard . . . 145
Overview . . . 145
Setup . . . 145
Steps . . . 145
Resources . . . 146
Lab 20.3 - Other Gradle Enhancements . . . 147
Overview . . . 147
Resources . . . 147

Lab 2.1 – Install Android Studio

Overview

In this lab you will install Android Studio and build a simple Hello World Android app.

Setup

This lab assumes that Android Studio has not yet been installed.

Steps

1. Install Android Studio

   Instructions for downloading Android Studio (AS) can be found at:

   ```
   http://developer.android.com/sdk/index.html
   ```

   Click the green "DOWNLOAD ANDROID STUDIO" button. Accept the terms. Click the "DOWNLOAD ANDROID STUDIO" button. This will place the installer on your machine.

   After downloading the install file, the website will display a page with detailed installation instructions. Follow the instructions on the page for installing Android Studio.

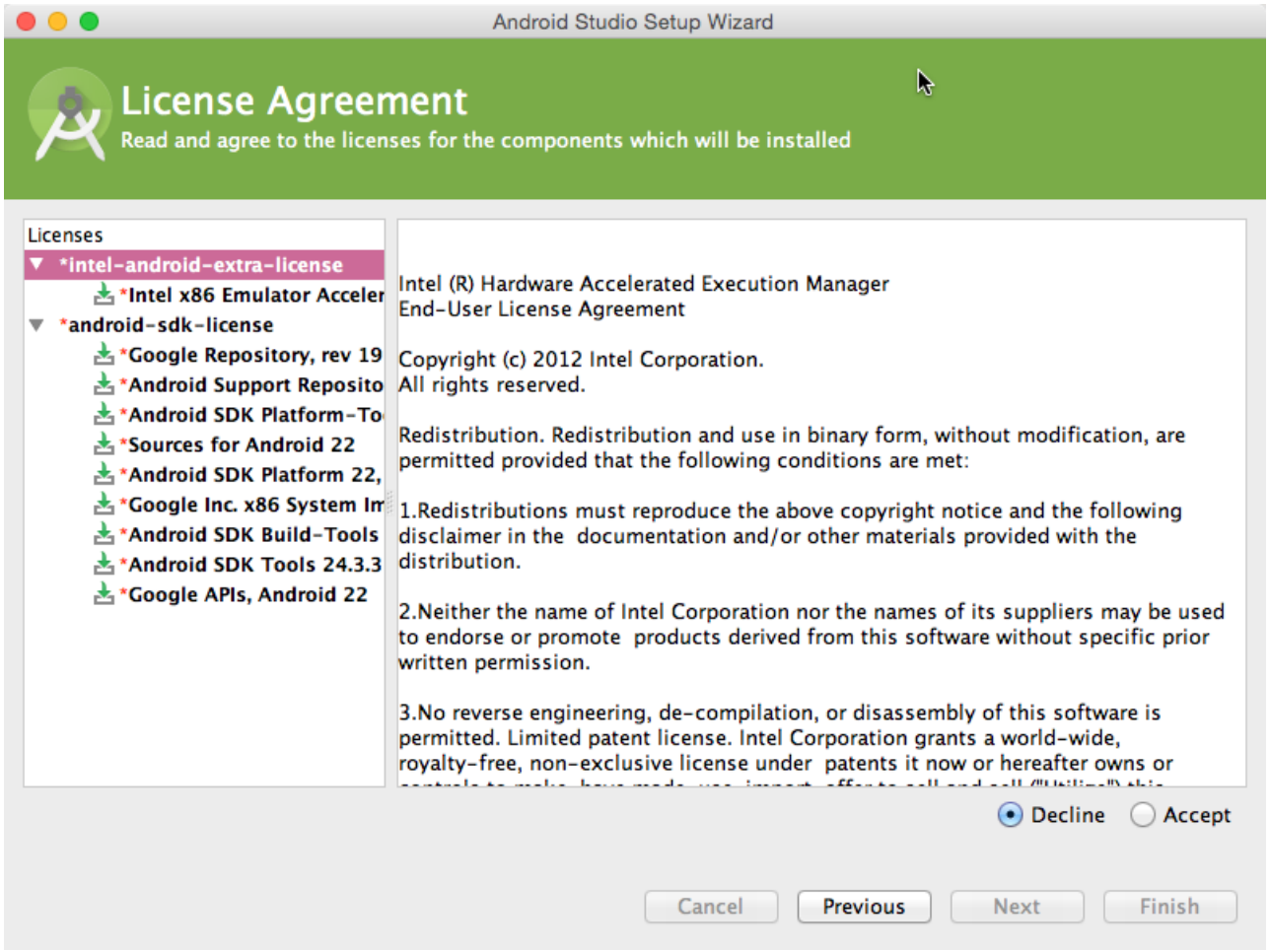
   a. Start AS by running the installed executable. Complete the installation by answering the prompts on the dialog screens.

      During startup, selecting the default options on each of the dialog screens should be sufficient for the installation.

   b. On the "Import Settings" dialog, don't import any prior settings

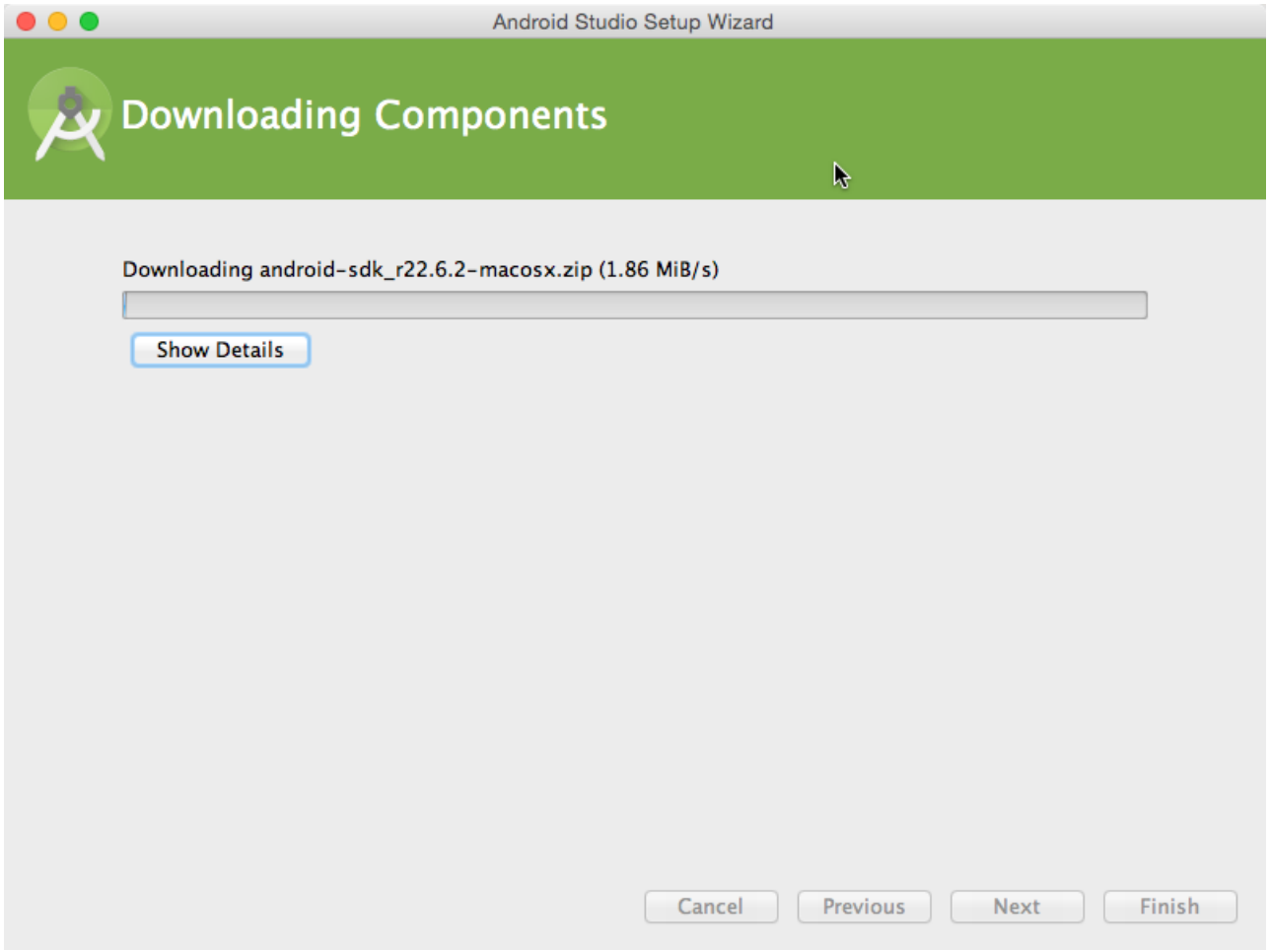
   c. Accept the licenses for the various components shown on the "License Agreement" dialog screen.



Select "intel-android-extra-license" and select "Accept"

Select "android-sdk-license" and select "Accept"

d. Download the additional components. This step could take 5 or more minutes.



e. You may be prompted for your password so that Android Studio can install a special component (called "HAXM") which allows Android emulators to run on your machine.

Additional Resources and Notes

To completely uninstall AS see the following:

http://stackoverflow.com/questions/17625622/how-to-completely-uninstall-android-studio

To remove AS use the following commands on the Mac

```
rm -Rf /Applications/Android\ Studio.app
rm -Rf ~/Library/Preferences/AndroidStudio*
rm ~/Library/Preferences/com.google.android.studio.plist
rm -Rf ~/Library/Application\ Support/AndroidStudio*
rm -Rf ~/Library/Logs/AndroidStudio*
rm -Rf ~/Library/Caches/AndroidStudio*
rm -Rf ~/AndroidStudioProjects
rm -Rf ~/.android
rm -Rf ~/Library/Android*
```

Android Studio will be installed at:

```
For Mac Users: /Applications/Android Studio.app
```

```
For Windows Users: ?
```

The SDK will be installed at:

```
For Mac users: /Users/<user>/Library/Android/sdk/
```

```
For Windows users: ?
```

On Windows machines, it maybe necessary to run Android Studio in Administrator mode (right click on icon and select "Run as Administrator")

Lab 2.2 – Create HelloWorld App

Overview

In this lab you will build a simple Hello World Android app.

Setup

This lab assumes that Android Studio has successfully been installed. If you have not yet installed Android Studio visit the following link to download the Android Studio SDK and read the setup instructions:

```
https://developer.android.com/studio/index.html
```

Steps

1. Create a new Android Project in Android Studio
    a. From the "Welcome to Android Studio" screen select "Start a New Android Studio Project"

    b. Enter the following values in the "Create New Project" wizard:

Application Name	Garage Sales App
Company Domain	com
Project Location	<place this in any directory where you have read/write permission>

Select Next

c. On the "Select the form factors your app will run on" page of the wizard enter the following:

Phone and Tablet	check
Minimum SDK	API 16: Android 4.1 (Jelly Bean)

Select Next

d. On the "Add an activity to Mobile" select "Empty Activity"

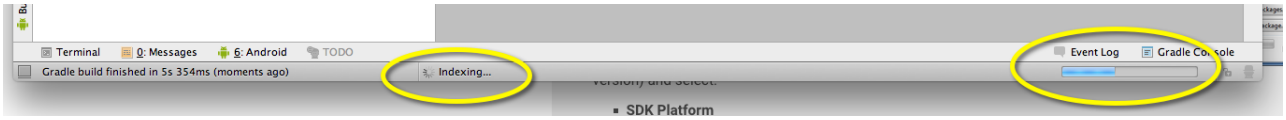
There are multiple activity types - be sure to select the correct one (Empty Activity)

e. On the "Chose options for your new file" enter the following:

Activity Name	MainActivity
Layout Name	activity_main
Backwards Compatability	check

Click Finish

The project is being built. This may take 1 or 2 minutes the first time. Watch for the "Indexing" message at the bottom of the screen (see screenshot). Continue to wait until the message is gone.



Android Studio should now have created a new project in your workspace.

Close the "Tip of the Day" dialog (if it appears).

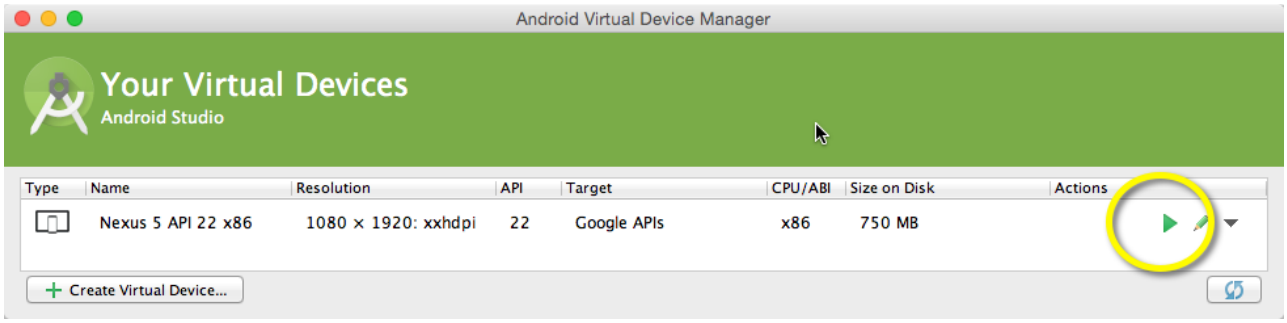
2. Configure an emulator and start it

a. From the menu, select: Tools | Android | AVD Manager

b. If you do not see an emulator configuration you will have to create one by clicking the "+Create Virtual Device" button.

   Create a "Nexus 5" device using the latest API.

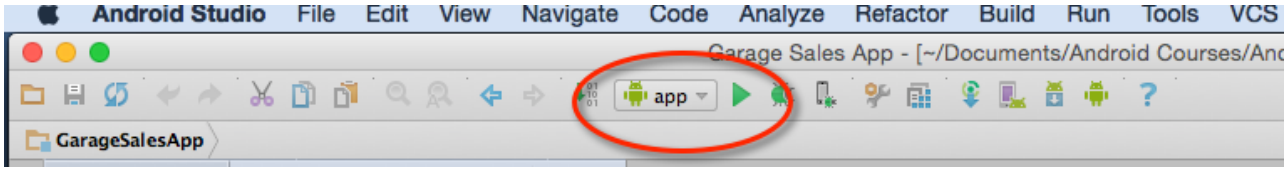
c. Start the emulator by clicking the green play button.



The emulator may take as long as 2 to 3 minutes to fully start. Wait for the home page on the emulator to confirm that it has started (not as the "ANDROID" splash screen).

If there is no emulator configuration you can create a new one by selecting the "+Create Virtual Device" button in the bottom left of the window.

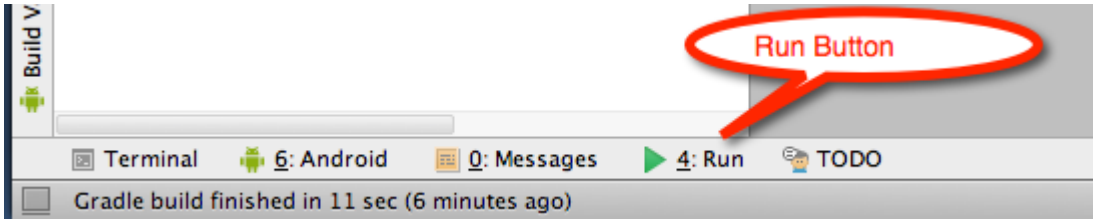
d. Run the application on the emulator by clicking on the green play button to the right of the run configuration list.



Review the console output. You should see output similar to the following:

```
Waiting for device.
Target device: Nexus_5_API_19_2 [emulator-5554]
Uploading file
    local path: /Users/jamesharmon/Documents/Android Courses/Android
Intro/GarageSalesApp/app/build/outputs/apk/app-debug.apk
    remote path: /data/local/tmp/com.garagesalesapp
Installing com.garagesalesapp
DEVICE SHELL COMMAND: pm install -r "/data/local/tmp/com.garagesalesapp"
WARNING: linker: libdvm.so has text relocations. This is wasting memory and is a
security risk. Please fix.
pkg: /data/local/tmp/com.garagesalesapp
Success


Launching application: com.garagesalesapp/com.garagesalesapp.MainActivity.
DEVICE SHELL COMMAND: am start -n
"com.garagesalesapp/com.garagesalesapp.MainActivity" -a android.intent.action.MAIN
-c android.intent.category.LAUNCHER
WARNING: linker: libdvm.so has text relocations. This is wasting memory and is a
security risk. Please fix.
Starting: Intent { act=android.intent.action.MAIN
cat=[android.intent.category.LAUNCHER] cmp=com.garagesalesapp/.MainActivity }
```

Note: To view the console output, select the "Run" at the bottom left of the screen.



The following should be visible in the emulator:


5554:Nexus_5_API_19_2
9:34
Garage Sales
Hello world!

Lab 2.3 – Modify a Simple Android App

Overview:

In this lab you will modify a simple app

Setup:

Prior lab must be completed

Steps

1. Import the library into the project
   a. Select "File | New | Import Module"
   b. From the lab files directory "AndroidLabsSetup" select "garagesales-library" as the "Source directory"

      A new module will be created called "garagesales-library"

      The library contains the following code.

Package	Class File	Description
com.garagesaleslibrary.domain	SaleEvent.java	POJO object containing properties for garage sale events. The events location, description, identity and other properties are contained in it.
com.garagesaleslibrary.feed	EventXMLProcessor	Interface for XML parser
com.garagesaleslibrary.feed	EventXMLProcessorAndroidSAX	XML parser
com.garagesaleslibrary.service	SaleEventManager.java	Service layer for event data access.

      INFO: The library also contains sample data. The directory "res/raw" contains a file called "naper_events.xml" which defines the garage sale events in XML format.

   c. Verify that the library has the correct versions

```
android {
    compileSdkVersion 25
    buildToolsVersion "25.0.2"

    defaultConfig {
        minSdkVersion 16
        targetSdkVersion 25
    }
}
```

The above should be replaced using the external variables that may be defined in the root "build.gradle"

```
ext {
    supportLibVersion = '24.2.1'
    global_compileSdkVersion = 24
    global_buildToolsVersion = "25.0.2"
    global_targetSdkVersion = 24
    global_minSdkVersion = 24
}
```

d. Add the library as a dependency to the app module by updating the dependencies section of the "app/build.gradle" file

```
compile project(':garagesales-library')
```

2. Modify the layout to assign a unique identifier to the TextView element in the layout.

   a. Edit the file “res/layout/activity_main.xml” and add the following attribute to the “TextView” element

```
    android:id="@+id/text_view"
```

   This provides a unique identifier to the view element so that we can reference it from Java code.

   View "R.java" in "app/build/generated/source/r/debug/com/garagesalesapp" and notice the change to the "id" class. If you don’t see "text_view" then clean the project and a new “R.java” file should be generated. The "R.java" file will allow us to access elements by a logical name rather then the integer value (i.e. use "event_count" instead of "0x7f090000").

3. Change the “MainActivity” java file to look up events, calculate a count and display the count.

a. Modify the activity to display the number of events. Edit "MainActivity.java" and add the following code to the end of the “onCreate” method which was generated by the Android project wizard.

```
// Retrieve items from server and get count
int eventCount = getEventCount();

// Build display string
String displayText = "Number of events: " + eventCount;

// Reference view element and set display text
TextView textView = (TextView) findViewById(R.id.text_view);
textView.setText(displayText);
```

Be sure to import the necessary Java classes. You will still get an error due to the reference to a method, “getEventCount”, which hasn’t been created yet. The “getEventCount” method will be created in the next step.

b. Modify the activity to display the number of events.

Create a new method called “getEventCount” which will return the number of events. Start out by hard coding the number of events.

```
private int getEventCount() {
    return 100;
}
```

a. Run the application and verify that it is working.

You should see the following results in your emulator:

Note: The number of events was hardcoded. In your code it should be 100.

a. Now modify the “getEventCount” method to call the service layer to return lookup the events from the server.

```
private int getEventCount() {
    return SaleEventManager.getAllEvents(this).size();
}
```

Run the application again. The number of events should now be "15" which is how many events are in the "naper_events.xml" file.

Note: If the app is not working review the output in the LogCat view to see if a stack trace has been logged.

Lab 3.1 – Unit Testing an Android App

Overview

In this lab you will create some unit tests for your AndroidLab project. Android tests can use jUnit3 or jUnit4. This lab will use jUnit4.

Setup

Lab 02-1 should be completed so that you have a working app to test.

Steps

1. Verify that the build.gradle file declares a dependency on Espresso.

```
dependencies {
    ...
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
}
```

2. Verify that the build.gradle file references the jUnit4 runner

```
defaultConfig {
        ...
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
```

3. Create the directory to contain unit tests.
    a. Inside the module you want to test, navigate to the "src" directory.
    b. Create a new directory called "androidTest". This will be at the same level as the "main" directory.

This directory may already exist depending on how you created the module.

    c. Inside the "androidTest" directory create a directory called "java". This is where the unit test code will be placed.
    d. Inside "java" create a package with the same value as in your "main/java" directory. Use the

package wizard.

Create a package named "com.garagesalesapp".

Using the same package name is not required but ensures that the test classes have the save "visibility" as the classes under test.

4. Create a unit test. Unit tests will be placed in the java package under the "androidTest" directory.

   Unit tests are created as java code.

   a. In src/androidTest/java/com.garagesalesapp create a java class named "SaleEventManagerTest"

      Select "New | Java Class". There is no specific wizard for unit tests.

      ```
      @RunWith(AndroidJUnit4.class)
      public class SaleEventManagerTest {
      }
      ```

   b. And create a method call "testGetAllEvents"

      ```
      @Test
      public void testGetAllEvents() {
              fail("Not yet implemented");
      }
      ```

      This follows the "fail first" strategy of unit testing

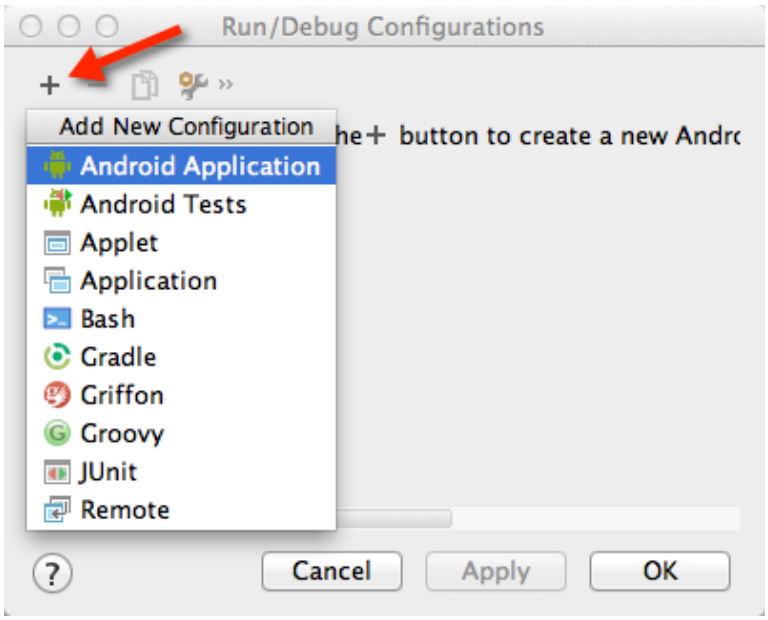
5. Run the unit test and verify that it fails.

   a. Right click on the file and select "Run"

6. You can also create a run configuration to run the test.

   a. Select "Run | Edit Configurations" from the top level menu.

   b. Select "+" to add a new run configuration and select "Android Tests"



c. Define the test by entering the following values in the wizard:

Name	app-test
Module	app
Test	All in Module
Specific instrumentation runner	blank
Target Device	Show chooser dialog

The wizard should look like this:

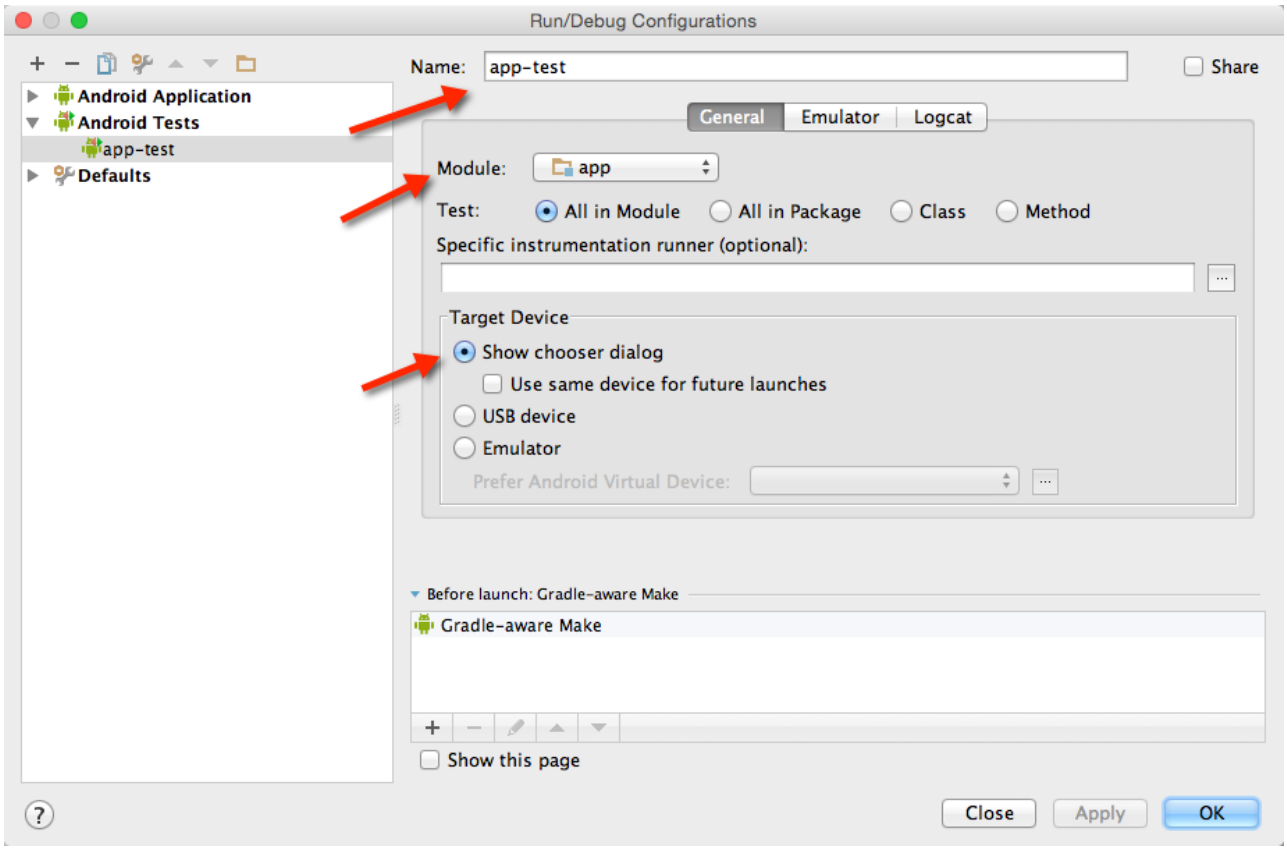
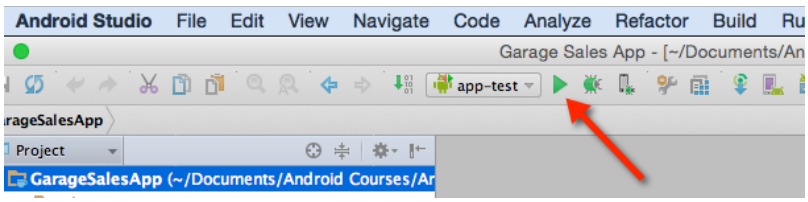


Figure 1. Android Test Wizard

d. Run "app-test" using the play button.



Under the "Run" tab at the bottom of the screen you should see the jRunner output:

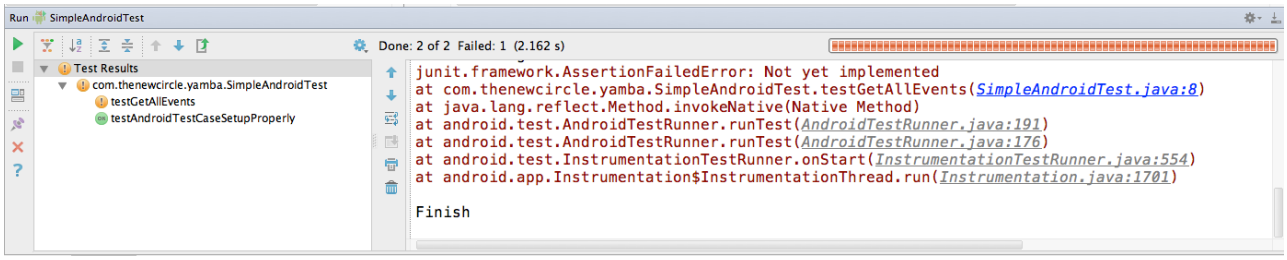


Figure 2. Runner Output

Note: The test failed because you forced it to.

7. Create a working test
8. Add the following test method to the test

```
@Test
public void getAllEvents() {
    Context context = InstrumentationRegistry.getTargetContext();
    List<SaleEvent> events =
                SaleEventManager.getAllEvents(context);

    assertNotNull(events);
    assertTrue(events.size() > 0);
    assertEquals(15, events.size());
}
```

9. Run "app-test" again using the play button.

   Under the "Run" tab at the bottom of the screen you should see the jRunner output:

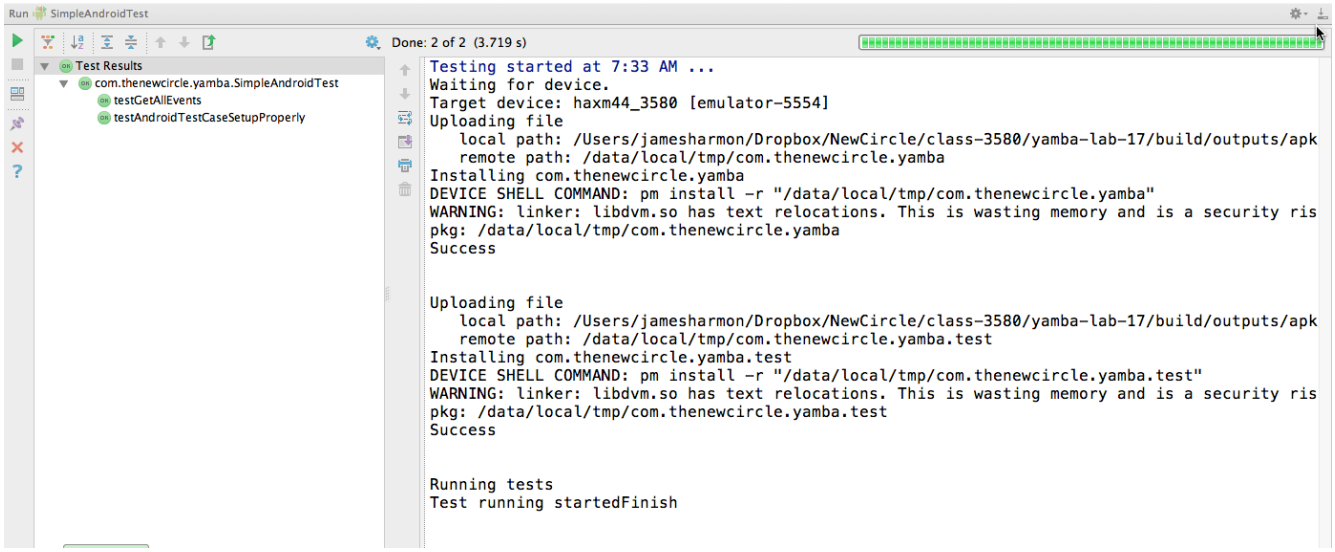


Figure 3. Runner Output

   Note: This time you should see a green bar to signify that the test worked (Green is clean!!!, Red is dead!!!)

Resources

Console output for long running tasks

http://trickyandroid.com/gradle-tip-4-log-unit-test-execution-events-into-console/

These instructions were adapted from the following:

```
http://developer.android.com/tools/testing-support-library/index.html
```

```
http://tools.android.com/tech-docs/new-build-system/user-guide#TOC-Testing
```

Lab 4.1.1 - Using ListView and ArrayAdaptor

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 the list items and binding the detail display to the Event domain object.

Setup

You must have completed the prior labs.

Steps

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. Edit the file "res/layout/activity_main.xml". Add the following code after the TextView element:

   ```
   <ListView android:id="@+id/eventlistview"
       android:layout_height="match_parent"
       android:layout_width="match_parent">
   </ListView>
   ```

   It is important to give the ListView an id so that it can be referenced from the code.

2. Get data from the SaleEventManager and populate the ListView. Incorporate the new functionality into a separate method that can be called from the onCreate method.

   a. Add the following to the end of the "onCreate" method in "MainActivity.java".

   ```
   displayListView();
   ```

   This will result in a compiler error because the method doesn't yet exist. You will create the method in the next step.

b. 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.

```
List<SaleEvent> events = SaleEventManager.getAllEvents(this);
```

The "getAllEvents" method was included in the lab setup files. You don't need to code it.

c. Convert the Event data to an array of Strings by adding the following code to the end of the "displayListView" method.

```
List<String> listData = new ArrayList<String>();
for (SaleEvent event : events) {
    listData.add(event.getStreet());
}
```

The String ArrayList contains a list of street addresses. This is all that will be displayed for the event right now.

d. Create an array adapter to hold the data. The ArrayAdapter is a special Android object used to provide a wrapper around the data and provide a standard interface that the ListView object can use.

```
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.

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 content 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="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:gravity="center_vertical"
    android:paddingLeft="6dip"
    android:minHeight="?android:attr/listPreferredItemHeight"/>
```

e. 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);
```

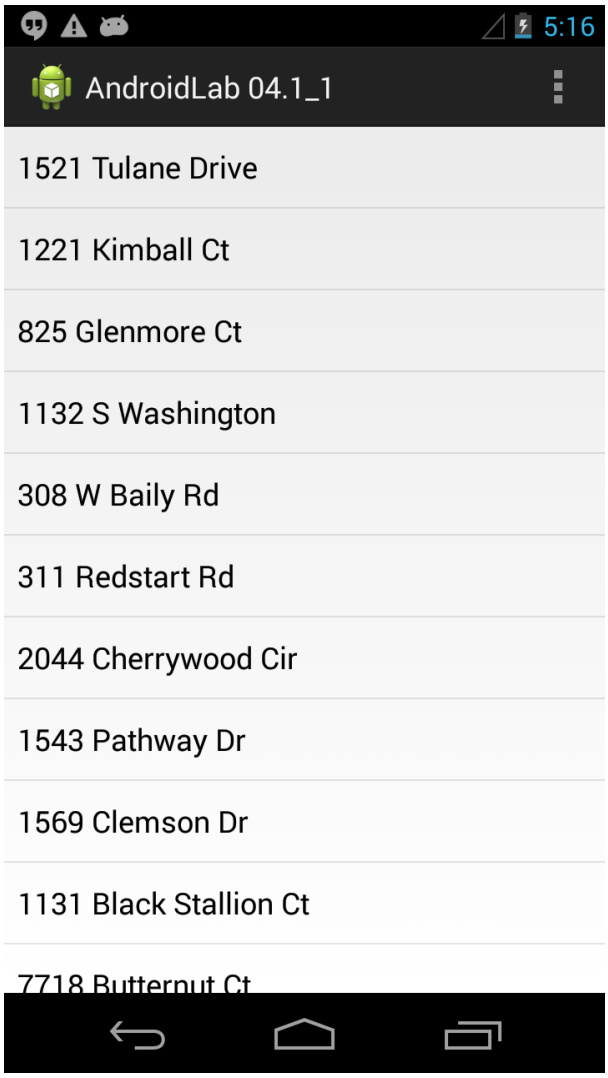
Notice that the code is finding the ListView element in the "main.xml" layout by using the id assigned to it in the XML file.

f. Add an event handler to be used when the user clicks on a row.

```
private final static String TAG = "MainActivity"

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
  @Override
  public void onItemClick(AdapterView<?> parent, View view, int position, long id)
{
    Log.v(TAG, "User Clicked On ListView Position " + position);
    Toast.makeText(getApplicationContext(),
        "ListView Position " + position, Toast.LENGTH_SHORT).show();
  }
});
```

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.

Click on a row to verify that the toast appears.

Resources

Remember to check the javadoc for ListView

List views also support headers, footers and separators.

Lab 4.1.2 - Building a Complex Adapter for ListView

Overview

In this lab you will create an adapter to display a more complex view

Setup

You must have completed the prior labs.

Steps

1. 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.

   a. Create a new layout file in the "res/layout" directory. The layout should have a meaningful name such as "event_list_item.xml"

      To create the new layout, highlight the "res" directory in the "AndroidLab" project and right click to get the context menu. Select "New > Other".

      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.

Resource Type	Layout
File	event_list_item
Root Element	LinearLayout

      Click the "Next" button.

      Don’t enter any qualifiers.

      Select "Finish"

      Notice the new file called "res/layout/event_list_item.xml".

      Android XML files can be editted 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".

2. For each item in the list display the following fields from the Event object:
    - Address (street address and city)
    - Rating
    - Distance

    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.

```
<TextView
    android:id="@+id/item_address"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_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"
        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="18sp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=" mi."
        android:textSize="18sp"/>

</LinearLayout>
```

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.

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 buy typing "<ctrl><shft>F" in the editor.

3. We will now have to create a new ArrayAdapter object where each element in the array is a full SaleEvent 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 SaleEvent with the elements in the layout.

   a. Start by creating a new class for the custom array adapter. In "src/com.garagesalesapp" create a new class called "SaleEventArrayAdapter" which should take "ArrayAdapter" as a super class. This class uses typing so be sure to specify "SaleEvent" as the type. Also create a new constructor that takes 3 parameters as shown in the example below:

```
public class SaleEventArrayAdapter extends ArrayAdapter<SaleEvent> {

  public SaleEventArrayAdapter
    (Context context, int resource, List<SaleEvent> events) {
    super(context, resource, events);
  }

}
```

   You must pass a list of Events when creating this adapter. Read the javadoc for ArrayAdapter to understandard the requirements for the constructor.

   b. 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

   In "src/com.garagesalesapp/SaleEventArrayAdapter.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.

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    SaleEvent 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;
}
```

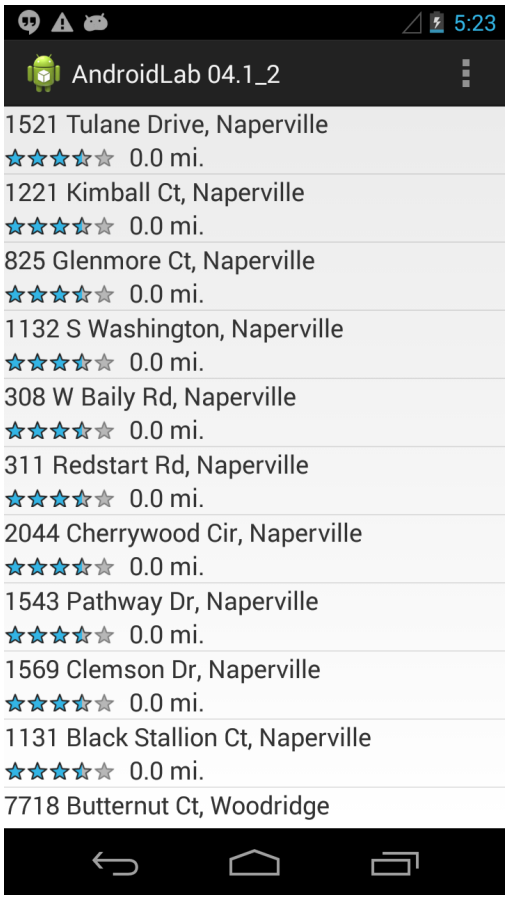
getView returns a View. ListView manages views not data.

c. Replace your current array adapter with the new custom array adapter.

   Replace the code in "MainActivity.java" that creates the arrayAdapter with the following:

```
final ArrayAdapter<SaleEvent> arrayAdapter =
    new SaleEventArrayAdapter(this, R.layout.event_list_item, events);
```

d. Run the app to verify that the new item format is shown. You should get the following results:


5:23
AndroidLab 04.1_2
1521 Tulane Drive, Naperville
0.0 mi.
1221 Kimball Ct, Naperville
0.0 mi.
825 Glenmore Ct, Naperville
0.0 mi.
1132 S Washington, Naperville
0.0 mi.
308 W Baily Rd, Naperville
0.0 mi.
311 Redstart Rd, Naperville
0.0 mi.
2044 Cherrywood Cir, Naperville
0.0 mi.
1543 Pathway Dr, Naperville
0.0 mi.
1569 Clemson Dr, Naperville
0.0 mi.
1131 Black Stallion Ct, Naperville
0.0 mi.
7718 Butternut Ct, Woodridge

Lab 4.1.3 - Optimize ListView using View Holder Pattern

Overview

In this lab you will optimize the performance of the ListView by implementing a View Holder

Setup

You must have completed the prior labs.

Steps

1. Improve the performance of the ListView by implementing the "ViewHolder" technique.
    a. 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?).

    The holder should contain a property for each view element that you've placed in the line item view.

    Following is the code for the suggested view. You can place with inside the "SaleEventArrayAdapter" class. Consider other places it could be placed.

    ```
    static class ViewHolder {

        ViewHolder(View view) {
            address = (TextView) view.findViewById(R.id.item_address);
            rating = (RatingBar) view.findViewById(R.id.item_rating);
            distance = (TextView) view.findViewById(R.id.item_distance);
        }

        public TextView address;
        public RatingBar rating;
        public TextView distance;
    }
    ```

    b. Create the holder object and assign values to the view element references.

    ```
    viewHolder = new ViewHolder(convertView);
    ```

c. Save the holder object along with the convert view element that it is associated with.

```
convertView.setTag(viewHolder);
```

d. In the getView method of the adapter retrieve the view holder from existing views.

```
ViewHolder viewHolder = null;
if (convertView == null) {
    LayoutInflater vi = (LayoutInflater)
    getContext().getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    convertView = vi.inflate(R.layout.event_list_item, null);
    viewHolder = new ViewHolder(convertView);
    convertView.setTag(viewHolder);
} else {
    viewHolder = (ViewHolder) convertView.getTag();
}
```

e. Then populate the layout using the view holder object

```
viewHolder.address.setText(event.getStreet() + ", " + event.getCity());
viewHoler.rating.setRating(event.getRating());
viewHolder.distance.setText(Double.toString(event.getDistance()));
```

f. The final version of the SaleEventArrayAdapter class looks like this:

```
public class SaleEventArrayAdapter extends ArrayAdapter<SaleEvent> {

    public SaleEventArrayAdapter(Context context, int resource, List<SaleEvent>
events) {
        super(context, resource, events);
    }

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

        ViewHolder viewHolder = null;

        if (convertView == null) {
            LayoutInflater vi = (LayoutInflater) getContext().getSystemService(
                    Context.LAYOUT_INFLATER_SERVICE);
            convertView = vi.inflate(R.layout.event_list_item, null);

            viewHolder = new ViewHolder(convertView);
```

```
            convertView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolder) convertView.getTag();
        }

        SaleEvent event = getItem(position);

        viewHolder.address.setText(event.getStreet() + ", " + event.getCity());
        viewHolder.rating.setRating(event.getRating());
        viewHolder.distance.setText(Double.toString(event.getDistance()));

        return convertView;

    }

    static class ViewHolder {

        ViewHolder(View view) {
            address = (TextView) view.findViewById(R.id.item_address);
            rating = (RatingBar) view.findViewById(R.id.item_rating);
            distance = (TextView) view.findViewById(R.id.item_distance);
        }

        public TextView address;
        public RatingBar rating;
        public TextView distance;
    }

}
```

Lab 4.1.4 - Replace ListView with RecylerView

Overview

In this lab you will replace the ListView with a RecyclerView. You will also use Cards to display the detail items.

Setup

You must have completed the prior labs.

Steps

1. Add the recycler view and card libraries to the module dependencies

```
dependencies {
    ...
    compile 'com.android.support:recyclerview-v7:25.3.1'
    compile 'com.android.support:cardview-v7:25.3.1'
}
```

Use the correct version for the library (i.e. most recent)

It is usually better to specify full library version instead of using '+'

2. Replace ListView with RecyclerView in "layout/activity_main.xml"

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/eventlistview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

3. Wrap top level layout with card view in "layout/event_list_item.xml" Insert Cardview at the top of the file.

```
<android.support.v7.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/placeCard"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    card_view:cardElevation="8dp">
```

4. Create new Adapter for RecyclerView. This is a new java class.

```
public class SaleEventRecyclerAdapter
        extends RecyclerView.Adapter<SaleEventRecyclerAdapter.ViewHolder> {

    private static final String TAG = SaleEventRecyclerAdapter.class.getSimpleName();

    Context mContext;

    List<SaleEvent> events;

    public SaleEventRecyclerAdapter(Context context, List<SaleEvent> events) {
        this.mContext = context;
        this.events = events;
    }

    public class ViewHolder extends RecyclerView.ViewHolder {

        public TextView address;
        public RatingBar rating;
        public TextView distance;

        public ViewHolder(View itemView) {
            super(itemView);
            address = (TextView) itemView.findViewById(R.id.item_address);
            rating = (RatingBar) itemView.findViewById(R.id.item_rating);
            distance = (TextView) itemView.findViewById(R.id.item_distance);
        }
    }

    @Override
    public int getItemCount() {
        return events.size();
    }

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
```

```
        View view =
            LayoutInflater.
            from(parent.getContext()).
            inflate(R.layout.event_list_item, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(final ViewHolder holder, final int position) {
        final SaleEvent event = events.get(position);
        holder.address.setText(event.getStreet() + ", " + event.getCity());
        holder.rating.setRating(event.getRating());
        holder.distance.setText(Double.toString(event.getDistance()));
    }
}
```

5. Replace "displayListView" method with new method "displayRecyclerView" and call it from "onCreate"

```
private RecyclerView mRecyclerView;
private LinearLayoutManager mLinearLayoutManager;

private SaleEventRecyclerAdapter mAdapter;

private void displayRecyclerView() {
  mRecyclerView = (RecyclerView) findViewById(R.id.eventlistview);

  // Build layout manager
  mLinearLayoutManager =
      new LinearLayoutManager(this, LinearLayoutManager.VERTICAL, false);
  mRecyclerView.setLayoutManager(mLinearLayoutManager);

  List<SaleEvent> events = SaleEventManager.getAllEvents(this);
  mAdapter = new SaleEventRecyclerAdapter(this, events);
  mRecyclerView.setAdapter(mAdapter);
}
```

6. OPTIONAL: Change the layout of the RecyclerView to horizontal

Additional Resources and Notes

This exercise was adapted from

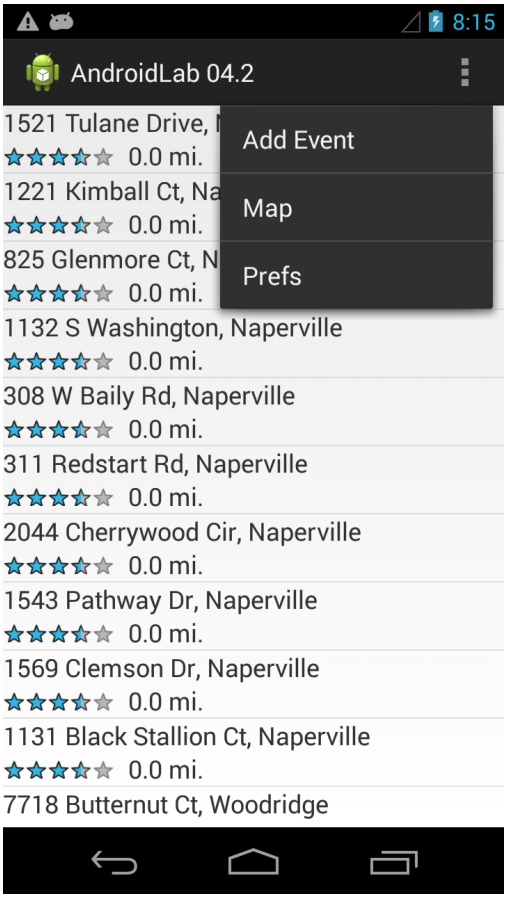
http://www.raywenderlich.com/103367/material-design

Lab 4.2 - Create a Menu Item

Overview

In this lab you create a new menu options that can be selected by the user.

In Android 4.1 and above the menu appears as 3 stacked dots in the upper right of the screen. The menu is activated by clicking on the dots which causes a list of menu items to appear.



Steps

1. Create the XML file which defines the menu options.

   Note: In some versions of Android Studio this will be generated for you automatically.

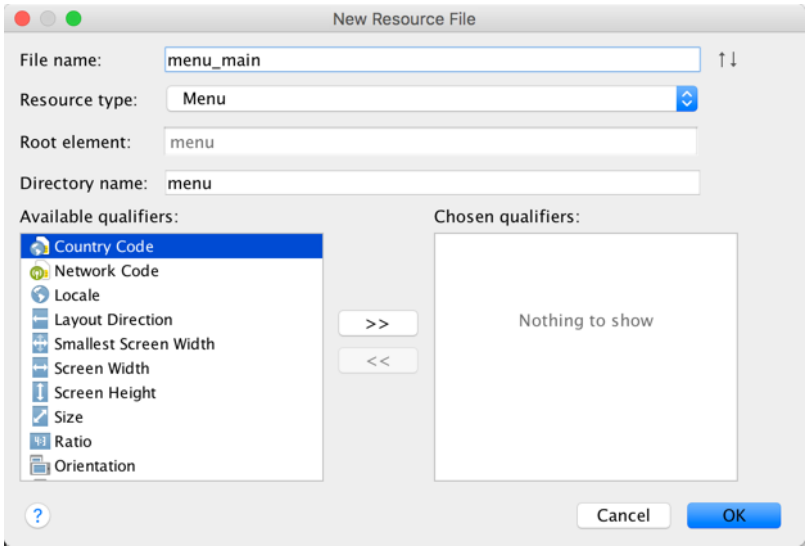
   a. Start the New Android XML File wizard by selecting "AndroidLab/res" and right clicking to reach the context menu

   b. Select "New | Android Resource File" and provide the following:

File Name	menu_main

Resource Type	Menu
Root Element	menu
Directory Name	menu

The wizard should appear like this:



c. Click "OK" and verify that the following file has been created "/res/menu/main_menu.xml"

d. Add items to the menu. The menu will have options for

- Adding a new event
- Showing the events on a map
- Displaying the preferences page

Edit the file "res/menu/menu_main.xml" and add "item" elements within the "menu" tag which will declare the menu options. The completed file should look like the following code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/mi_add_event"
        android:title="@string/mi_add_event" />
    <item
        android:id="@+id/mi_show_map"
        android:title="@string/mi_show_map" />
    <item
        android:id="@+id/mi_settings"
        android:title="@string/mi_settings" />
</menu>
```

Note: Icons for the menus can be found in "AndroidLabSetup/Lab4.2"

e. The menu layout file references some external strings. Enter string values for each of the menu items. Open the file "res/values/strings.xml" and add the following:

```
<string name="mi_add_event">Add Event</string>
<string name="mi_show_map">Map</string>
<string name="mi_settings">Settings</string>
```

2. Modify the main activity to display the menu

a. Add the code for declaring the menu. Open the file "src/com.garagesalesapp/MainActivity.java" and add the following method to the end of the program:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

Note: In some versions of Android Studio this method may already have been generated for you.

Note: Review the javadoc for the "MenuInflator" class.

Test the application by running it on the emulator. Press the "Menu" icon (vertical series of 3 squares) to display the menu. The menu options should display. You can click them but no functionality has been added to them yet.

3. Modify the activity to respond to menu selections.

a. Add the code for responding to the menu selections from the user.

Open the file "src/com.garagesalesapp/MainActivity.java" and override the "onOptionsItemSelected" method.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
    case R.id.mi_add_event:
        addEvent();
        return true;
    case R.id.mi_show_map:
        showMap();
        return true;
    case R.id.mi_settings:
        showPrefs();
        return true;
    default:
        return super.onOptionsItemSelected(item);
    }
}
```

4. Add code for implementing each of the menu selections
    a. Add the following methods to MainActivity.java:

```
private void showPrefs() {
  Log.v(TAG, "Running showPrefs method.");
}

private void showMap() {
  Log.v(TAG, "Running showMap method.");
}

private void addEvent() {
  Log.v(TAG, "Running addEvent method.");
}
```

Note: Run the app again and select each of the menu options. The application will write a log message when a menu item is clicked by the user. In subsequent labs we'll be adding the functionality for each menu option.

Additional Notes

In some versions of Android Studio, the wizards may generate default menus. You can modify the menu or delete it so that you can create it yourself.

The default menu is contained in the file "res/menu/menu_main.xml" and looks like the following:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

<item
  android:id="@+id/action_settings"
  android:orderInCategory="100"
  android:showAsAction="never"
  android:title="@string/action_settings"/>
</menu>
```

Lab 4.3 - Add Styling

Overview

In this lab you will add styling, themes and icons to the application.

Steps

1. Create style for the text fields. All of the text fields should share a set of common style properties. Encapsulate those properties in a style and reference the style instead of repeating each property for every text field.

    a. The list view item text fields in the suggested layout (address, distance and the literal "mi") in "event_list_item.xml" all share the following properties.

    ```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="18sp"
    ```

    Rather than repeat these values for each field, they can be referenced as an external style.

    Modify the file "res/values/styles.xml" and enter the following at the bottom of the file before the ending </resource> tag:

    ```
    <style name="standardText">
        <item name="android:layout_width">wrap_content</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:textSize">18sp</item>
    </style>
    ```

    Note: Notice that a new for "textColor" was added. This will make your changes stand out so it is easier to verify that the style was applied. You can remove or change this value later.

    b. Replace the individual style properties with the new style in "res/layout/event_list_item.xml". Do this for all the TextView fields.

    ```
    <TextView
        style="@style/standardText"
    ```

    The style name "standardText" is the same as the name in the "style.xml" file.

Also notice that you don’t use the "android:" prefix on the attribute name.

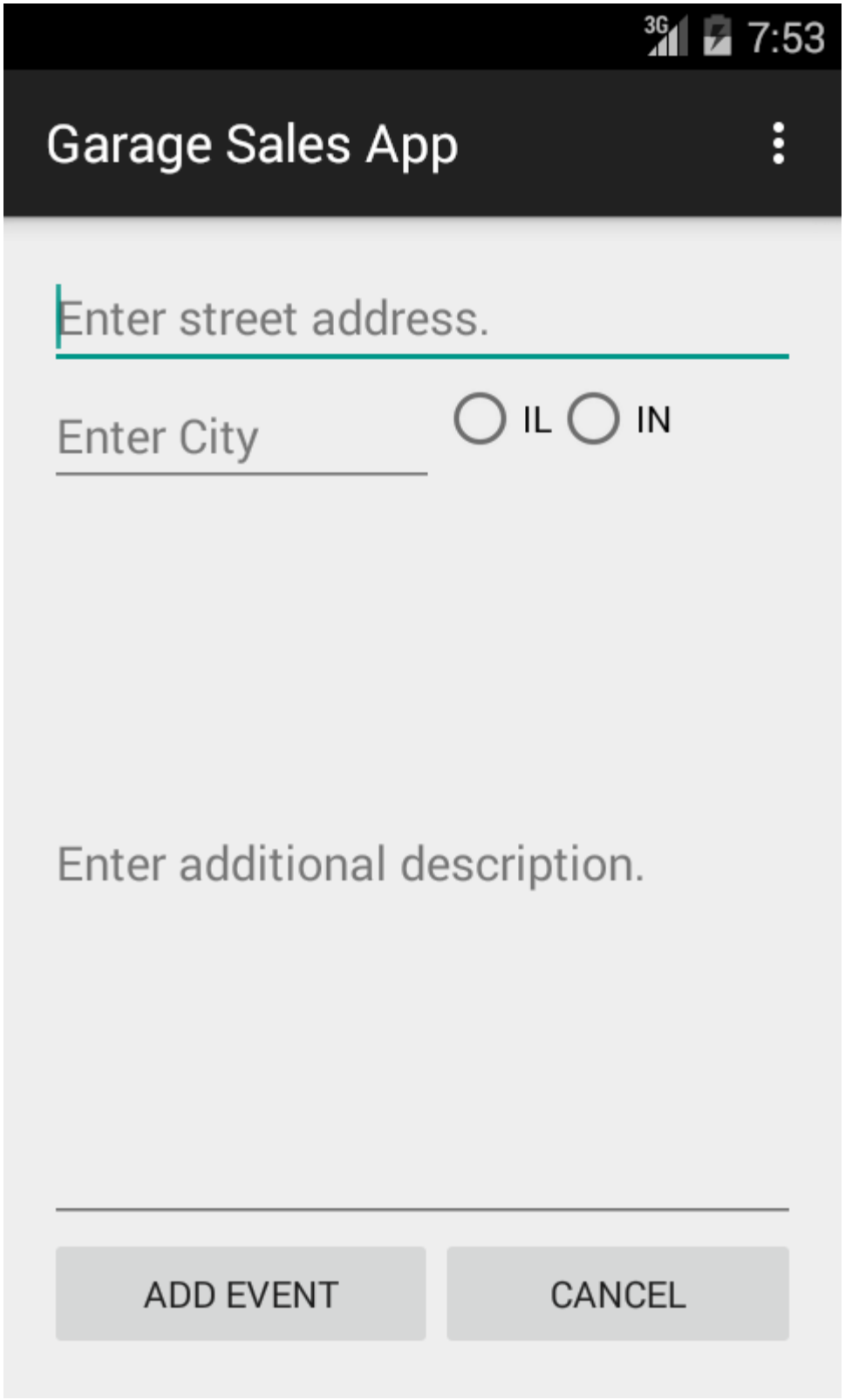
Lab 5.1 – Create a Layout

Overview:

In this lab you will create a layout using various techniques discussed in the topic slides

Steps

1. Create a portrait mode layout to match the screen image below. Call the layout "add_event.xml"



Use both the text and design modes

Use the "tools" attributes to maximize the usefulness of the design (preview) mode

2. Modify the layout containing the ListView to specify which view should be used for each item in the view.

   When editing the Add a tools attribute to specify the list item layout

   ```
   <TextView
       ...
       tools:text="123 Main St">
   </TextView>
   ```

   You can also tell a ListView which detail view to use.

   ```
   <ListView
       ...
       tools:listitem="@layout/event_list_item">
   </ListView>
   ```

Resources

Tools Attributes http://tools.android.com/tech-docs/tools-attributes

Tools Design Layout Attributes http://tools.android.com/tips/layout-designtime-attributes

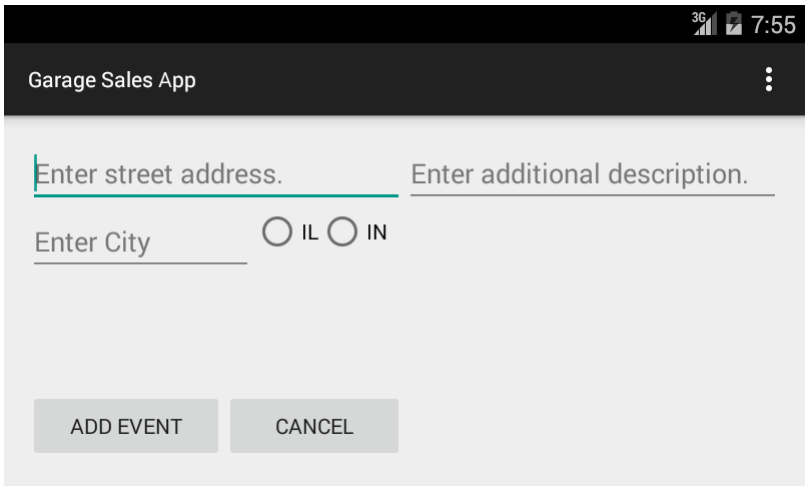
Lab 6.1 – Use Resource Qualifiers

Overview:

In this lab you will create a layout for wide screens

Steps

1. Create a landscape mode layout to match the screen image below:



Try using different qualifiers such as "wNNNdp" and "swNNNdp"

2. (OPTIONAL) Create a spanish language version of your app.
   a. Create a new resource directory "values-es"
   b. Copy "strings.xml" from "values" to "values-es"
   c. Provide translations for strings in "strings.xml"
   d. Change the device language settings to Spanish

Lab 7.1 - Create A New Activity

Overview:

In this lab you will create a new activity to allow a user to enter a new event item. This activity will be initiated by an Intent that will created when the user taps on the appropriate menu button.

Steps

1. Create a new layout for entering event information.

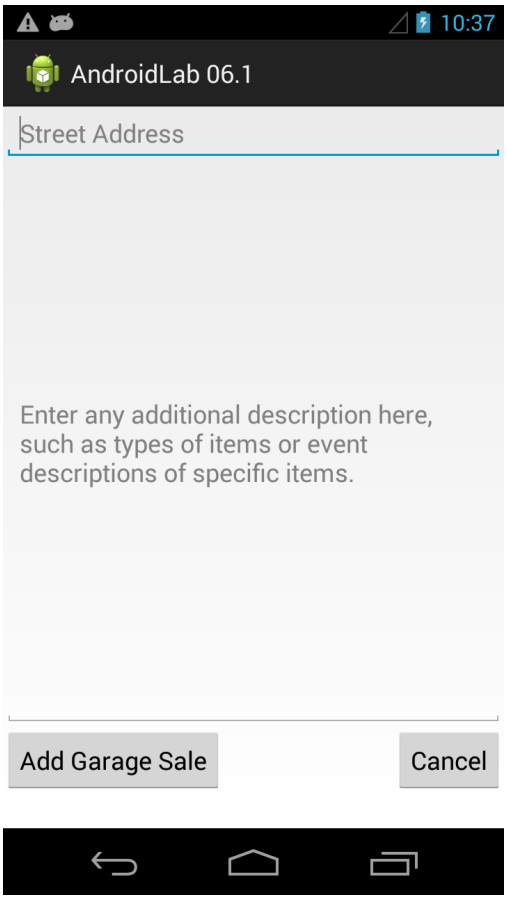
You already created a layout for new events in the last lab. You may use that layout if you prefer.

a. Create a new layout using the Android XML wizard. Call the layout "res/layout/add_event.xml".

Add the following view elements

- EditText view for entering address
- EditText view for entering description
- Button for adding event
- Button for cancelling activity

Although you do not need to match this layout exactly, yours should look something like the following:



Use the hint attribute to show suggestion text in the edit box.

```
android:hint="@string/event_description_hint"
```

You can use the following layout or create your own:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <EditText
        android:id="@+id/event_address"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:ems="10"
        android:hint="@string/address_hint">
        <requestFocus />
    </EditText>

    <EditText
        android:id="@+id/event_description"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/add_garage_sale"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_below="@+id/event_address"
        android:ems="10"
        android:hint="@string/desc_hint"/>

    <Button
        android:id="@+id/add_garage_sale"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_marginBottom="25dp"
        android:text="@string/add_button_text" />

    <Button
        android:id="@+id/cancel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/add_garage_sale"
        android:layout_alignBottom="@+id/add_garage_sale"
        android:layout_alignParentRight="true"
        android:text="@string/cancel_button_text" />

</RelativeLayout>
```

b. Externalize string values in the layout as much as possible. For example, the text for the view showing the event name could have a default value. Rather than hard coding this in the layout, reference it in the layout and put the value in "res/values/strings.xml" as shown below:

   In "res/layout/add_event.xml" replace direct string references such as the following:

   ```
   android:text="Add Garage Sale"
   ```

   with an external string reference such as the following:

   ```
   android:text="@string/add_button_text"
   ```

   and then add the value for "add_button_text" to "res/values/strings.xml as follows:

   ```
   <string name="add_button_text">Add Garage Sale</string>
   ```

   Apply this technique to all the strings in the XML layout files.

2. Create an activity to display the layout for a new event and save the event data using the event service.

   a. Create a new activity called "AddEventActivity.java" in "com.garagesalesapp" package. Extend "AppCompatActivity"

   b. Over-ride the "onCreate" method. Add code to call the super class constructor and inflate the "add_event" layout:

   ```
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.add_event);
   }
   ```

3. Declare the new activity in the application manifest.

   a. Open the file "AndroidManifest.xml" Add the following element to the application tag.

   ```
   <activity
       android:name=".AddEventActivity">
   </activity>
   ```

4. Call the activity from the main menu in "MainActivity.java"

a. Add the following code to the "addEvent" method in the "MainActivity.java" file.

```
Intent intent = new Intent(this, AddEventActivity.class);
startActivity(intent);
```

Run the application. Press the Menu button and select "Add Event". The add event activity should appear.

5. In the "AddEventActivity", add an event handler to the button to actually save the new event. This code should be added to the "onCreate" method of the "AddEventActivity.java" file. Make sure you are editting the correct file.

a. The "Add Garage Sale" button should create an event object and run the SaleEventManager.addEvent method. Add the following code to the end of the "onCreate" method in "AddEventActivity.java". Be sure that the id for the view elements correspond to what you named them in the "add_event.xml" file.

```
final Button addButton = (Button) findViewById(R.id.add_garage_sale);

addButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        final EditText addressView =
            (EditText) findViewById(R.id.event_address);
        String address = addressView.getText().toString();
        final EditText descriptionView =
            (EditText) findViewById(R.id.event_description);
        String description = descriptionView.getText().toString();

        SaleEvent event = new SaleEvent();
        event.setStreet(address);
        event.setDescription(description);
        event.setRating(3.0F);
        event.setCity("Naperville");
        SaleEventManager.addEvent(AddEventActivity.this, event);

        finish();
    }
});
```

b. Add an event handler to the cancel button also. The cancel button should just call the activity "finish()" method.

Optional: There is another technique for assigning event handlers to buttons. Declare an "onClick" attribute for the button in the XML layout file and reference a method name in the activity. The method in the activity should then contain the event handler code. The instructor

will demonstrate this technique during the review of the exercise.

6. When the "AddEventActivity" finishes, Android automatially returns to the item on the stack just below it which is the "MainActivity". As desribed in the activity lifecycle, when an item on the stack comes to the top, it's "onResume" method is called. You need to provide an implementation of this method which will create a new "SaleEventArrayAdapter" to recognize the update to the events data.

   a. Add the following method to the "MainActivity.java" file.

```
@Override
protected void onResume() {
    super.onResume();
    Log.v("MainActivity", "Running onResume method");
    displayRecyclerView();
}
```

   Optional: Explore the error you get if you do not provide an "onResume" method.

   Is it necessary to also call "displayRecyclerView" in the "onCreate" method:

   Run the application and add a new event. Verify that the event is now appearing in the list view.

   The event is only persisted while the application is active. If you run the application again, the new events will be gone.

Lab 7.2 - Another New Activity

Overview

In this lab you will create an additional activity to display the detail for an event.

Steps

1. Add an additional activity for showing Event details so that we have multiple activities (fragments) to display at one time on larger screens. You will create a new activity for viewing event detail information. This activity will be shown when the user clicks on an item in the event list view.

    a. Create a new layout called "res/layout/event_detail.xml" for displaying the detail of an event item.

    Display data from the Event object. Following are some suggestions for the data to display.

    - Street
    - City
    - Description
    - Rating
    - Distance

    The TextView for description should be nested inside a "ScrollView" since this field may contain a large amount of text that would extend beyond the boundaries of the screen.

    The layout should look similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/event_street"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:padding="6dp" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/event_description"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="clip_vertical"
            android:layout_weight="1"
            android:gravity="top"
            android:padding="6dp" />

    </ScrollView>

</LinearLayout>
```

2. Create a new Activity for displaying the detail of an event item. The activity should be called "com.garagesalesapp.EventDetailActivity.java".
3. Implement the method "onCreate" and add code to the method to display the view.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.event_detail);
}
```

4. Add additional code to the "onCreate" method of "EventDetailActivity.java" to display the event item by using data passed with the intent. You will create the calling intent in a subsequent step.

```
// get references to the view elements in the detail layout
TextView street = (TextView) findViewById(R.id.event_street);
TextView description = (TextView) findViewById(R.id.event_description);
Intent intent = getIntent();
// Get the event data from the intent and update view
street.setText(intent.getStringExtra("street"));
description.setText(intent.getStringExtra("description"));
```

5. Add Activity to the Android Manifest. Open the file "AndroidManifest.xml" and add the following element to the application tag.

```
<activity
    android:name=".EventDetailActivity">
</activity>
```

6. Update "MainActivity.java" to call the activity for displaying the detail of an event item.

   There are a number of ways to listen for the click event and to start the new activity. It could be done within the view holder but that would couple the viewholder to the activity. More flexible would be to let the "MainActivity" decide what to do on the event which is the approach we will take.

   a. Declare an interface which will define the method to be called when an item on the RecyclerView is clicked.

      Add the following code to "SaleEventRecyclerAdapter" as an inner class

      ```
      public interface OnItemClickListener {
          void onItemClick(SaleEvent saleEvent);
      }
      ```

   b. Change the adapter to accept a listener

      Add the following code to "SaleEventRecyclerAdapter". Replace the existing constructor with this new constructor. Also add a reference to the listener as a instance variable.

```
OnItemClickListener listener;

public SaleEventRecyclerAdapter(Context context, List<SaleEvent> events,
OnItemClickListener listener) {
    this.mContext = context;
    this.events = events;
    this.listener = listener;
}
```

c. Create a listener for each data item by adding the following code to the end of the "onBindViewHolder" method.

```
holder.itemView.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
        SaleEvent saleEvent = events.get(position);
        listener.onItemClick(saleEvent);
        Log.d(TAG, "Creating listener for position: " + position);
    }
});
```

Is there a better place to create the "onClickListener"?

d. Modify the code in "MainActivity" to create the adapter. Pass a reference to the activity in the constructor. This will register the activity as a listener.

```
mAdapter = new SaleEventRecyclerAdapter(this, events, new SaleEventRecyclerAdapter
.OnItemClickListener() {
    @Override
    public void onItemClick(SaleEvent saleEvent) {
        Intent intent = new Intent(getBaseContext(), EventDetailActivity.class);
        intent.putExtra("street", saleEvent.getStreet());
        intent.putExtra("description", saleEvent.getDescription());
        startActivity(intent);
    }
});
```

Use code complete to generate the listener.

Lab 7.3 - Use Espresso to Test App

Overview

In this lab you will create Espresso tests to verify that the Add Event functionality is working properly.

Steps

1. Include the Espresso libraries in the dependencies section of the "build.gradle" file

```
dependencies {
    ...
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
}
```

It is always a good idea to test any "plumbing" before testing the UI. You may want to create a test to test "addEvent" directly. This is not an Espresso test but should be done before creating a UI test

1. Write an Espresso test to verify that the stand-alone "AddEventActivity" activity is working

```
@RunWith(AndroidJUnit4.class)
public class AddEventActivityTest {

    @Rule
    public ActivityTestRule<AddEventActivity> mActivityRule =
        new ActivityTestRule<>(AddEventActivity.class);

    @Test
    public void testAddEvent() {

        onView(withId(R.id.event_address)).perform(typeText("123 Main"),
closeSoftKeyboard());

        onView(withId(R.id.add_garage_sale)).perform(click());

        Context context = InstrumentationRegistry.getTargetContext();

        List<SaleEvent> events = SaleEventManager.getAllEvents(context);

        assertEquals(16, events.size());
    }
}
```

Advanced

1. Write an Espresso test to verify that the a new event conversation is working. This will require that you check the "ListView" in "MainActivity" to see if the event has been added in the UI.

Research "onData" to find item in ListView

Research UIAutomator to select menu item

1. Write a test to verify that a new activity is started. Use the espresso intent mechanism.

   https://google.github.io/android-testing-support-library/docs/espresso/intents/

Resources

Further info on Espresso is available here:

```
Using Espresso (and adapters)
http://developer.android.com/training/testing/ui-testing/espresso-testing.html
```

```
Select menu items
http://stackoverflow.com/questions/29556883/how-to-click-on-action-bar-items-when-
testing-with-android-espresso
```

```
Testing Samples from Android
https://github.com/googlesamples/android-testing
```

```
Android Testing Support Library official documentation
https://google.github.io/android-testing-support-library
```

```
Espresso Cheat Sheet
https://google.github.io/android-testing-support-library/docs/espresso/cheatsheet/
```

```
Espresso Idling Resources
http://dev.jimdo.com/2014/05/09/wait-for-it-a-deep-dive-into-espresso-s-idling-resources/
```

```
Espresso Custom Matchers
http://blog.xebia.com/android-custom-matchers-in-espresso/
```

```
Espresso Test Toolbar Title
http://blog.sqisland.com/2015/05/espresso-match-toolbar-title.html
```

```
RecyclerView example
https://github.com/Karumi/KataSuperHeroesAndroid
```

Lab 7.4 - Running Spoon

Overview

In this lab you will run the tests with Spoon which is a testing framework from Square that captures jUnit output and screen shots and displays them in a nice looking static web page.

Steps

1. Add the Spoon runner to "build.gradle" in the app module. Add it to the end of the file.

```
buildscript {
  repositories {
    mavenCentral()
  }

  dependencies {
    classpath 'com.stanfy.spoon:spoon-gradle-plugin:1.0.4'
  }
}

apply plugin: 'spoon'

// This section is optional
spoon {
  // for debug output
  debug = true

  // fix for multiple devices
  adbTimeout = 60 // 60 seconds

  // To run a single test class
  //className = 'fully.qualified.TestCase'

  // To run a single method in TestCase
  //methodName = 'testMyApp'
}
```

2. Run spoon

```
gradlew spoon
```

or use the following to run a single test class only

```
gradlew spoon -PspoonClassName=fully.qualified.TestCase
```

Update the build.gradle file with the following:

```
spoon {
  ...
  if (project.hasProperty('spoonClassName')) {
    className = project.spoonClassName
  }
}
```

View the output in a browser by opening "app/build/spoon/debug/index.html"

3. Display screenshots in test output using Spoon
   a. Add the Spoon client jar to "build.gradle"

```
dependencies {
  ...
  androidTestCompile 'com.squareup.spoon:spoon-client:1.3.1'
}
```

   b. Add read and write permissions to the AndroidManifest for the target app

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

   c. Modify the UI testing methods to show screen shots

```
Spoon.screenshot(activityRule.getActivity(), "screen_name")
```

The name of the screen shot can NOT contain spaces

Resources

Spoon https://github.com/square/spoon

Spoon plugin for Gradle https://github.com/stanfy/spoon-gradle-plugin

Problems writing screen image to sdcard. Apparently, you need to create a sdcard image file using mksdcard in order to have a writable sdcard.

+ https://github.com/square/spoon/issues/283

+ Make an sdcard image (NOT in shell)

+

```
mksdcard -l e 512M mysdcard.img
```

+ Create emulator with sdcard image

+

```
emulator -avd circleci-android22 -no-audio -no-window -sdcard mysdcard.img
```

Lab 8.1 - Fragments

Overview:

In this lab you will add fragments to activities. You will utilize the Android compatability package. The purpose of fragments is to allow UI components to be reused in multiple activities. One of the typical use cases is to use the same code to effectively display the UI on both small screens (where a single fragment will be shown at one time) and larger screens where multiple fragments will be shown at the same time. The Fragments API is Androids suggested technique for developing apps.

Setup

Complete prior labs.

Summary

Add fragments to the list and detail activities to fragments. This is a multistep process that involves the following individual steps.

- Convert the list activity (MainActivity) to an Activity and Fragment

— Create a new fragment for event list

— Modify the MainActivity to use the fragment

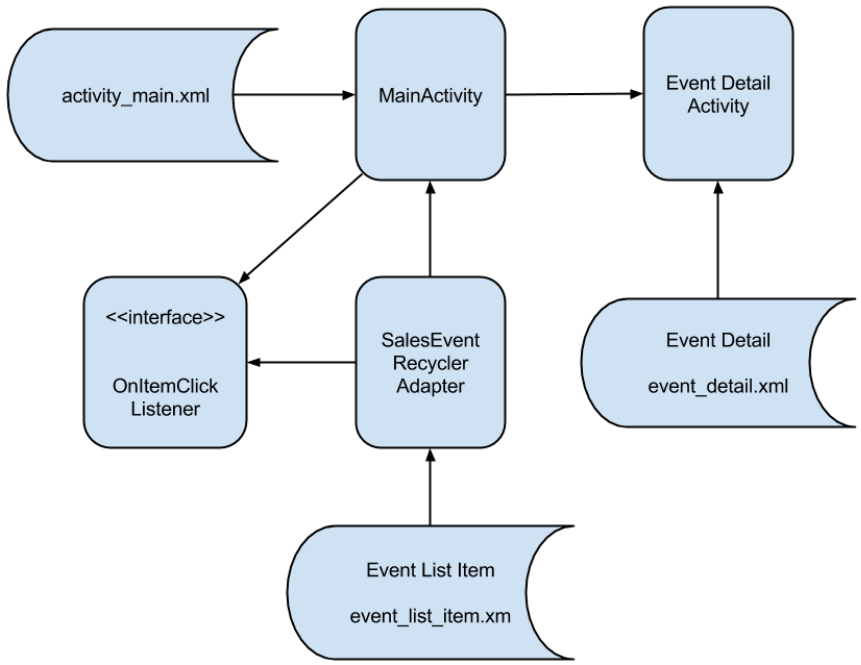
- Convert Detail activity (EventDetailActivity) to Activity and Fragment

— Create a new fragment for the event detail

— Modify the detail activity to use the fragment

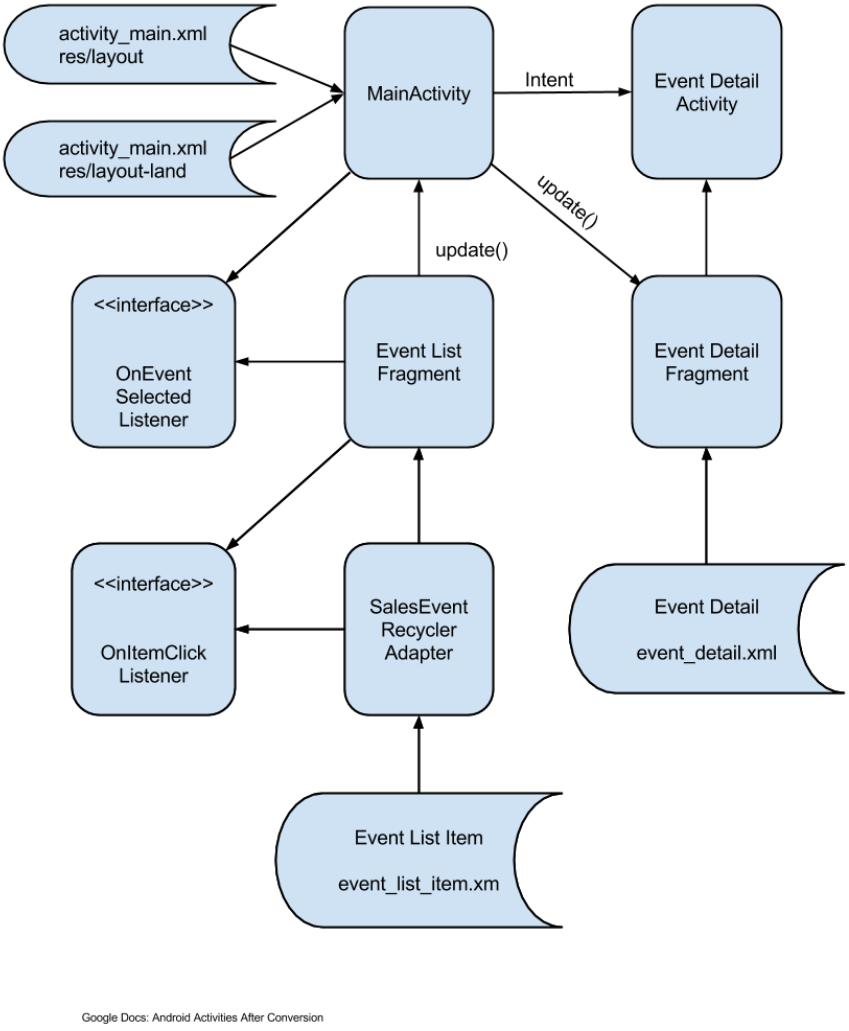
- Run new activities using fragments
- Create Fragment layout for multiple fragments
- Convert EventListActivity to call the detail fragment properly

  This diagram depicts the components and their relationship before the conversion:



Google Docs: Android Activities Before Conversion

This diagram depicts the components and their relationship after the conversion:



Google Docs: Android Activities After Conversion

Steps

1. Create a new fragment named EventListFragment by moving the UI logic from MainActivity
    a. Create a new fragment class called EventListFragment which extends android.support.v4.app.Fragment. If there are compiler errors for "Fragment" then the compatability library was probably not added properly.
    b. Move the displayRecyclerView method from MainActivity to EventListFragment. The reference to context will have to be changed since the code is no longer within an Activity class. Use the getActivity method inherited from Fragment to get the activity context.

```
private void displayRecyclerView(View view) {
    RecyclerView mRecyclerView = (RecyclerView) view.findViewById(R.id
.eventlistview);

    LinearLayoutManager mLinearLayoutManager =
        new LinearLayoutManager(getActivity(), LinearLayoutManager.VERTICAL, false
);
    mRecyclerView.setLayoutManager(mLinearLayoutManager);
    List<SaleEvent> events = SaleEventManager.getAllEvents(getActivity());

    SaleEventRecyclerAdapter mAdapter =
        new SaleEventRecyclerAdapter(this, events, new
              SaleEventRecyclerAdapter.OnItemClickListener() {
                  @Override
                  public void onItemClick(SaleEvent saleEvent) {
                      Intent intent =
                              new Intent(getBaseContext(), EventDetailActivity
.class);
                      intent.putExtra("street", saleEvent.getStreet());
                      intent.putExtra("description", saleEvent.getDescription());
                      intent.putExtra("event", saleEvent);
                      startActivity(intent);
                  }
              });

    mRecyclerView.setAdapter(mAdapter);
}
```

c. Override the onCreateView method to build the UI. There is no inherited setContentView method so you will need to inflate the layout by writing the code. Fortunately, you do get a reference to the inflater passed as an argument.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {

    RelativeLayout view =
            (RelativeLayout) inflater.inflate(R.layout.event_list, container,
false);

    displayRecyclerView(view);

    return view;
}
```

You will receive an error in the code due to an invalid reference to the "event_list" layout.

d. Copy "res/layout/activity_main.xml" to "list_event.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.garagesalesapp.MainActivity">

    <android.support.v7.widget.RecyclerView
        android:id="@+id/eventlistview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

2. Modify the MainActivity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

a. Create a new layout for the activity which will instantiate the fragment. Modify the existing "activity_main.xml" in the "res/layout" directory. The layout contains an element to declare the fragment. The fragment will be created when an activity using this layout inflates the layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <fragment
        android:id="@+id/list_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:name="com.garagesalesapp.EventListFragment" >
    </fragment>
</LinearLayout>
```

All the activity does is inflate a layout which builds the fragment. The UI code is in the fragment.

Remove any UI code that has been moved to the fragment.

3. Create a new fragment named EventDetailFragment by moving the UI logic from EventDetailActivity
   a. Create a new fragment class called EventDetailFragment which extends android.support.v4.app.Fragment.
   b. Move UI code from the activity to the fragment

```
TextView street;
TextView description;
CardView view;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

    CardView view = (CardView) inflater.inflate(R.layout.event_detail, container,
false);

    // get references to the view elements in the detail layout
    street = (TextView) view.findViewById(R.id.event_street);
    description = (TextView) view.findViewById(R.id.event_description);

    Intent intent = getActivity().getIntent();
    // Get the event data from the intent and update view
    street.setText(intent.getStringExtra("street"));
    description.setText(intent.getStringExtra("description"));

    return view;
}
```

4. Modify the "EventDetailActivity" to create the fragment by using the FragmentManager

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    EventDetailFragment fragment = new EventDetailFragment();
    FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();
    transaction.add(android.R.id.content, fragment);
    transaction.commit();
}
```

All the activity does is build the fragment.

Remove any unnecessary UI code from the activity.

5. Create a new layout for showing both a list and detail fragment when there is room on the screen.
   a. Create a layout called "activity_main.xml" and place it in a new directory called "res/layout-land". This will be used when the device is in landscape orientation.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <fragment
        android:id="@+id/list_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:name="com.garagesalesapp.EventListFragment" />
    <fragment
        android:id="@+id/detail_fragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:name="com.garagesalesapp.EventDetailFragment" />

</LinearLayout>
```

6. Modify "EventListActivity" to display the event detail when an item is selected in the list.
   a. Add a method to the detail fragment "EventDetailFragment" to display a specific event.

```
public void update(SaleEvent event) {
    street.setText(event.getStreet());
    description.setText(event.getDescription());
}
```

7. Modify "EventListFragment" to listen to onClick events in the RecyclerView and notify listeners (activities) when those events occur.
   a. Add listener interface to the list fragment "EventListFragment".

```
public interface OnEventSelectedListener {
    public void onEventSelected(SaleEvent event);
}
```

b. Create code for registering a listener in "EventListFragment"

```
private OnEventSelectedListener listener;

@Override
public void onAttach(Context context) {
    super.onAttach(context);

    try {
        listener = (OnEventSelectedListener) context;
    } catch (ClassCastException e) {
        throw new ClassCastException(context.toString() +
                " must implement OnEventSelectedListener");
    }
}
```

c. Assign the listener to the RecyclerView

```
SaleEventRecyclerAdapter mAdapter =
        new SaleEventRecyclerAdapter(getActivity(), events, this);
```

d. Call the onEventSelected method when the onItemClick listener is called

```
@Override
public void onItemClick(SaleEvent saleEvent) {
    listener.onEventSelected(saleEvent);
}
```

8. When an item is selected from the list, communicate through the activity to display the item. In "MainActivity" implement the "OnEventSelectedListener" and implement the "onEventSelected" method:

```
    @Override
    public void onEventSelected(SaleEvent event) {
        Log.d(TAG, "Running eventSelected listener in activity_main activity");
        EventListFragment eventListFragment = (EventListFragment)
                getSupportFragmentManager().findFragmentById(R.id.list_fragment);
        EventDetailFragment viewer = (EventDetailFragment)
                getSupportFragmentManager().findFragmentById(R.id.detail_fragment);
        if (viewer == null || !viewer.isInLayout()) {
            Intent intent = new Intent(this, EventDetailActivity.class);
            intent.putExtra("street", event.getStreet());
            intent.putExtra("description", event.getDescription());
            startActivity(intent);
        } else {
            viewer.update(event);
        }
    }
```

Run the application in portrait mode. Only the list should display. Switch the device to landscape mode. The list and detail fragments should appear at the same time.

Optional

When the screen orientation changes the performance is a little slow. Can this be improved?

See the following link to explore one technique for this:

http://android-developers.blogspot.com/2009/02/faster-screen-orientation-change.html

References

http://android-developers.blogspot.com/2011/02/android-30-fragments-api.html

http://android-developers.blogspot.com/2011/09/preparing-for-handsets.html

Lab 9.1 - Local Storage

Overview:

In this lab you will save data to local storage. You will save a version of an new SaleEvent object as a text file in local Android storage.

Steps

1. The lab provides code for the methods necessary to perform the object serialization. Check "FileUtils.java" in the library for the following method:

   - writeFile

     Additionally there is code for serializing the SaleEvent object as XML. Review these methods.

   - writeXML
   - serializeTag

     Add the following code to "AddEventActivity" in the "onClick" method of the "onClickListener" before the "finish()" call.

     ```
     writeFile( AddEventActivity.this, event.getId()+".txt", event.toString() );
     ```

     There is one major different between the Android code and typical Java File IO code. The Android code returns a FileOutputStream object by calling the "openFileOutput" method on the Android app context. The problem for this method is to get the context.

     Run the application and add a new event. You should see the new event in the list view but how do you verify that it has been written to the Android file system?

2. Verify that the file has been written. Review the logging but also check the file system for the file. You can see the file by using the Android command line shell "adb".

   a. From the terminal, type the following:

      ```
      adb root
      adb shell
      cd /data/data/com.garagesalesapp/files
      cat *.txt
      ```

      You can learn more about using adb at the following URL:

http://developer.android.com/guide/developing/tools/adb.html

Lab 9.2.1 - Preferences

Overview:

In this lab you create an activity for allowing the user to enter application preferences.

Steps

1. Create a new preference layout. The layout will allow the user to enter their user name. For now, you don't need to validate the name.

   a. Create a new folder called "xml" in the "res" directory. Create a new xml file called "preferences.xml" in "res/xml". It should look like the following:

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <PreferenceScreen
     xmlns:android="http://schemas.android.com/apk/res/android" >
     <EditTextPreference
       android:key="PREF_USERNAME"
       android:summary="Enter your user name"
       android:title="Username" />
   </PreferenceScreen>
   ```

   Note: Are there any strings in the above code that you could externalize in "strings.xml"?

   Optional: Add some additional preference fields. There are many different kinds of preference fields. See the direct and indirect subclass of "Preference" to see them all. You may want to research the following links:

   http://developer.android.com/reference/android/preference/Preference.html

   http://developer.android.com/resources/samples/ApiDemos/res/xml/preferences.html

2. Create new preference activity. This activity will display the preferences and allow the user to change them.

   a. Create a new class in the package "com.garagesalesapp" called "SettingsActivity.java" and make it a subclass of "PreferenceActivity". Register the preference layout by calling the "addPreferenceFromResource" method with the id for your preference layout.

```
public class SettingsActivity extends PreferenceActivity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
  }

}
```

3. Add the preference activity to the manifest.
   a. Open the file "AndroidManifest.xml" and add the following element to the application tag.

```
<activity
  android:name=".SettingsActivity">
</activity>
```

4. Change the menu in the MainActivity so that the preference activity is started when the user clicks the "Prefs" menu. Item.
   a. Add the following code to the "showPrefs" method in the "MainActivity.java" file.

```
Intent intent = new Intent(this, SettingsActivity.class);
startActivity(intent);
```

   Run the application. Press the Menu button and select "Settings". The settings activity should appear.

5. Now that you have allowed the user to save a preference value, you should use it in the application. One way of using it would be to display the user name in the MainActivity.
   a. Add a new TextView element to the layout for the MainActivity in "res/layout/activity_main.xml".

```
<TextView
    android:id="@+id/userName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

   b. In "MainActivity.java" find the value for the user name and display it. Add the following code to the top of the "displayRecyclerView" method.

```
SharedPreferences prefs =
  PreferenceManager.getDefaultSharedPreferences(this);

String defaultValue="";
String userName = prefs.getString("PREF_USERNAME", defaultValue).trim();

TextView userNameView = (TextView) findViewById(R.id.userName);

if (userName.equals("")) {
    userNameView.setVisibility(View.INVISIBLE);
} else {
    userNameView.setVisibility(View.VISIBLE);
    userNameView.setText(userName);
}
```

Lab 9.2.2 – Preference Fragments

Overview:

In this lab you will convert the Settings activity to a Fragment.

Setup:

Prior labs must be completed.

Steps

1. Change SettingsActivity to extend AppCompatActivity
2. Create a new SettingsFragment class which extends PreferenceFragment

```
public static class SettingsFragment extends PreferenceFragment {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Load the preferences from an XML resource
    addPreferencesFromResource(R.xml.preferences);
  }
}
```

   a. Create the fragment in the "onCreate" method (replacing the "addPreferencesFromResource" method)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  // Display the fragment as the main content.
  FragmentManager mFragmentManager = getFragmentManager();
  FragmentTransaction mFragmentTransaction =
      mFragmentManager.beginTransaction();
  SettingsFragment mSettingsFragment = new SettingsFragment();
  mFragmentTransaction.replace(android.R.id.content, mSettingsFragment);
  mFragmentTransaction.commit();
}
```

Lab 9.2.3 - Testing Preferences

Overview:

In this lab you create a test for Preferences settings.

Steps

1. Create a Preference wrapper

```
public class Preferences {

    private static final String NOTIFICATION = "NOTIFICATION";
    private static final String USERNAME = "USERNAME";

    private final Context context;

    public Preferences(Context context) {
        this.context = context;
    }

    public String getUsername() {
        return getPreferences().getString(USERNAME, null);
    }

    public void setUsername(String username) {
        getPreferences().edit().
                        putString(USERNAME, username).
                        apply();
    }

    public boolean hasNotificationEnabled() {
        return getPreferences().getBoolean(NOTIFICATION, false);
    }

    public void setNotificationEnabled(boolean enable) {
        getPreferences().edit().
                         putBoolean(NOTIFICATION, enable).
                         apply();
    }

    private SharedPreferences getPreferences() {
        return context.getSharedPreferences("user_prefs", MODE_PRIVATE);
    }
}
```

2. Create a Preference test

```
@RunWith(RobolectricTestRunner.class)
@Config(manifest = Config.NONE)
public final class PreferencesTest {

    private Preferences preferences;

    @Before
    public void setUp() {
        preferences = new Preferences(RuntimeEnvironment.application);
    }

    @Test
    public void should_set_username() {
        preferences.setUsername("jmartinez");
        assertThat(preferences.getUsername()).isEqualTo("jmartinez");
    }

    @Test
    public void should_set_notification() {
        preferences.setNotificationEnabled(true);
        assertThat(preferences.hasNotificationEnabled()).isTrue();
    }

    @Test
    public void should_match_defaults() {
        assertThat(preferences.getUsername()).isNull();
        assertThat(preferences.hasNotificationEnabled()).isFalse();
    }
}
```

Resources

http://jeremie-martinez.com/2016/02/16/unit-tests/

Lab 9.3.1 - Database

Overview:

In this lab you will create a database that the SaleEventManager will use to persist the event objects on the device.

Steps:

1. Create a new class that extends SQLiteOpenHelper which will be used to create and upgrade the database.
   a. In the garagesales-library, create a new package called "com.garagesaleslibrary.event.database".
   b. Create a new class in the package called "GarageDbHelper" and make it a subclass of "SQLiteOpenHelper".

      You will receive errors because of missing constructors. Generate the constructors from the super class.

   c. Write the database create method. This method will run the DDL to create a new database.

      Declare the "create" statement as a string constant. Add this code to the "GarageDbHelper.java" file.

```
private static final String DATABASE_CREATE =
    "create table events (" +
    "_id integer primary key autoincrement, " +
    "id text, " +
    "date date, " +
    "title text, " +
    "street text, " +
    "city text, " +
    "state text, " +
    "zip text, " +
    "latitude double, " +
    "longitude double, " +
    "description text, " +
    "rating double, " +
    "distance double " +
");";
```

If you are using cut and paste, verify that the "+" signs copied properly.

d. Create the method which will run the DDL create statement.

   In "GarageDbHelper.java" add the following method:

```
@Override
public void onCreate(SQLiteDatabase db) {
    Log.v("GarageDbHelper", "Create Table using " + DATABASE_CREATE);
    db.execSQL(DATABASE_CREATE);
}
```

   Note: This method uses the string you've already created and writes to the log so you can verify that the DDL runs. This method is called automatically by Android when it detects that the database must be created.

e. Write the database upgrade method.

   This method is run when Android detects that the database version has changed. This method will alter the database. It may do so by dropping the old database and creating a new run or by executing alter statements to upgrade the database in place. Altering the database is more complex since it must be determined which version the existing database is.

   We will chose a simple strategy. When the database version changes, drop the table if it exists and create a new table.

   In "GarageDbHelper.java" and the following code:

```
private static final String DATABASE_DROP =
    "drop table if exists events";

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.v("GarageDbHelper", "Drop Table, Upgrading from " +oldVersion + " to " +
newVersion);
    db.execSQL(DATABASE_DROP);
    db.execSQL(DATABASE_CREATE);
}
```

2. In the garagesales-library, create a new DbAdapter class to provide access to the database.

   This class does not extend any existing Android class but its use is part of the recommended best practices for working with databases in Android. This is the class that will be used by your application code to provide access to the database. You shouldn't access the database classes directly.

   a. Create a new class in package "com.garagesalesapp.event.database" called "GarageDbAdapter"

3. Provide DbAdapter with an implementation of the DbHelper

    a. Add the following code to "GarageDbAdapter.java". This code uses "composition" to provide an implementation of "GarageDbHelp" inside the adapter. The constructor creates the instance of "GarageDbHelper".

    ```
    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "garage.db";
    private GarageDbHelper dbHelper;

    public GarageDbAdapter(Context context) {
        dbHelper = new GarageDbHelper(
        context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    ```

    Note: Notice that the constructor requires a context object.

    b. In "GarageDbAdapter.java" Create methods for opening and closing the database. These are just delegates to the real method in the helper class.

    ```
    public SQLiteDatabase db;

    public GarageDbAdapter open() {
        db = dbHelper.getWritableDatabase();
        return this;
    }
    public void close() {
        dbHelper.close();
    }
    ```

4. Instantiate the "GarageDbAdapter" class.

    Because this class requires a context object when it is created, we must find a place in our code that has access to the context and will be executed when the application starts up. Fortunately, Android provides the ideal place for this - the application object.

    a. Create a new application object called "GarageApplication.java" in package "com.garagesalesapp", by extending the "Application" class.

    b. Register the application object in the Android Manifest by assigning the "name" attribute of the "application" tag to "GarageApplication"

    c. Add a static method to "GarageApplication" to hold a reference to the adapter

    ```
    private static GarageDbAdapter dbAdapter; .
    ```

d. Create the adapter in the "onCreate" method (after getting the application context) and open the database.

```
GarageApplication.dbAdapter = new GarageDbAdapter(getApplicationContext());

GarageApplication.dbAdapter.open();
```

e. Add a static method to return a reference to the adapter.

```
public static GarageDbAdapter getDbAdapter() {
    return dbAdapter;
}
```

5. Run and test.

a. Review log cat for logging from the creation methods. Find the DDL for creating the database.

b. Verify that the database file has been created by using the File Explorer in DDMS. Look for the file "garage.db" in the "/data/data/com.garagesalesapp/databases" directory.

Note: If you need to re-run the test, be sure to increment the version number in "GarageDbAdapter" to drop and re-create the database.

6. Write code to query the database. A database query is returned as a "Cursor". Results must be extracted from the cursor. This is similar in concept to "ResultSet" in JDBC but the API is not the same.

a. Create the query method in the adapter "GarageDbAdapter.java"

```
private Cursor getAllEntries() {
    String[] columns = new String[10];
    columns[0] = "id";
    columns[1] = "date";
    columns[2] = "title";
    columns[3] = "street";
    columns[4] = "city";
    columns[5] = "rating";
    columns[6] = "distance";
    columns[7] = "description";
    columns[8] = "latitude";
    columns[9] = "longitude";
    return db.query("events", columns, null, null, null, null, null);
}
```

Note: Review the javadoc for the query method to understand the null parameters.

b. The application should not have to interact direction with the Database API so put a wrapper around the "getAllEntries" method to return the data as event objects. The "getAllEntries" method can't be called from outside the adapter since it is private.

   Add the following method to "GarageDbAdapter".

```
public List<SaleEvent> getAllSaleEvents() {
    ArrayList<SaleEvent> events = new ArrayList<SaleEvent>();
    Cursor cursor = getAllEntries();
    if (cursor.moveToFirst()) {
        do {
            SaleEvent event = new SaleEvent();
            event.setId(cursor.getString(0));
            event.setDate( new java.util.Date( cursor.getLong(1)) );
            event.setTitle(cursor.getString(2));
            event.setStreet(cursor.getString(3));
            event.setCity(cursor.getString(4));
            event.setRating(Float.parseFloat(cursor.getString(5)));
            event.setDistance(Double.parseDouble(cursor.getString(6)));
            event.setDescription(cursor.getString(7));
            event.setLatitude(cursor.getDouble(8));
            event.setLongitude(cursor.getDouble(9));
            events.add(event);
            Log.v("GarageDbAdapter", "Get event: id="+event.getId());
        } while (cursor.moveToNext());
    }
    return events;
}
```

7. Create a method for adding data to the database.

   a. Create a new method called "insertSaleEvent" in "GarageDbAdapter". It should take a SaleEvent object as a parameter.

```
public long insertSaleEvent(SaleEvent event) {
    ContentValues values = new ContentValues();
    values.put("id", event.getId());
    values.put("date", Long.toString(event.getDate().getTime()) );
    values.put("title", event.getTitle());
    values.put("street", event.getStreet());
    values.put("city", event.getCity());
    values.put("rating", event.getRating());
    values.put("distance", event.getDistance());
    values.put("description", event.getDescription());
    values.put("latitude", event.getLatitude());
    values.put("longitude", event.getLongitude());
    Log.v("GarageDbAdapter", "Insert event:id="+event.getId());
    return db.insert("events", null, values);
}
```

This code creates a new "android.content.ContentValues" which is simply a map holding the name/value pairs for the columns in the database. The "insert" method takes the map and adds it as a new row to the specified table. Notice that the application doesn't need to work with "ContentValues" only events.

Note: Be sure to review the javadoc for the "insert" method.

8. Optional: We have not implemented all the CRUD method. If you have time, implement the following additional methods.
   - deleteSaleEvent(String eventId)
   - getSaleEvent(String eventId)
   - updateSaleEvent(SaleEvent event)

   Be sure to create unit tests to verify that your new methods are working correctly.

9. Change the "SaleEventManager" to use the database rather than a "List" for persisting the event data. By using a database, we can make the new event data persist when we re-run the app.
   a. Modify the "getAllEvents" method in "com.garagesaleslibrary.event.service.SaleEventManager.java" to put items in the database and lookup items from the database. Add the following code to the end of the "getAllSaleEvents" method.

```
public static List<SaleEvent> getAllEvents(Context context) {

    Log.d(TAG, "running getAllEvents");

    GarageDbAdapter adapter = new GarageDbAdapter(context);
    return adapter.open().getAllSaleEvents();
}
```

b. Modify the "addEvent" method to use the database

```
public static void addEvent(Context context, SaleEvent event) {

    Log.d(TAG, "running addEvent");

    event.setId(java.util.UUID.randomUUID().toString());
    event.setDate(new java.util.Date());

    GarageDbAdapter adapter = new GarageDbAdapter(context);
    adapter.open().insertSaleEvent(event);
}
```

c. The database needs to be populated when it is created. Update the "onCreate" method in "GarageDbHelper"

```
@Override
public void onCreate(SQLiteDatabase db) {
    Log.v(TAG, "Create Table using: " + DATABASE_CREATE);
    db.execSQL(DATABASE_CREATE);

    populateDatabase(db);
}

private void populateDatabase(SQLiteDatabase db) {
    List<SaleEvent> events = SaleEventManager.parseSaleEventXmlFile(context);
    for (SaleEvent event: events) {
        ContentValues values = new ContentValues();

        values.put("id", event.getId());
        if (event.getDate() != null) {
            values.put("date", Long.toString(event.getDate().getTime()) );
        }
        values.put("title", event.getTitle());
        values.put("street", event.getStreet());
        values.put("city", event.getCity());
        values.put("rating", event.getRating());
        values.put("distance", event.getDistance());
        values.put("description", event.getDescription());
        values.put("latitude", event.getLatitude());
        values.put("longitude", event.getLongitude());

        Log.v(TAG, "Insert event:id="+event.getId());

        db.insert("events", null, values);
    }
}
```

10. Open the database and run SQL against it. You can find out how to use the tools at the following URLs.

    http://developer.android.com/guide/developing/tools/adb.html

    http://developer.android.com/guide/developing/tools/sqlite3.html

    a. Change to the tools directory. The tools are in "<sdk_home>/platform-tools" This is only necessary if you have not already added the tools to the path.

    b. Start the adb shell

    ```
    adb shell
    ```

You should see a "\#" prompt on the command line.

Note: Mac users should run the following command (in the correct directory) instead:

```
./adb shell
```

c. Start the sqlite admin shell

```
sqlite3 /data/data/com.garagesalesapp/databases/garage.db
```

You should see a "sqlite>" prompt on the command line.

d. Verify that the data is in the database

```
.schema events
select * from events;
```

e. Exit both shells.

```
.exit
exit
```

Lab 9.3.1 - Database (using Room)

Overview:

In this lab you will create a database that the SaleEventManager will use to persist the event objects on the device.

Steps:

1. Add the Google Maven repo to the repositories. This is where the persistence libraries are located.

   <projectDir>/gradle.build

   ```
   allprojects {
       repositories {
           jcenter()
           maven {
               url "https://maven.google.com"
           }
       }
   }
   ```

2. Add the persistence library dependencies to the library project.

   <projectDir/garagesales-library/gradle.build

   ```
   dependencies {

       compile 'android.arch.persistence.room:runtime:1.0.0-alpha1'
       annotationProcessor 'android.arch.persistence.room:compiler:1.0.0-alpha1'

   }
   ```

3. Add persistence annotations to the SaleEvent domain object

<projectDir/garagesales-library/domain/SaleEvent.java

```
@Entity(tableName = "sale_event")
public class SaleEvent implements Serializable {

  @PrimaryKey private String id;
  @Ignore private Date date;
  @ColumnInfo private String title;
  @ColumnInfo private String street;
  @ColumnInfo private String city;
  @ColumnInfo private String state;
  @ColumnInfo private String zip;
  @ColumnInfo private double latitude;
  @ColumnInfo private double longitude;
  @ColumnInfo private String description;
  @ColumnInfo private float rating;
  @ColumnInfo private double distance;
  @ColumnInfo private String imageFileName;
```

4. Create the DAO object for SaleEvent
   a. Create a new package in the library named "database"
   b. In the "database" package create a new interface named "SaleEventDao"

   <projectDir/garagesales-library/database/SaleEventDao.java

```
@Dao
public interface SaleEventDao {

    @Query("SELECT * FROM sale_event")
    List<SaleEvent> getAll();

    @Insert
    void insertAll(SaleEvent... saleEvent);
}
```

5. In the "database" package create a new abstract class named "AppDatabase"

<projectDir/garagesales-library/database/AppDatabase.java

```
@Database(entities = {SaleEvent.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {

    public abstract SaleEventDao saleEventDao();

}
```

6. In the app project, create an application object to build the database object.

   <projectDir/app/src/main/java/com/garagesalesapp/GarageApplication.java

```
public class GarageApplication extends Application {

    private static AppDatabase db;

    @Override
    public void onCreate() {
        super.onCreate();

        db = Room.databaseBuilder(getApplicationContext(), AppDatabase.class,
"garage.db")
                .allowMainThreadQueries()
                .build();
    }

    public static AppDatabase getDb() {
        return db;
    }
}
```

7. Register the application object in the Android manifest.

   <projectDir/app/src/main/AndroidManifest.xml

```
android:name=".GarageApplication"
```

8. In the app project, replace the code to retrieve the events.

   <projectDir/app/src/main/java/com/garagesalesapp/MainActivity.java

```
List<SaleEvent> events = GarageApplication.getDb().saleEventDao().getAll();
```

9. In the app project, replace the code to save the event.

<projectDir/app/src/main/java/com/garagesalesapp/AddEventActivity.java

```
GarageApplication.getDb().saleEventDao().insertAll(event);
```

Optional

1. Open the database and run SQL against it.
   a. Change to the tools directory. The tools are in "<sdk_home>/platform-tools" This is only necessary if you have not already added the tools to the path.
   b. Start the adb shell

      ```
      ./adb root
      ./adb shell
      ```

      You should see a "\#" prompt on the command line.

   c. Start the sqlite admin shell

      ```
      sqlite3 /data/data/com.garagesalesapp/databases/garage.db
      ```

      You should see a "sqlite>" prompt on the command line.

   d. Verify that the data is in the database

      ```
      .schema events
      select * from sale_event;
      ```

   e. Exit both shells.

      ```
      .exit
      exit
      ```

Resources

You can find out more about how to use the tools at the following URLs.

+ http://developer.android.com/guide/developing/tools/adb.html

+ http://developer.android.com/guide/developing/tools/sqlite3.html

Lab 9.3.2 – Testing the database

Overview

In this lab you will write a unit test for the database

Setup

Prior labs must be completed.

Steps

1. Create a new Android test called DatabaseTest. This should be an instrumentation test using jUnit4.
2. Test the insert method of GarageDbAdapter

```
package com.garagesalesapp;

@RunWith(AndroidJUnit4.class)
public class DatabaseTest {

    @Test
    public void testInsertEvent() {
        Context context = InstrumentationRegistry.getTargetContext();

        GarageDbAdapter adapter = new GarageDbAdapter(context);

        SaleEvent event = new SaleEvent();
        event.setId("1001");
        event.setStreet("123 Main Street");
        adapter.open().insertSaleEvent(event);

        SaleEvent returnedEvent = adapter.open().getEvent("1001");
        assertEquals("123 Main Street", returnedEvent.getStreet());
    }
}
```

Lab 10.1 – Creating a ContentProvider

Overview

In this lab you will build a ContentProvider to expose Sale Events. You will also create a new app which will use the ContentProvider.

Setup

Prior labs must be completed.

Steps

1. Create ContentProvider

   Use the following resource from Android to develop the Provider

   http://developer.android.com/intl/zh-CN/guide/topics/providers/content-provider-creating.html

   a. Create subclass of ContentProvider called "GarageContentProvider"
   b. Add constructors and stub methods
   c. Implement getType
   d. Implement insert
   e. Implement query
   f. Implement update
   g. Implement delete
   h. Add provider to manifest

2. You should create your class using Android Developer resources and your solution will be unique to you. However, following is sample code that you may use.

```
package com.garagesalesapp.event.database;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
```

```
import android.net.Uri;
import android.text.TextUtils;
import android.util.Log;

public class GarageContentProvider extends ContentProvider {

    private static final String TAG = GarageContentProvider.class.getSimpleName();

    private static final UriMatcher sURIMatcher = new UriMatcher(
            UriMatcher.NO_MATCH);

    static {
        sURIMatcher.addURI(GarageContract.AUTHORITY, GarageContract.TABLE,
                GarageContract.EVENT_DIR);
        sURIMatcher.addURI(GarageContract.AUTHORITY, GarageContract.TABLE
                + "/#", GarageContract.EVENT_ITEM);
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // Implement this to handle requests to delete one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public String getType(Uri uri) {
        // TODO: Implement this to handle requests for the MIME type of the data
        // at the given URI.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO: Implement this to handle requests to insert a new row.
        //throw new UnsupportedOperationException("Not yet implemented");

        GarageDbAdapter dbAdapter = new GarageDbAdapter(getContext());
        SQLiteDatabase db = dbAdapter.open().db;

        long rowId = db.insert("events", null, values);

        if (rowId == -1) {
            Log.d(TAG, "Failed to insert " + values + " to " + uri);
            return null;
        } else {
            Uri itemUri = ContentUris.withAppendedId(uri, rowId);
            // Notify listeners of data changes
            super.getContext().getContentResolver().notifyChange(itemUri, null);
```

```
            return itemUri;
        }

    }

    @Override
    public boolean onCreate() {
        // TODO: Implement this to initialize your content provider on startup.
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {

        GarageDbAdapter dbAdapter = new GarageDbAdapter(getContext());
        SQLiteDatabase db = dbAdapter.open().db;

        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
        qb.setTables(GarageContract.TABLE);

        switch (sURIMatcher.match(uri)) {
            case GarageContract.EVENT_DIR:
                break;
            case GarageContract.EVENT_ITEM:
                qb.appendWhere(GarageContract.Column.ID + "="
                        + uri.getLastPathSegment());
                break;
            default:
                throw new IllegalArgumentException("Illegal uri: " + uri);
        }

        String orderBy = (TextUtils.isEmpty(sortOrder)) ? GarageContract.DEFAULT_SORT
                : sortOrder;

        Cursor cursor = qb.query(db, projection, selection, selectionArgs, null, null,
orderBy);

        Log.d(TAG, "queried records: " + cursor.getCount());
        return cursor;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        // TODO: Implement this to handle requests to update one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }
```

```
}
```

3. Content Providers require lots of string values when configuring. Rather that use these string values directly, it is better to declare them as constants. A useful design approach is to assign all the constants in a special class. You should create a class called "GarageContract" to contain the constants.

```
public class GarageContract {

    // DB specific constants
    public static final String DB_NAME = "garage.db";
    public static final int DB_VERSION = 1;
    public static final String TABLE = "events";

    // Provider specific constants
    // content://com.garagesaleslibrary.event.EventProvider/events
    public static final String AUTHORITY = "
com.garagesaleslibrary.event.EventProvider";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/" +
TABLE);
    public static final int EVENT_ITEM = 1;
    public static final int EVENT_DIR = 2;
    public static final String EVENT_TYPE_ITEM =
"vnd.android.cursor.item/vnd.com.garage.provider.event";
    public static final String EVENT_TYPE_DIR =
"vnd.android.cursor.dir/vnd.com.garage.provider.event";
    public static final String DEFAULT_SORT = Column.DISTANCE + " DESC";  //
Descending

    public class Column {
        public static final String ID = BaseColumns._ID;  // "_id"
        //public static final String ID = "id";
        public static final String DATE = "date";
        public static final String TITLE = "title";
        public static final String STREET = "street";
        public static final String CITY = "city";
        public static final String RATING = "rating";
        public static final String DISTANCE = "distance";
        public static final String DESCRIPTION = "description";
        public static final String LATITUDE = "latitude";
        public static final String LONGITUDE = "longitude";

    }
}
```

4. Add the content provider to the "AndroidManifest.xml" file

You will test the Content Provider but we will not use it in the app.

Lab 10.2 – Testing a ContentProvider

Overview

In this lab you will test the Content Provider.

Setup

Prior labs must be completed.

Use the following resource from Android to develop the Provider

http://developer.android.com/intl/zh-CN/tools/testing/contentprovider_testing.html

Steps

1. Create a new test called ContentProviderTest which extends ProviderTestCase2. This should be a jUnit3 test class.
2. Assign the provider
3. Write an insert test
4. Write a query test
5. You should create your class using Android Developer resources and your solution will be unique to you. However, following is sample code that you may use.

```
public class ContentProviderTest extends ProviderTestCase2<GarageContentProvider> {

    public ContentProviderTest() {
        this(GarageContentProvider.class, GarageContract.AUTHORITY);
    }

    public ContentProviderTest(Class<GarageContentProvider> providerClass, String
providerAuthority) {
        super(providerClass, providerAuthority);
    }

    @Override
    protected void setUp() throws Exception {
        super.setUp();
        getMockContentResolver().addProvider(GarageContract.AUTHORITY, getProvider());
    }

    public void testInsertSaleEvent() {

        ContentValues values = new ContentValues();
        values.put("title", "New garage sale");

        Uri uri = getMockContentResolver().insert(GarageContract.CONTENT_URI, values);

        Cursor cursor = getMockContentResolver().query(uri, null, null, null, null);
        assertEquals(1, cursor.getCount());
        // move to first row
        cursor.moveToNext();
        assertEquals("New garage sale", cursor.getString(cursor.getColumnIndex("title
")));
    }

}
```

Lab 11.1 - Asynchronous Tasks

Overview:

In this lab you will turn a long running process on the UI thread into a separate asynchronous tasks that will still be able to communicate with the UI thread.

Steps

1. Add a new menu item to the main menu called "Refresh" which will get the events from the SaleEventManager and display them in the main activity.
   a. Add the new menu item to "res/menu/main_menu.xml" .

   ```
   <item
       android:id="@+id/mi_refresh"
       android:title="@string/mi_refresh"/>
   ```

   Also add the new string to "res/values/strings.xml"

   ```
   <string name="mi_refresh">Refresh</string>
   ```

   b. In "MainActivity" handle the new menu selection and create a new method called "showRefresh" which will be called when the user selects the "Refresh" menu item.

   Add the following code to the "onOptionsItemSelected" method

   ```
   case R.id.mi_refresh:
       refresh();
       return true;
   ```

   In the "showRefresh" method, call "displayRecyclerView"

   ```
    private void refresh() {
       Log.v(TAG, "Running refresh method.");
       displayRecyclerView();
    }
   ```

   Note: the "displayRecyclerView" method calls "SaleEventManager.getAllSaleEvents()" which returns the list of events.

Run the application to verify that the events are being re-displayed when the "Refresh" menu option is selected.

2. Increase the execution time of the SaleEventManager method "addEvent" until it forces the UI activity to be stopped by the Android OS

   a. Add the following code to the top of the "getAllEvents" method in the "SaleEventManager" class:

```
try {
    Log.v("SaleEventManager", "Thread sleep");
    Thread.sleep(10000);
    Log.v("SaleEventManager", "Thread awake");
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

   This will cause the UI thread to sleep for 10 seconds. It is simulating the effect of making a very long call to the server. The UI will either now crash or hang. This is a problem since the UI should be very responsive. Notice that the UI freezes after the user presses the "Refresh" button on the menu.

3. Change the "displayRecyclerView()" method by removing the call to "getAllSaleEvents". Instead, pass the events in as an argument

```
public void displayRecyclerView(List<SaleEvent> events) {...}
```

4. Change the "refresh()" method to create the AsyncTasks

```
private void refresh() {
   Log.v(TAG, "Running refresh method.");
   GetAllEventsTask task = new GetAllEventsTask(this);
   task.execute();
}
```

There will be compiler errors since the "GetAllEventsTask" class does not yet exist. Use code complete in the IDE to create it.

5. Change the "onResume" method to call "refresh()" instead of "displayRecyclerView()".

```
@Override
protected void onResume() {
    super.onResume();
    refresh();
}
```

6. Create the asynchronous task which will be run from the main UI thread. Create a new class in the "src/com.garagesalesapp" package called "GetAllEventsTask" and make "AsyncTask" the super class.

```
public class GetAllEventsTask
    extends AsyncTask<Void, Void, List<SaleEvent>>
```

The first parameter (Void) is passed to the "doInBackground" method

The second parameter (Void) is passed to the "onProgress" method

The third parameter (ArrayList<SaleEvent>) is returned from the "doInBackground" method and then passed to "onPostExecution"

a. Create a constructor to pass the context which is a reference to MainActivity.

```
private MainActivity mainActivity;

public GetAllEventsTask(MainActivity mainActivity) {
  this.mainActivity = mainActivity;
}
```

b. Declare the method "doInBackground" which runs on a separate thread. The long running process will be executed in this method, outside of the UI thread.

```
@Override
protected List<SaleEvent> doInBackground(Void... params) {
    return SaleEventManager.getAllEvents(mainActivity);
}
```

7. The user should be notified that data is being retrieved.

a. In the "preExecute" method in "GetAllEventsTask.java", start a progress bar.

```
ProgressDialog pd;

@Override
protected void onPreExecute() {
    mainActivity.displayRecyclerView(new ArrayList<SaleEvent>());
    pd = new ProgressDialog(mainActivity);
    pd.setMessage("Loading Events ...");
    pd.show();
}
```

b. In the "postExecute" method in "GetAllEventsTask.java", stop the progress bar and display the RecyclerView with the data from the background task.

```
@Override
protected void onPostExecute(List<SaleEvent> events) {
    mainActivity.displayRecyclerView(events);
    pd.hide();
}
```

Lab 12.1 - Making HTTP Calls and Parsing JSON

Overview:

In this lab you will retrieve data from a cloud-based service provider.

Review the following resources before starting the lab:

http://android-developers.blogspot.com/2011/09/androids-http-clients.html

http://developer.android.com/intl/zh-CN/reference/java/net/HttpURLConnection.html

Steps

1. Write code to make HTTP request to get all events
    a. In the package "com.garagesaleslibrary.event.service" create a new class called "SaleEventRestManager".
    b. Create a method named "getAllSalesEvents" to make an Http request to the server and to return a collection of SaleEvent objects.

```
public static List<SaleEvent> getAllSalesEvents(Context context) {
    Log.v(TAG, "Running getAllEvents");
    String readEventFeed = readEventFeed();
    return parseJson(readEventFeed);
}
```

    c. Create a method named "readEventFeed" which returns a JSON string retrieved from the server.

```
public static String readEventFeed() {

    StringBuilder builder = new StringBuilder();

    try {
        URL url =
            new URL("https://garagesalesapp.firebaseio.com/events.json");
        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

        // check response code from server
        int response = urlConnection.getResponseCode();
        Log.d(TAG, "Http Response: " + response);

        InputStream content =
            new BufferedInputStream(urlConnection.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(content));
        String line;
        while ((line = reader.readLine()) != null) {
            builder.append(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return builder.toString();
}
```

The code will return a JSON string from the server.

2. Create a method named "parseJson" which will parse JSON string and return SaleEvent objects.

```
private static List<SaleEvent> parseJson(String readEventFeed) {
    List<SaleEvent> events = new ArrayList<SaleEvent>();
    try {

        JSONObject json = new JSONObject(readEventFeed);

        Iterator itr = json.keys();
        while(itr.hasNext()) {

            String key = (String) itr.next();
            JSONObject eventHolder = json.getJSONObject(key);
            JSONObject jsonObject = eventHolder.getJSONObject("event");

            // create event object here
            SaleEvent event = new SaleEvent();

            try {
                event.setId(jsonObject.getString("id"));
                event.setDescription(jsonObject.getString("description"));
                event.setTitle(jsonObject.getString("title"));
                event.setCity(jsonObject.getString("city"));
                event.setStreet(jsonObject.getString("street"));
                event.setRating(Float.parseFloat(jsonObject.getString("rating")));

            } catch (JSONException e) {
                e.printStackTrace();
            }

            events.add(event);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
    return events;
}
```

3. Create a method named "addEvent" which will add an event to the cloud.

```
public static void addEvent(Context context, SaleEvent event) {

     event.setId(UUID.randomUUID().toString());
     event.setDate(new java.util.Date());

     try {
         JSONObject json = new JSONObject();
```

```
            json.put("_id", event.getId());
            json.put("street", event.getStreet());
            json.put("description", event.getDescription());
            json.put("title", event.getTitle());
            json.put("city", event.getCity());
            json.put("rating", event.getRating());

            String urlString =
                    String.format(
"https://garagesalesapp.firebaseio.com/events/%s/event.json", event.getId());

            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();

            conn.setConnectTimeout(5000);
            conn.setRequestProperty("Content-Type", "application/json; charset=UTF-8");
            conn.setDoOutput(true);
            conn.setDoInput(true);
            conn.setRequestMethod("PUT");

            OutputStreamWriter writer = new OutputStreamWriter(conn.getOutputStream());
            String output = json.toString();
            writer.write(output);
            writer.flush();
            writer.close();

            // read the response

            InputStream input = conn.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            StringBuilder result = new StringBuilder();
            String line;

            while ((line = reader.readLine()) != null) {
                result.append(line);
            }
            Log.d("doInBackground(Resp)", result.toString());
            JSONObject response = new JSONObject(result.toString());

            conn.disconnect();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

4. Write a unit test to test the code.

```
public class SaleEventRestManagerTest extends AndroidTestCase {

    public void testJsonParsing() {
        SaleEventRestManager manager = new SaleEventRestManager();
        List<SaleEvent> events = manager.getAllSaleEvents(getContext());

        for (SaleEvent event : events) {
            System.out.println(event.getStreet());
        }
        assertEquals(15, events.size());
    }
}
```

5. Accessing the internet from an app requires explicit permission. Add the following permission to the AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

Lab 13.1 - Creating a Service

Overview

In this lab you will create a service to synchronize the data in the database with the cloud

Setup

Prior labs must be completed.

Overview

Steps

1. Create the service class

```
public class SaleEventSyncService extends IntentService {

    private static final String TAG = SaleEventSyncService.class.getSimpleName();

    public SaleEventSyncService() {
        super(SaleEventSyncService.class.getSimpleName());
        Log.v(TAG, "Running constructor");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.v(TAG, "Running method onHandleIntent");
    }
}
```

2. Add code to "onHandlerIntent" to perform the sync with the cloud

```
@Override
protected void onHandleIntent(Intent intent) {
    Log.v(TAG, "Running method onHandleIntent");
    // get data from the cloud
    List<SaleEvent> events
            = SaleEventRestManager.getNewEvents(this);
    for (SaleEvent event : events) {
        SaleEventManager.addEvent(this, event);
    }

    // put data into the cloud

}
```

3. Register the service

   Add the following to the Android Manifest in the <application> section.

```
<service
    android:name=".SaleEventSyncService"/>
```

4. For now, start the service by running it from the menu in MainActivity by replacing the "Refresh" option.

```
private void refresh() {
    Log.v(TAG, "running refresh method.");
    Intent intent = new Intent(this, SaleEventSyncService.class);
    startService(intent);
}
```

Lab 14.1 - Permissions

Overview

In this lab you a unit test for permissions

Setup

Prior labs must be completed.

Steps

1. Add Robolectric to project

```
dependencies {
  ...
  testCompile "org.robolectric:robolectric:3.0"
  testCompile "com.google.truth:truth:0.27"
}
```

2. Create test

```
@RunWith(RobolectricTestRunner.class)
@Config(manifest = Config.NONE)
public final class PermissionsTest {

    private static final String[] EXPECTED_PERMISSIONS = {
        [...]
    };

    private static final String MERGED_MANIFEST =
        "build/intermediates/manifests/full/debug/AndroidManifest.xml"

    @Test
    public void shouldMatchPermissions() {
        AndroidManifest manifest =
            new AndroidManifest(Fs.fileFromPath(MERGED_MANIFEST), null,null);

        assertThat(new HashSet<>(manifest.getUsedPermissions())).
            containsOnly(EXPECTED_PERMISSIONS);
    }
}
```

Resources

Testing Permissions http://jeremie-martinez.com/2016/02/16/unit-tests/

Lab 15.1 - Broadcast Receiver

Overview

In this lab you will create a Broadcast Receiver.

Setup

Prior labs must be completed.

Overview

Steps

1. Create Broadcast Receiver in the app module

```
public class BootReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d("BootReceiver", "Running boot receiver")
    }
}
```

2. Register Broadcast Receiver

   Add the following to the Android Manifest in the <application> section.

```
<receiver
    android:name=".BootReceiver"/>
```

3. Add permission for Boot Completed) Add the following to the Android Manifest in the <manifest> section (not the <application> section). The convention is to place permissions above the <application> section.

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

4. Update the Broadcast Receiver registration with Intent information

```
<receiver android:name=".BootReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
    </intent-filter>
</receiver>
```

Install the new apk on the device by running it.

5. Trigger the broadcast receiver by sending an Intent

```
adb shell su 0 am broadcast -a android.intent.action.BOOT_COMPLETED
   -c android.intent.category.HOME
   -n com.garagesalesapp/com.garagesalesapp.BootReceiver
```

Instead of "su 0" you can run "adb root" first

Leaving out the "-n" argument will cause a security exception.

Lab 16.1 - System Services

Overview:

In this lab you will utilize many of the fundamental application components of the Android architecture. You will use the AlarmManager to schedule jobs, the BroadCast Receiver to respond to an alarm and start a Service, and the Notification manager to alert the user to new data.

Steps

1. Create a utility class to hold synchronization related methods. Create method to add an alarm for scheduling the event update to be run at a regular interval.
    a. Create a new packaged called "com.garagesalesapp.sync". In the package, create a new class called "EventUpdateUtils.java".
    b. In "EventUpdateUtils" create a method called "scheduleEventUpdate". This method will be called when we want to set an alarm to run the update.

```
public static void scheduleEventUpdate(Context context) {

    Log.v(TAG, "Creating alarm for event update");
    Intent intent =
        new Intent(context, EventUpdateBroadcastReceiver.class);
    int requestId = (int) System.currentTimeMillis();
    PendingIntent pendingIntent =
        PendingIntent.getBroadcast(context.getApplicationContext(), requestId,
intent,
    PendingIntent.FLAG_UPDATE_CURRENT);

    // Get a reference to the alarm manager
    AlarmManager alarmManager =
        (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);

    // We want the alarm to go off 15 seconds from now.
    Calendar startTime = Calendar.getInstance();
    startTime.setTimeInMillis(System.currentTimeMillis());
    startTime.add(Calendar.SECOND, 15);

    // Schedule the alarm, the alarm should repeat every 30 seconds
    int repeatSeconds = 30;
    alarmManager.setRepeating(AlarmManager.RTC, startTime.getTimeInMillis(),
        repeatSeconds * 1000, pendingIntent);
}
```

This code will contain a compiler error until you create the "EventUpdateBroadcastReceiver" is a later step.

c. In "GarageApplication" schedule the event update in the "onCreate" method.

```
@Override
public void onCreate() {
    super.onCreate();
    Log.d(TAG, "running onCreate");
    EventUpdateUtils.scheduleEventUpdate(getApplicationContext());
}
```

2. Create the BroadcastReceiver class that will be run by the alarm manager.

a. In the package "com.garagesalesapp.sync" create a new class called " EventUpdateBroadcastReceiver" which extends " android.content.BroadcastReceiver". Override the "onReceive" method which will be executed when the intent is scheduled by the alarm manger.

This method will start a service which will update the events. Do not execute an AsyncTask directly in this method since the AsyncTask will be killed once the BroadcastReceiver is done executing which is immediately after it would start the AsyncTask.

```
@Override
public void onReceive(Context context, Intent intent) {
    Log.v(TAG, "Starting event update broadcast receiver");
    Intent myIntent = new Intent(context, SaleEventSyncSerivce.class);
    context.startService(myIntent);
}
```

b. Add the BroadcastReciever to the manifest

```
<receiver
    android:name=".sync.EventUpdateBroadcastReceiver"/>
```

3. Create a new utility method for creating a notification.

a. Copy the notification icon from "AndroidLabsSetup/Lab16.1" to the "res/drawable" directory.

b. In "EventUpdateUtils" add the following method.

```
public static void createNotification (Context context, String eventStreet) {

    // Define the Notification's expanded message and Intent:
    int icon = R.drawable.notify;
    CharSequence tickerText = "New garage sale at " + eventStreet;
    CharSequence contentTitle = "New Garage Sale";
    CharSequence contentText = "Sale at " + eventStreet;

    Intent intent = new Intent(context, MainActivity.class);

    PendingIntent contentIntent =
        PendingIntent.getActivity(context, requestId, intent, 0);

    NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(context)
        .setSmallIcon(icon)
        .setContentTitle("New Garage Sale")
        .setContentText("Sale at " + eventStreet)
        .setTicker(tickerText)
        .setContentIntent(contentIntent);

    Notification notification = mBuilder.build();

    //Get a reference to the NotificationManager:
    String ns = Context.NOTIFICATION_SERVICE;
    NotificationManager mNotificationManager =
        (NotificationManager) context.getSystemService(ns);

    // Pass the Notification to the NotificationManager:
    int notificationId = (int) System.currentTimeMillis();
    mNotificationManager.notify(notificationId, notification);
}
```

4. Update the sync service to create notifications for new events.

```
@Override
protected void onHandleIntent(Intent intent) {

    Log.v(TAG, "Running method onHandleIntent");

    // get data from the cloud
    List<SaleEvent> newEvents
            = SaleEventRestManager.getNewEvents(this);
    for (SaleEvent event : newEvents) {
        Log.d(TAG, "Writing to local: " + event);
        SaleEventManager.addEvent(this, event);
        EventUpdateUtils.createNotification(getApplicationContext(), event.getStreet(
));
    }

    // put data into the cloud
    List<SaleEvent> unpostedEvents
            = SaleEventRestManager.getUnpostedEvents(this);
    for (SaleEvent event : unpostedEvents) {
        Log.d(TAG, "Writing to cloud: " + event);
        SaleEventRestManager.addEvent(event);
    }
}
```

5. Update rest service

```
public static List<SaleEvent> getNewEvents(Context context) {
    List<SaleEvent> localEvents = SaleEventManager.getAllEvents(context);
    List<SaleEvent> cloudEvents = SaleEventRestManager.getAllEvents();
    cloudEvents.removeAll(localEvents);
    return cloudEvents;
}

public static List<SaleEvent> getUnpostedEvents(Context context) {
    List<SaleEvent> localEvents = SaleEventManager.getAllEvents(context);
    List<SaleEvent> cloudEvents = SaleEventRestManager.getAllEvents();
    localEvents.removeAll(cloudEvents);
    return localEvents;
}
```

Run the application and verify that notifications have been created.

Lab 17.1 - Using WebView

Overview

In this lab you add a WebView widget to a screen.

Setup

Prior labs must be completed.

Steps

1. Create a new project.
    a. Create a new Android project called "Lab17.1 - WebView".
2. Add WebView to main layout
    a. Use the <WebView> element
3. Reference the WebView in the activity and load the intial url.

```
loadUrl("http://garagesalesapp.firebase.com/demos/lab17.html")
```

Run the app and verify that the page loads and is visible.

4. Add a method to the app that can be executed from the page.
    a. Enable JavaScript on the WebView
    b. Declare a new JavaScriptInterface class
    c. Bind the JavaScriptInterface object to the web page
    d. Execute the app and verify that you can enter a message in the web page and display it as a toast in the app.

Lab 18.1 - Display A Map

Overview:

In this lab you will display a map using the Google Play services.

Steps

1. Add the Google Play Services to the module by including a dependency in the build.gradle file.

```
dependencies {
       compile 'com.google.android.gms:play-services:10.0.1'
    ...
}
```

Use the most current version of play services.

2. Add the debug keystore to the app. Get the file from "AndroidLabsSetup/Lab18.1/debug.keystore" and copy to the "app" directory.
3. Sign the debug version of the app with the new debug keystore.

   Add the following to the "build.gradle" file in the app after the "defaultConfig" section.

```
signingConfigs {
    // Place file in "apps" directory
    debug {
        storeFile file("debug.keystore")
    }
}
```

4. Configure the "AndroidManifest.xml" file to use Google maps
   a. Add permissions to the "AndroidManifest.xml" before the "application" section.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.INTERNET" />
```

   b. Specify the Maps API key in the "application" section after the activities.

```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyDIDGRPM9xwJuKGJifvi_QEGiY0JZzo27o" />
```

You will eventually need your own API key. Detailed instructions are at: https://developers.google.com/maps/documentation/android/start\#installing_the_google_maps_android_v2_api

5. Create new layout using MapView

   In the "res/layout" directory, create a file called "event_map.xml"

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <fragment
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.MapFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

6. Create a new activity to display the map

   Create a new activity called "EventMapActivity.java" that extends "AppCompatActivity"

```
public class EventMapActivity extends AppCompatActivity {

    private static final String TAG = EventMapActivity.class.getSimpleName();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_map);
    }
}
```

   a. Add the new activity to the application manifest.

```
<activity
    android:name=".EventMapActivity"></activity>
```

7. Call the map activity from the main menu by starting the intent.
   a. Add the intent to the "showMaps" method.

```
Intent intent = new Intent(this, EventMapActivity.class);
startActivity(intent);
```

8. Add the code to display the map
   a. Add the following code to the "onCreate" method.

```
MapFragment mapFragment =
 (MapFragment) getFragmentManager().findFragmentById(R.id.map);

mapFragment.getMapAsync(this);
```

   b. EventMapActivity should implement the "OnMapReadCallback"

```
public class EventMapActivity
    extends AppCompatActivity
    implements OnMapReadyCallback
```

   c. Add the following code to the "EventMapActivity.java"

```
private GoogleMap map;

@Override
public void onMapReady(GoogleMap googleMap) {
  map = googleMap;
}
```

9. Run the application on a emulator using the Google APIs

   You should see a map similar to the following:

If you see a blank gray screen (possibly with grid lines), you probably have either the permissions or the API key wrong.

10. Configure the map by specifying the zoom level and the center of the map.
    a. Create a new method named "displayMap"

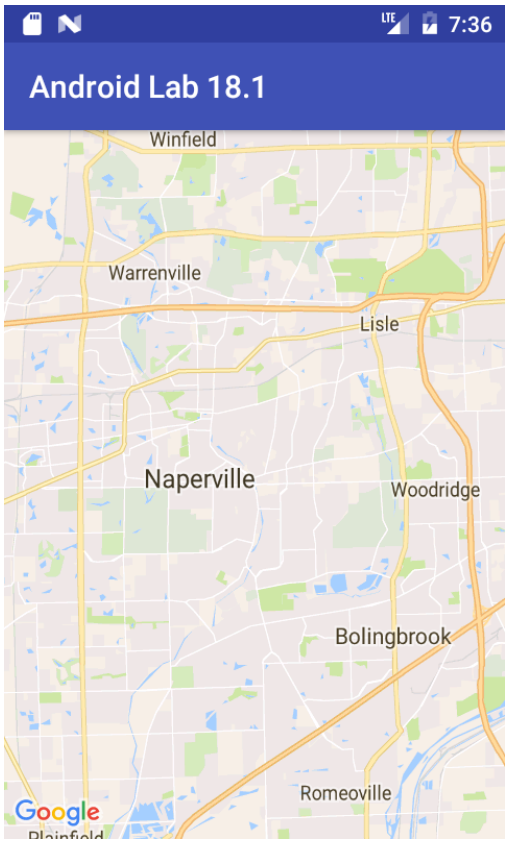
```
private void displayMap() {

  // Center on Naperville, Illinois
  LatLng latLng = new LatLng(41.748276D, -88.129797D);

  CameraUpdate center = CameraUpdateFactory.newLatLng(latLng);

  map.moveCamera(center);

  CameraUpdate zoom = CameraUpdateFactory.zoomTo(11);

  map.animateCamera(zoom);

}
```

    b. Call "displayMap() from the "onMapReady" method

```
@Override
public void onMapReady(GoogleMap googleMap) {
  map = googleMap;
  displayMap();
}
```

11. Run the application again.

    You should see a map centered over Naperville similar to the following:



Resources

1. Get your own Google Map API key. This is required to use your own signing keystore.

   The instructions for getting an API key are at:

   <hhttps://developers.google.com/maps/documentation/android-api/signup>

Explore the API for MapFragment https://developers.google.com/android/reference/com/google/android/gms/maps/MapFragment

Explore the API for CameraUpdate

https://developers.google.com/android/reference/com/google/android/gms/maps/CameraUpdate
https://developers.google.com/android/reference/com/google/android/gms/maps/GoogleMap

Lab 18.2 - Map Markers

Overview:

In this lab you will display markers on a map. A map will be displayed and markers will be displayed on top of the map containing information about events.

Setup:

Prior lab must be completed.

Steps

1. Create a new method called "displayMarkers()"

```
private void displayMarkers() {
}
```

2. Call the "displayMarkers" method as the last step in the "displayMaps" method.

```
displayMarkers();
```

3. Add code is "displayMarkers() to iterate through each event.

   In the "displayMarkers" method add the following code:

```
List<SaleEvent> events = SaleEventManager.getAllEvents(this);

for (SaleEvent event : events) {
  // Add additional code here
}
```

The following steps require you to add code to the loop to process each event object.

4. Create the latitude/longitude position for each event.

```
markerOptions.position(new LatLng(event.getLatitude(), event.getLongitude()));
```

5. Create the Marker item for each event.

```
MarkerOptions markerOptions = new MarkerOptions();
```

6. Assign the LatLng to the Marker

```
markerOptions.position(itemLatLng);
```

7. Copy "garage.png" from "AndroidLabsSetup/Lab18.2" into "res/drawable"
8. Add an icon to the marker

```
markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.garage));
```

9. Add a message to the icon that will appear when the user taps the icon.

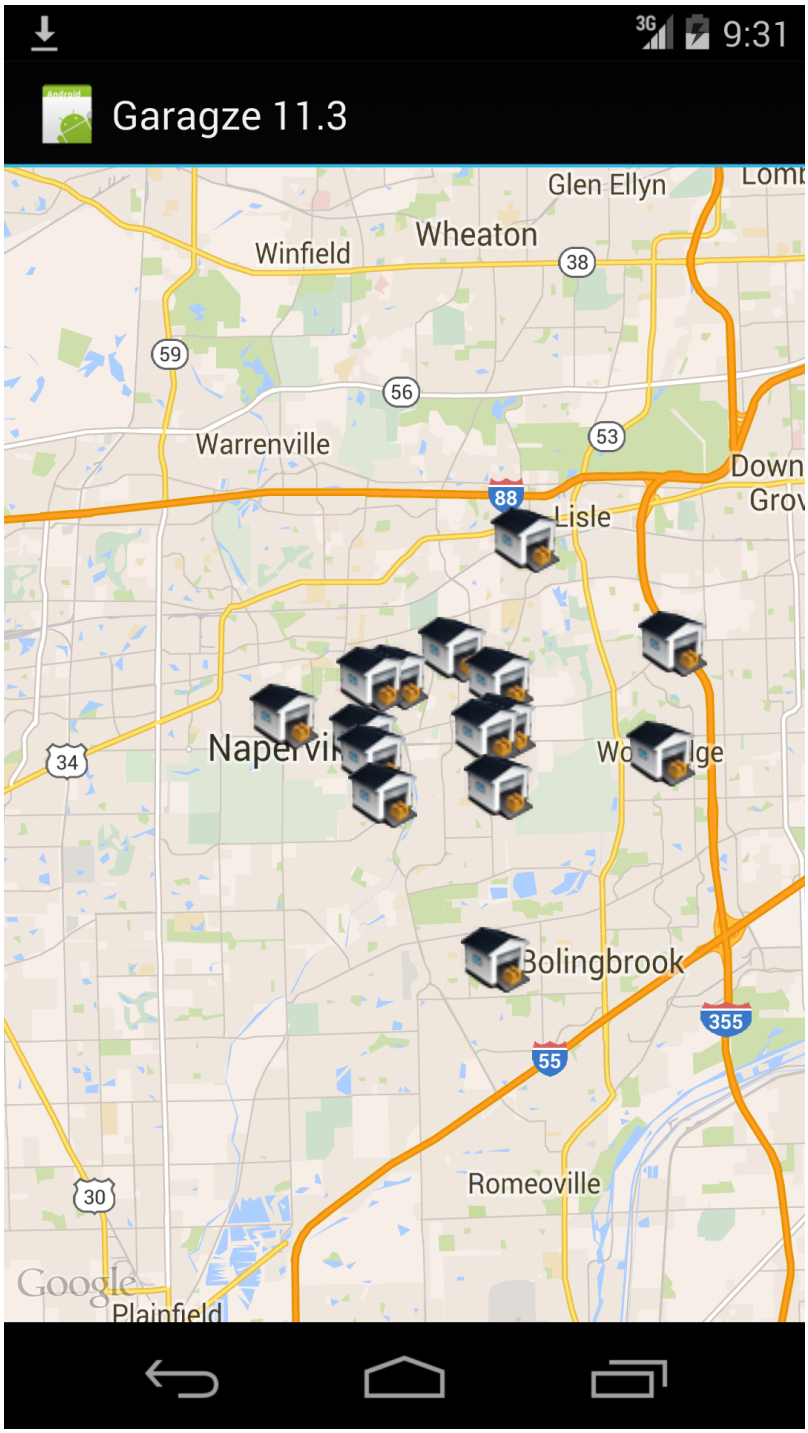
```
markerOptions.title(event.getStreet() + ", " + event.getCity());
```

10. Add the Marker to the map:

```
map.addMarker(markerOptions);
```

11. Run the application on a Google emulator

    You should see a map similar to the following:



12. Here is the listing for the EventMapActivity.

```
public class EventMapActivity
  extends AppCompatActivity
  implements OnMapReadyCallback {

    private static final String TAG = EventMapActivity.class.getSimpleName();

    private GoogleMap map;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {

        Log.v(TAG, "Starting EventMap");
        super.onCreate(savedInstanceState);
        //requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.event_map);

        MapFragment mapFragment =
                (MapFragment) getFragmentManager().findFragmentById(R.id.map);

        mapFragment.getMapAsync(this);

    }

    private void displayMap() {
        // Center on Naperville, Illinois
        LatLng latLng = new LatLng(41.748276D, -88.129797D);

        CameraUpdate center = CameraUpdateFactory.newLatLng(latLng);

        map.moveCamera(center);

        CameraUpdate zoom = CameraUpdateFactory.zoomTo(11);

        map.animateCamera(zoom);

        displayMarkers();
    }

    private void displayMarkers() {
        List<SaleEvent> events = SaleEventManager.getAllEvents(this);
        for (SaleEvent event : events) {

            MarkerOptions markerOptions = new MarkerOptions();
            markerOptions.position(new LatLng(event.getLatitude(), event.getLongitude
()));

            markerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.garage
));

            markerOptions.title(event.getStreet() + ", " + event.getCity());

            map.addMarker(markerOptions);

        }
    }
```

```
    @Override
    public void onMapReady(GoogleMap googleMap) {
        map = googleMap;
        displayMap();
    }

}
```

18.3 - Location Using Google Mobile Services (GMS)

Overview

In this lab you use the GMS location service to show the current location of the device.

Setup

Prior location labs must be completed.

Additional Instructions

This lab is based on content from

http://developer.android.com/training/location/index.html

Definitions

GoogleApiClient
https://developers.google.com/android/reference/com/google/android/gms/common/api/GoogleApiClient

FusedLocation Location

Steps

1. Create a GoogleApiClient in the onCreate method of EventMapActivity

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(LocationServices.API)
        .build();
```

1. Implement the methods for addConnectionCallbacks. Add logging to the methods.
2. Implement the methods for addOnConnectionFailedListener. Add logging to the methods.
3. Start the GoolgeApiCient by running the connect method.

```
mGoogleApiClient.connect();
```

1. Run the app and verify that the Location Services are connected. Inspect the log.
2. Extract the map display code into a separate method called displayMap
   a. The method should take latitude and longitude as parameters.
   b. Modify the displayMap method to use the passed arguements as to center the map
3. Remove any code to display the map in the onCreate method since the location services may not have started yet.
4. Modify the onConnected method to retrieve the last location and to display the map if a location was found.

```
Location lastLocation =
    LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);

if (lastLocation != null) {
    Log.d(TAG, "Display map at last known location");
    double latitude = lastLocation.getLatitude();
    double longitude = lastLocation.getLongitude();
    displayMap(latitude, longitude);
} else {
    Log.d(TAG, "Last location is null");
}
```

5. Create a location update listener. This will be called when the location is updated by the location service.
   a. Change the signature of the class to implement com.google.android.gms.location.LocationListener
   b. Implement the Listener

```
@Override
public void onLocationChanged(Location location) {
    if (location != null) {
        Log.d(TAG, "Display map at updated location");
        double latitude = location.getLatitude();
        double longitude = location.getLongitude();
        displayMap(latitude, longitude);

    } else {
        Log.d(TAG, "Updated location is null");
    }
}
```

6. Create a LocationRequest object. Ask for location updates every 5 seconds.

```
LocationRequest mLocationRequest = new LocationRequest();
mLocationRequest.setInterval(5000);
mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

7. Register to receive the location updates.

```
LocationServices.FusedLocationApi
    .requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
```

8. Run the app. Watch the log for location updates.

Resources

Based on content from

+ https://www.raywenderlich.com/144066/introduction-google-maps-api-android?utm_source=androiddevdigest

Lab 19.1 - Advanced UI

Overview

In this lab you will use material design and animations.

Setup

You must have completed the prior labs.

Steps

1. Create new color values in "values/colors.xml"

```
<color name="primary">#673ab7</color>
<color name="primary_dark">#512da8</color>
<color name="accent">#ffc400</color>
```

2. Modify the theme in "values/styles.xml"

```
<style name="AppTheme" parent="android:Theme.Material.Light.DarkActionBar">
    <!--   your app branding color for the app bar -->
    <item name="android:colorPrimary">@color/primary</item>
    <!--   darker variant for the status bar and contextual app bars -->
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <!--   theme UI controls like checkboxes and text fields -->
    <item name="android:colorAccent">@color/accent</item>
</style>
```

3. Add a Floating Action Bar to replace menu option for adding a new activity.

Additional Resources

Free app which prints 8dp gridlines on top of other apps Useful for verifying adherance to Material design guidelines

```
https://play.google.com/store/apps/details?id=com.faizmalkani.keylines&hl=en
```

Use "Settings → Developer options → Show layout bounds" to show boundaries around layout elements when app is running

Lab 19.2 - Card View

Overview

In this lab you will implement drag and drop with CardView

Setup

You must have completed the prior labs.

Steps

1. Update method "displayRecyclerView" in "MainActivity".

```
ItemTouchHelper.SimpleCallback simpleItemTouchCallback =
        new ItemTouchHelper.SimpleCallback(
            (ItemTouchHelper.UP |
            ItemTouchHelper.DOWN |
            ItemTouchHelper.LEFT |
            ItemTouchHelper.RIGHT), (ItemTouchHelper.UP |
            ItemTouchHelper.DOWN |
            ItemTouchHelper.LEFT |
            ItemTouchHelper.RIGHT))
  {
    @Override
    public boolean onMove(RecyclerView recyclerView, RecyclerView.ViewHolder
viewHolder, RecyclerView.ViewHolder target) {
        mAdapter.swap(viewHolder.getAdapterPosition(), target.getAdapterPosition());
        return true;
    }

    @Override
    public void onSwiped(RecyclerView.ViewHolder viewHolder, int direction) {
        int remInd = viewHolder.getAdapterPosition();
        events.remove(remInd);
        mAdapter.notifyItemRemoved(remInd);
        mAdapter.notifyItemRangeChanged(remInd, events.size());
        Toast.makeText(MainActivity.this, String.format("Item %s removed", remInd),
Toast.LENGTH_LONG).show();
    }
};
+
ItemTouchHelper itemTouchHelper = new ItemTouchHelper(simpleItemTouchCallback);
itemTouchHelper.attachToRecyclerView(mRecyclerView);
```

Be sure "events" are declared as an instance variable.

2. Add "swap" method to adapter

```
public void swap(int firstPosition, int secondPosition) {
      String logMsg = "BEFORE SWAPED method. : " + events;
      Log.v(TAG, logMsg);
      Collections.swap(events, firstPosition, secondPosition);
      logMsg = "After SWAPED method. : " + events;
      Log.v(TAG, logMsg);
      notifyItemMoved(firstPosition, secondPosition);
}
```

Lab 19.3 - Activity Transition

Overview

In this lab you will implement a transition between the list of events and the display of a single event.

Setup

You must have completed the prior labs.

Steps

1. Examine the transition from list to detail by clicking on an item in the RecyclerView. Can you describe what the transition is. How does the first screen leave the page? How does the detail screen enter the page?
2. Add a transition name to the CardView element in the layout file for the RecyclerView row and the add_event.xml layout.

   ```
   android:transitionName="cardview"
   ```

   It is not necessary to make the id of the CardView the same in both layouts.

3. Implement a transition between the two activities using the image as the connecting element.
   a. Start the Add activity using a transition. Replace item click listener with the following:

   ```
   Intent intent = new Intent(this, AddEventActivity.class);
   if (android.os.Build.VERSION.SDK_INT >= android.os.Build.VERSION_CODES.LOLLIPOP) {
       ActivityOptions options = ActivityOptions
               .makeSceneTransitionAnimation(getActivity(), cardView, "cardview");
       startActivity(intent, options.toBundle());
   } else {
       startActivity(intent);
   }
   ```

   b. In the AddEventActivity set the transition.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
    // inside your activity (if you did not enable transitions in your theme)
    getWindow().requestFeature(Window.FEATURE_CONTENT_TRANSITIONS);
    // set an exit transition
    getWindow().setExitTransition(new Explode());
}

setContentView(R.layout.activity_article_detail);
```

You must set the transition before the setting the content.

Resources

https://developer.android.com/training/material/animations.html#Transitions

Lab 20.1 - Improving Gradle Build Times

Overview

In this lab you will implement various Gradle optimizations.

Setup

Prior labs must be completed.

Steps

1. Whenever working to increase performance, start with a baseline timing.
   a. Run gradle and receive a profile report

      ```
      gradlew build -profile
      ```

      NOTE: Look for the profile report in <app>/build/output/...

   b. Measure how long it takes to run gradle without any app specific processing

      ```
      ./gradlew :android:assembleDebug --dry-run --profile
      ```

2. Set Gradle Properties. Try these properties individually to measure the performance of each one.

```
# Improves Gradle startup time
org.gradle.daemon=true

# When configured, Gradle will run in incubating parallel mode.
# This option should only be used with decoupled projects. More details, visit
# http
://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:decoupled_proje
cts
org.gradle.parallel=true

# Enables new incubating mode that makes Gradle selective when configuring projects.
# Only relevant projects are configured which results in faster builds for large multi
-projects.
# http
://www.gradle.org/docs/current/userguide/multi_project_builds.html#sec:configuration_o
n_demand
org.gradle.configureondemand=true

# Specifies the JVM arguments used for the daemon process.
# The setting is useful for tweaking memory settings.
# Default value: -Xmx10248m -XX:MaxPermSize=256m
org.gradle.jvmargs=-Xmx2048m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -
Dfile.encoding=UTF-8
```

You can assign the properties at the command line instead:

```
./gradlew :android:assembleDebug --dry-run --profile --configure-on-demand --daemon
```

3. Remove unnecessary tasks. Do you really need to run link checks on every build?

   Add the following to "build.gradle" to permanently remove lint checks:

```
tasks.whenTaskAdded { task ->
    if (task.name.equals("lint")) {
        task.enabled = false
    }
}
```

Resources

Speeding Up Gradle

https://medium.com/the-engineering-team/speeding-up-gradle-builds-619c442113cb

https://medium.com/@shelajev/6-tips-to-speed-up-your-gradle-build-3d98791d3df9

https://speakerdeck.com/madisp/squeezing-the-last-drop-of-performance-out-of-your-gradle-builds-droidcon-paris-2015

Using Gradle Build Cache

http://zeroturnaround.com/rebellabs/using-build-cache-in-android-studio-makes-gradle-build-faster/

Other Resources

Additional information on compiling Android

http://trickyandroid.com/the-dark-world-of-jack-and-jill/?utm_source=androiddevdigest

Other Android Gradle Plugin Documentation

http://google.github.io/android-gradle-dsl/current/ http://tools.android.com/build/gradleplugin
http://tools.android.com/tech-docs/new-build-system/user-guide
http://developer.android.com/tools/building/plugin-for-gradle.html

Lab 20.2 - Using Proguard

Overview

In this lab you will use Proguard to remove logging messages from the production version of the app.

Setup

Prior labs must be completed.

Steps

1. Create a separate build type for "debug_no_logging"
2. Add logging to "MainActivity" in the "onCreate" method

   ```
   System.out.println("MainActivity", "Message from System.out");
   Log.d("MainActivity", "Message from logging")
   ```

   Run the app to verify that both messages are visible in the log.

3. Update the "build.gradle" file to enable proguard.

   ```
   debug {
       minifyEnabled true
       proguardFiles 'proguard-rules.pro'
   }
   ```

   The generated "build.gradle" file contains a proguardFiles attribute assignment as show below:

   ```
   proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
   ```

   This causes a system proguard file to be loaded from the following directory:

   <sdk>/tools/proguard/proguard-android.txt'

   This file declares proguard properties. One of the properties causes proguard to not remove the classes.

```
# Optimization is turned off by default. Dex does not like code run
# through the ProGuard optimize and preverify steps (and performs some
# of these optimizations on its own).
-dontoptimize
```

Either comment out the "-dontoptimize" option or remove the reference to the system proguard file.

4. Add the rule to the proguard file to exclude android logging

   proguard-rules.pro

```
-assumenosideeffects class android.util.Log {
    public static boolean isLoggable(java.lang.String, int);
    public static int v(...);
    public static int i(...);
    public static int w(...);
    public static int d(...);
    public static int e(...);
}
```

5. Run the application to verify that logging is gone

Resources

Presentation by Eric LaFortune https://www.youtube.com/watch?v=PSpL2tShmAY

Proguard Documentation http://proguard.sourceforge.net/

Remove Logging Code http://proguard.sourceforge.net/manual/examples.html#logging

Lab 20.3 - Other Gradle Enhancements

Overview

In this lab you will implement various Gradle enhancements

Resources

Enhancing Gradle

https://medium.com/@sergii/make-your-build-gradle-great-again-c84cc172a654#.bwav33p28

Gradle Tasks (Mark launcher with version number)

https://www.reddit.com/r/androiddev/comments/3ig3gm/show_us_your_gradle_tasks