# CS 213 Spring 2016
# Lecture 21: April 5

RU NB Bus Routes Android App

# Create Project

- Make a project called RU NB Bus Routes
- Name company domain whatever you want (e.g. `<your name>.example.com`)
- Package name will be automatically set to `<domain name reversed>.<runbbusroutes>`
- Use the recommended Min SDK level API 15
- Stay with default Empty Activity
- Call the main activity `Routes`
- Call the (generated) main activity layout file `routes_list`

# Part 1:
# Showing a List of Route Names

# Route Names List Layout

- Replace `res/layout/routes_list.xml` with the `routes_list.xml` file in from Sakai Resources -> Apr 5 (apparently drag and drop doesn't work for some reason, so you will need to copy the file and paste it into `res/layout`)

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

<LinearLayout 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.example.sesh.runbbusroutes.Routes">

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

The `ListView` does not contain actual names, it will be populated in the Java code from an external source

(See Develop -> API Guides -> User Interface -> Layouts -> List View)

# Defining route names in `strings.xml`

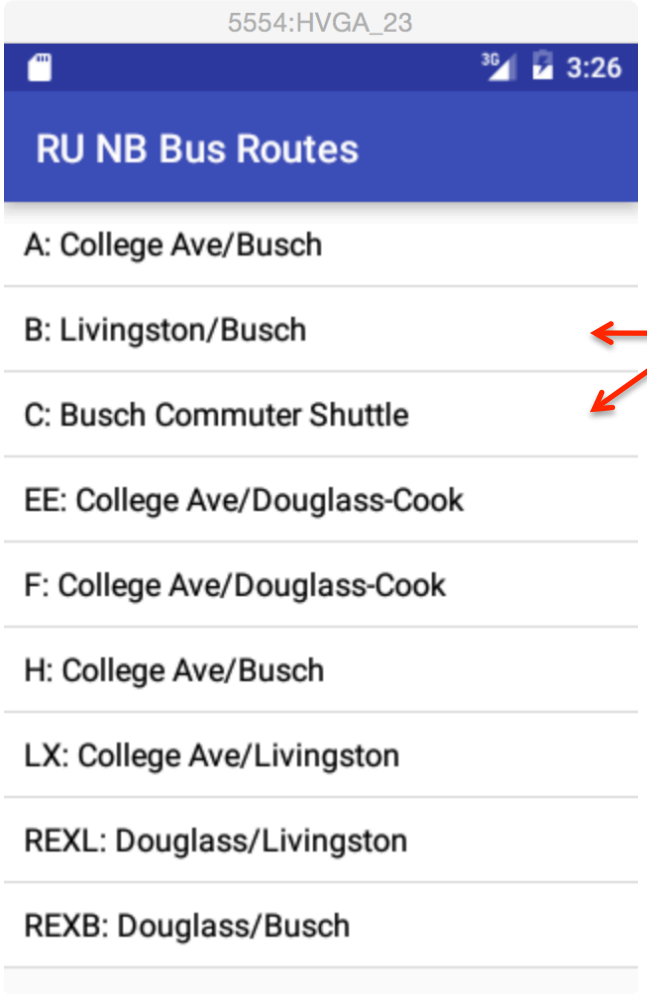
- The `res/values/strings.xml` file will list all the route names in a `string-array` tag (replace with file from Resources -> Apr 5)

```
<resources>
    <string name="app_name">RU NB Bus Routes</string>
    <string-array name="routes_array">
        <item>A: College Ave/Busch</item>
        <item>B: Livingston/Busch</item>
        <item>C: Busch Commuter Shuttle</item>
        <item>EE: College Ave/Douglass-Cook</item>
        <item>F: College Ave/Douglass-Cook</item>
        <item>H: College Ave/Busch</item>
        <item>LX: College Ave/Livingston</item>
        <item>REXL: Douglass/Livingston</item>
        <item>REXB: Douglass/Busch</item>
    </string-array>
</resources>
```

(See Develop -> API Guides -> App Resources -> Resource Types -> String)

# Preview of List

- When the list is populated (how to do this is up in a few), it will look like this




A separate layout file (say `route_name.xml`) needs to be set up to specify the layout of the list items

# Route Name Layout

- Create a `res/layout/route_name.xml` file: for instance, each route name is rendered in black lettering on a white background (copy from Resources -> Apr 5)

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ffffff"
    android:textColor="#000000"
    android:textSize="16dp"
    android:padding="10dp" />
```

Hex color code: RGB, 1 byte per color

# Populating the `ListView` with Route Names

- In the `Routes.java` file, we first need to get the `ListView` object:

```
private ListView listView;

protected void onCreate(Bundle savedInstanceState) {
    ...
    listView = (ListView)findViewById(R.id.routes_list);
}
```

- Then we need to read into an array the names from the `strings.xml string-array`:

```
...
private String[] routeNames;

protected void onCreate(Bundle savedInstanceState) {
    ...
    routeNames = getResources().getStringArray(R.array.routes_array);
}
```

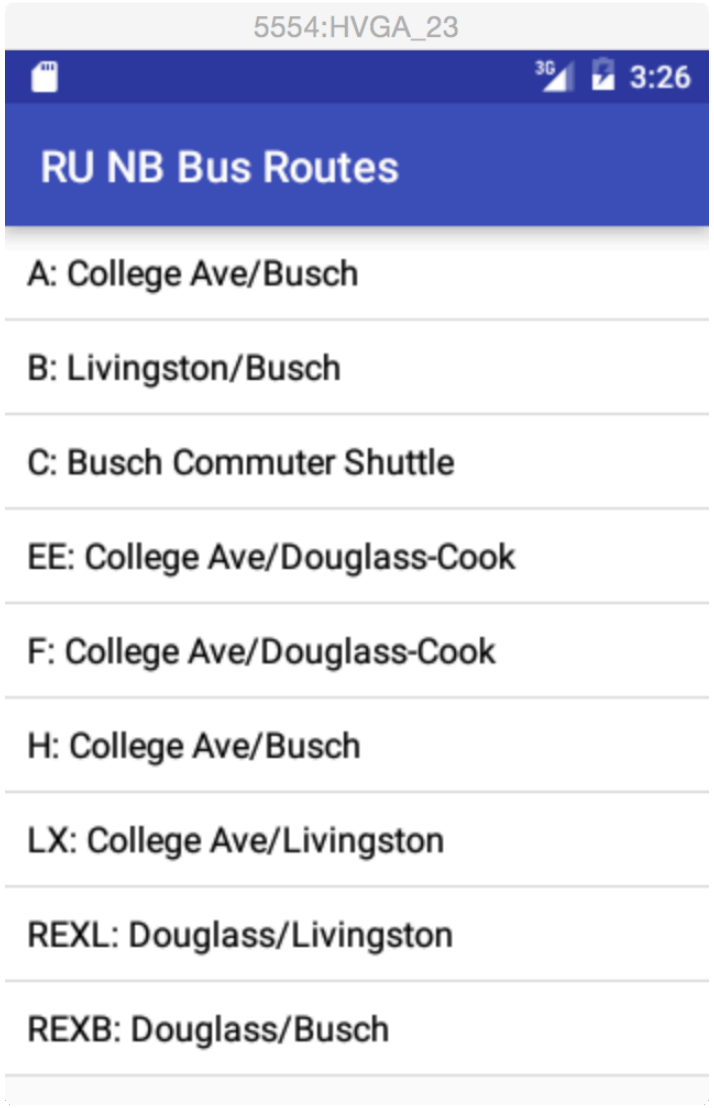
# Populating the `ListView` with Route Names

- Next, we need to create an `ArrayAdapter` to feed the array items into the `ListView`:

```
protected void onCreate(Bundle savedInstanceState) {
   ...

   ArrayAdapter<String> adapter =
     new ArrayAdapter<String>(this,
                              R.layout.route_name,
                              routeNames);

   listView.setAdapter(adapter);
}
```

This is the layout for the list items

# Run the App


5554:HVGA_23
3:26
RU NB Bus Routes
A: College Ave/Busch
B: Livingston/Busch
C: Busch Commuter Shuttle
EE: College Ave/Douglass-Cook
F: College Ave/Douglass-Cook
H: College Ave/Busch
LX: College Ave/Livingston
REXL: Douglass/Livingston
REXB: Douglass/Busch

# Part 2:
# Setting up Details for Routes (Sequence of Stops)

# Route Details

- When the user taps on a route, another screen comes up with the detailed list of stops in each route.
- In this part, we will see how to set up the route details by reading from a file
- In the next part (part 3), we will finish up with the event handling code to link route name tap event to route details

# Route Detail Layout

- Create a `res/layout/route_detail.xml` file (copy from Resources -> Apr 5): for instance, the route name is rendered in a header of while lettering on a red background, and the sequence of stops (detail) is rendered in white lettering on a black background

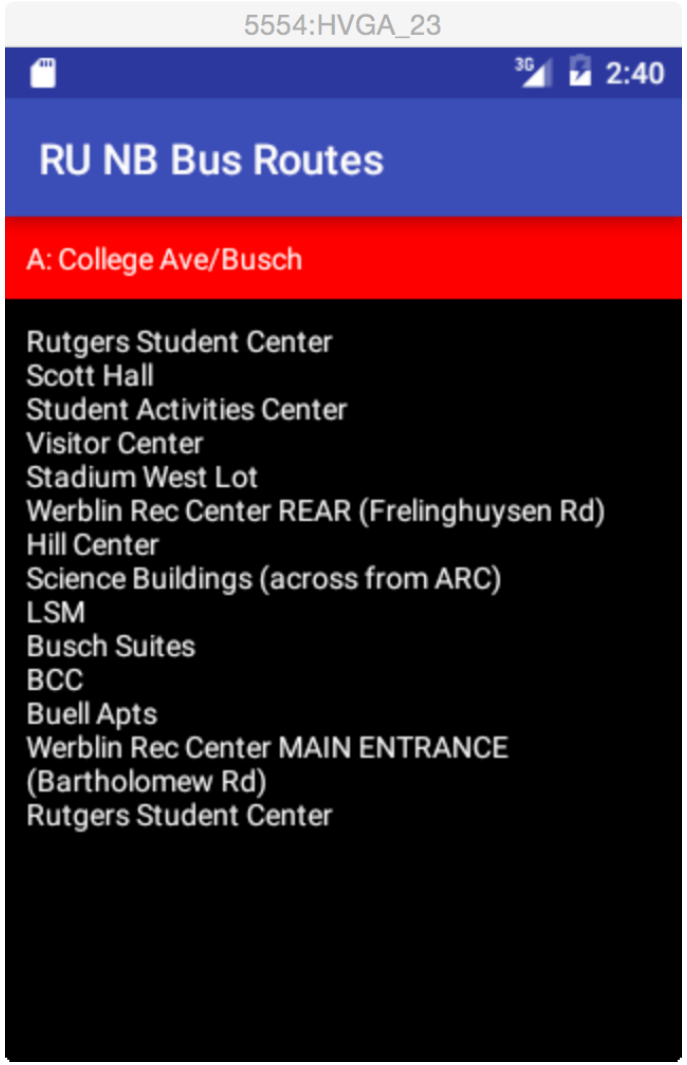
# Route Detail Layout

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


    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/route_name"
        android:background="#ff0000"
        android:textColor="#ffffff"
        android:padding="10dp" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/route_detail"
        android:padding="10dp"
        android:textColor="#ffffff"
        android:background="#000000" />

</LinearLayout>
```

Need to id these so we can set them in the Java code after reading from file

# Preview of Detail

- When the detail is shown (on user tap of route name), it will look like this

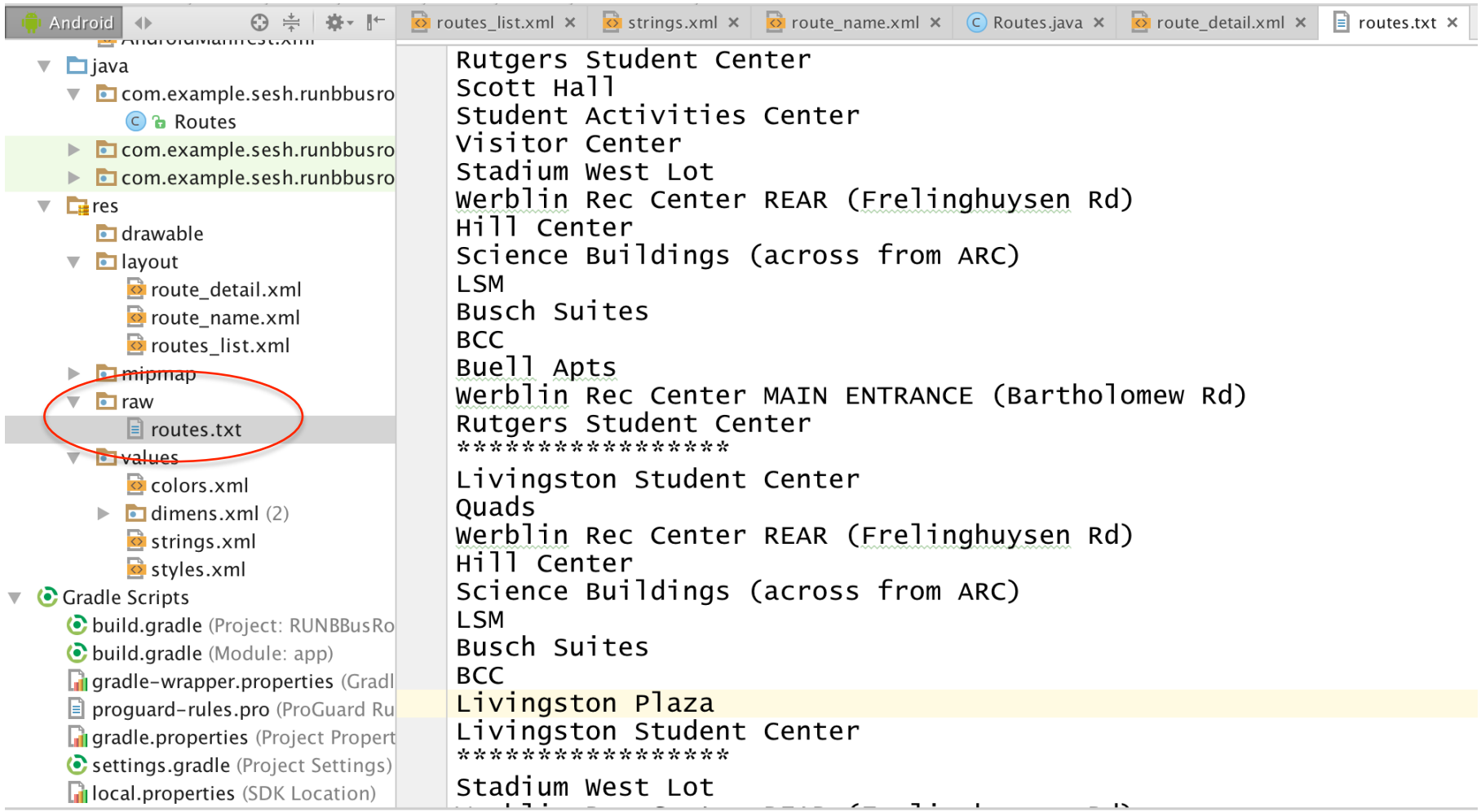


# The Details File

- If you need to read “static” data from a file, i.e. file will not be modified by the app, then you can put the file in a directory called `res/raw`

- Make a directory called `raw` under `res`, and copy the `routes.txt` file (Resources -> Apr 5) in

# The Details File



# Route details as a raw resource

- To read the route details in, you need to open an InputStream to the raw file, like this:

```
InputStream is = getResources().openRawResource(R.raw.routes);
```

(See API Guides -> Data Storage -> Storage Options -> Internal Storage -> Tip: If you want to save a static file in your application ...)

# Reading route details into the program

```
...
private String[] routeDetails;

protected void onCreate(Bundle savedInstanceState) {
   ...
   InputStream is = getResources().openRawResource(R.raw.routes);
   BufferedReader br = new BufferedReader(new InputStreamReader(is));
   routeDetails = new String[routeNames.length];
   try {
       for (int i = 0; i < routeNames.length; i++) {
           StringBuilder sb = new StringBuilder();
           String line = br.readLine();
           while (!line.startsWith("*")) {
               sb.append(line+"\n");
               line = br.readLine();
           }
           routeDetails[i] = sb.toString();
        }
   } catch (IOException e) { }
}
```

# Part 3:
# Responding to selection of route in list

# Launching a new activity

- When the user taps on a route, another screen comes up with the detailed list of stops in each route.
- In this part, we will see how to make this happen by launching a new activity when a route is selected in the routes list

# Set a listener for route list items

- When one of the routes is clicked/tapped on, the details for that route should show up – need to set a listener for the list items

```
listView.setOnItemClickListener(
    new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView,
                                View view,
                                int i,
                                long l) {
            showRoute(i);
        }
    });
```

Index of the item in the list

We will write this method to handle the event by LAUNCHING a new activity to show the route details

# Launching an activity

- Launching an activity requires creating an `Intent` and starting that activity with that `Intent`
- The activity is also listed in the `Manifest` file
- Data can be sent to the activity via a `Bundle`

Key (defined as constant in class)

Value

```
private void showRoute(int pos) {
    Bundle bundle = new Bundle();
    bundle.putString(ROUTE_NAME_KEY,routeNames[pos]);
    bundle.putString(ROUTE_DETAIL_KEY,routeDetails[pos]);

    Intent intent = new Intent(this, ShowRoute.class);
    intent.putExtras(bundle);

    startActivity(intent);
}

public static final String ROUTE_NAME_KEY = "route_name";
public static final String ROUTE_DETAIL_KEY = "route_detail";
```

The activity to launch

# Create the ShowRoute activity

```
public class ShowRoute extends AppCompatActivity {

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

        // get the name and detail from bundle
        Bundle bundle = getIntent().getExtras();
        String routeName = bundle.getString(Routes.ROUTE_NAME_KEY);
        String routeDetail = bundle.getString(Routes.ROUTE_DETAIL_KEY);

        // get the name and detail view objects
        TextView routeNameView = (TextView)findViewById(R.id.route_name);
        TextView routeDetailView = (TextView)findViewById(R.id.route_detail);

        // set name and detail on the views
        routeNameView.setText(routeName);
        routeDetailView.setText(routeDetail);
    }
}
```

Get all bundle data that was passed in

# Manifest file should list the activity

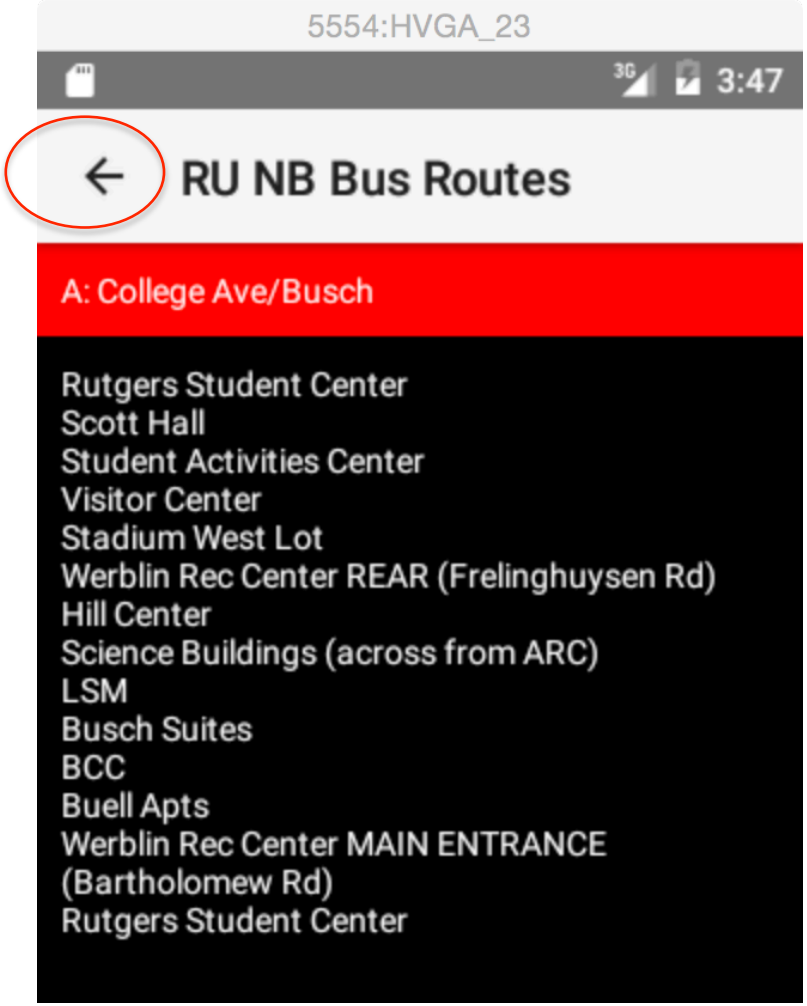
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sesh.runbbusroutes">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".Routes">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".ShowRoute"></activity>
    </application>

</manifest>
```

# Part 4:
# Using the Action Bar

See http://developer.android.com/training/appbar/index.html

# Using the Action Bar

- In the `ShowRoute` activity, we want to set up a way to go back to the Routes activity
- In Android UI, the standard way to do this is to set it up in the Action Bar (also called "app bar"), to go "up" to the previous activity in the stack
- When set up, this is what it would look like:




(See Design -> Pure Android -> Navigation)
http://developer.android.com/design/patterns/navigation.html

# Managing the Action Bar in ShowRoute

- In the application manifest file, turn off the ActionBar:

```
<application
  …
  android:theme="@style/Theme.AppCompat.Light.NoActionBar”/>
```

- Add a ToolBar widget to the top of the route_detail and routes_list layouts:

```
<LinearLayout
  ...>
  <android.support.v7.widget.Toolbar
    android:id="@+id/my_toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
  ...

</LinearLayout>
```

# Managing the Action Bar in `ShowRoute`

- In the `ShowRoute` code's `onCreate` method, get the `Toolbar`, and pass it in to the method to set the support action bar:

```
protected void onCreate() {
  ...
  Toolbar toolbar = (Toolbar)findViewById(R.id.my_toolbar);
  setSupportActionBar(toolbar);
}
```

- Repeat, in the code for `Routes`
- You can get access to the `ActionBar` instance (which now points to the `ToolBar`), with the `getSupportActionBar` method
- For instance, you can hide the action bar like this:

```
getSupportActionBar().hide();
```

# Adding the "Up" Action to ShowRoute

- In the app's manifest, add a parentActivityName attribute to the ShowRoute activity tag, AND add a meta-data tag for the Up navigation to work on older APIs:

```
<activity
    android:name=".ShowRoute">
    android:parentActivityName=".Routes"
    <meta-data
          android:name="android.support.PARENT_ACTIVITY"
          android:value=".Routes" />
>
</activity>
```

- In the ShowRoute onCreate method, enable the Up button:

```
ActionBar ab = getSupportActionBar();
ab.setDisplayHomeAsUpEnabled(true);
```

# Setting up a callback for Up event

- In the ShowRoute activity, override `onOptionsItemSelected` method:

```
public boolean onOptionsItemSelected(MenuItem item) {
    return super.onOptionsItemSelected(item);

}
```

- This method is called back whenever an item is clicked on in the App Bar. For the special case of the Up navigation, the event is handled by the superclass

# Run the App and Enjoy!