MIT Accelerating Information Technology Innovation

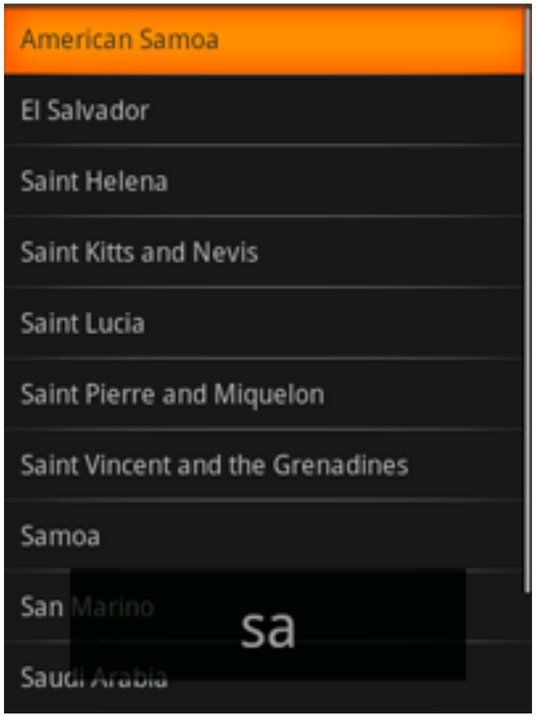
Android Lecture 2: Applications with Multiple Activities

http://aiti.mit.edu

Agenda

- More Views
- Data Binding
- Switching Activities
- Passing data between Activities
- The Activity lifecycle

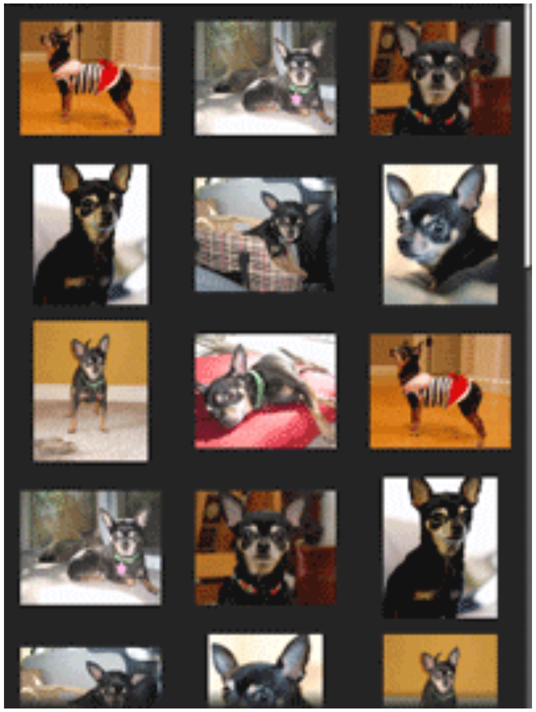
Views we haven’t yet learned



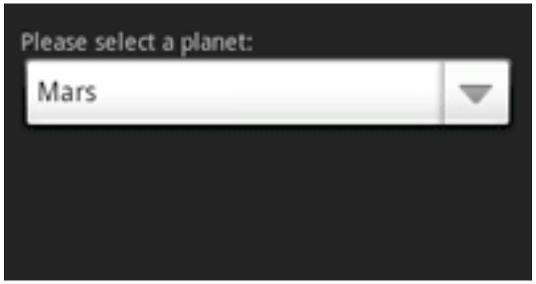
American Samoa
El Salvador
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Pierre and Miquelon
Saint Vincent and the Grenadines
Samoa
San Marino
Saudi Arabia

sa

ListView



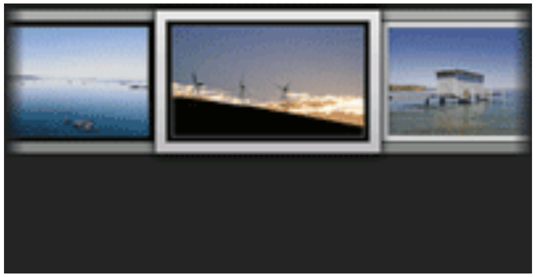
GridView



Please select a planet:

Mars

Spinner (drop-down)



Gallery

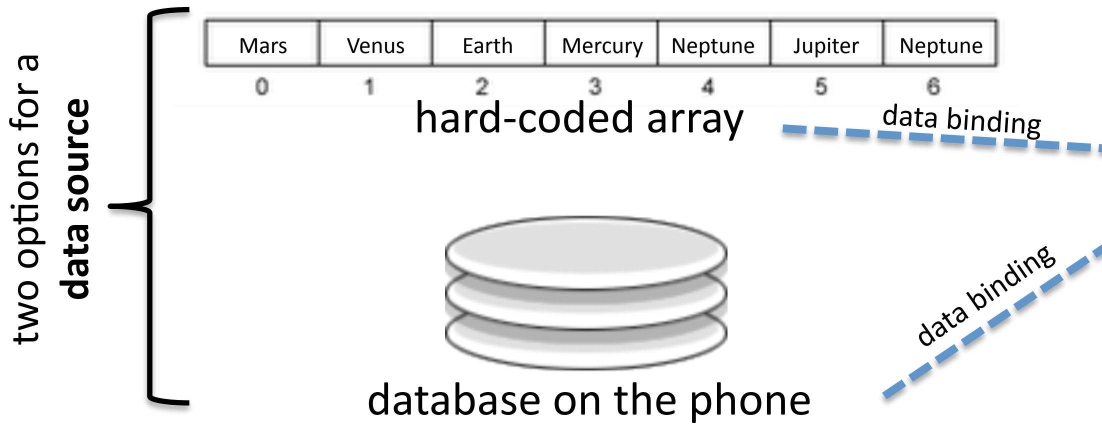
What do all of these have in common?

What do all those views have in common?

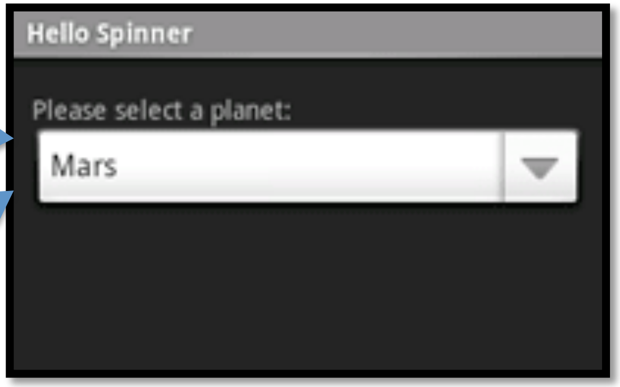
- All of them store and display **multiple** items!
  - A ListView displays items in a 1-D vertical, scrollable list
  - A GridView displays items in a 2-D, scrollable grid
  - A Spinner displays items in 1-D vertical *drop-down* component.
  - A Gallery displays items in a 1-D, horizontal, scrollable list.
- How do we provide multiple items to these views?
  - Use Data Binding

Importance of Data Binding

- **Data Binding**: the process of connecting views that display multiple items to a **data source**
- Any modifications to the data source will be reflected on the view immediately and automatically.



Mars	Venus	Earth	Mercury	Neptune	Jupiter	Neptune
0	1	2	3	4	5	6



Hello Spinner

Please select a planet:

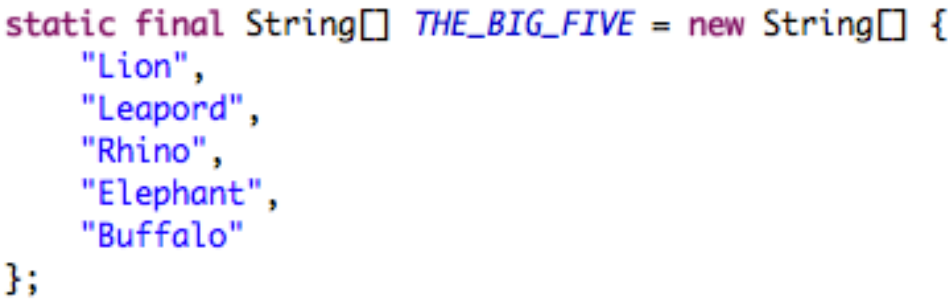
Mars

Possible Data Sources

- Data can be fetched from multiple sources:
  - Hard-coded arrays, defined in code
  - XML Resource Files
  - Databases on the phone
  - Content Providers / Content Resolvers (e.g. to populate a ListView with all the contacts on your phone)

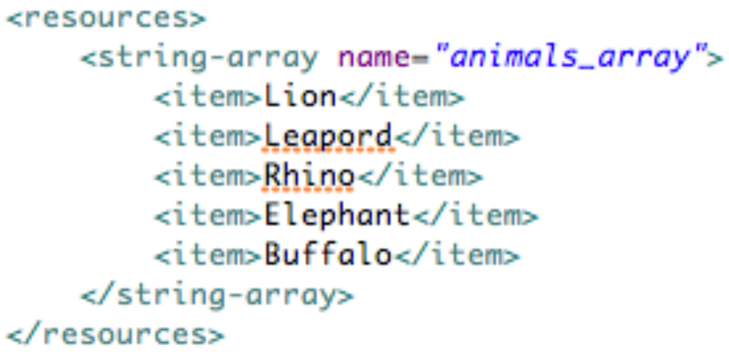
Example: ListView with an array data source

- Step 1: Create a class of type ListActivitiy (as opposed to Activity)
- Step 2: Create the array data source. Two ways of doing this:
  - Hard-code the array in the ListActivity Class :



```
static final String[] THE_BIG_FIVE = new String[] {
    "Lion",
    "Leapord",
    "Rhino",
    "Elephant",
    "Buffalo"
};
```

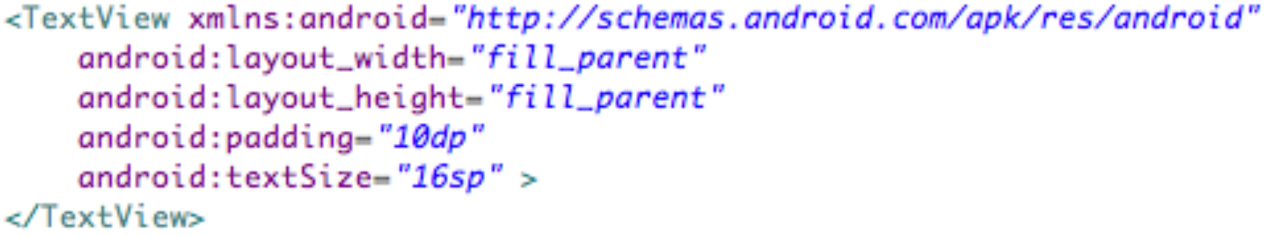
  - Define the array in as an XML resource. Add the <string-array> to **`res/values/strings.xml`**



```
<resources>
    <string-array name="animals_array">
        <item>Lion</item>
        <item>Leapord</item>
        <item>Rhino</item>
        <item>Elephant</item>
        <item>Buffalo</item>
    </string-array>
</resources>
```

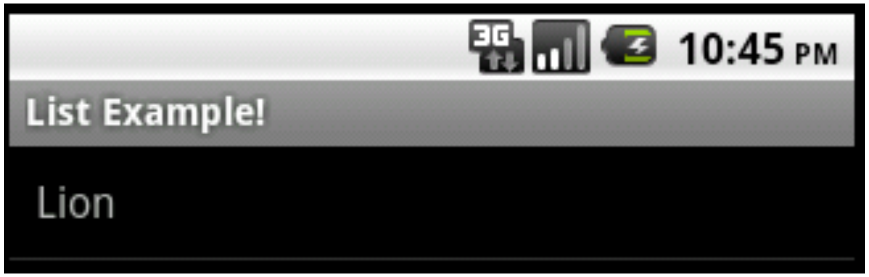
ListView Example, continued…

- Step 3: Create an XML layout file that will define how each cell or item in the ListView will look. Call this file “list_item.xml” and add to **`res/layout/`**



```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="16sp" >
</TextView>
```

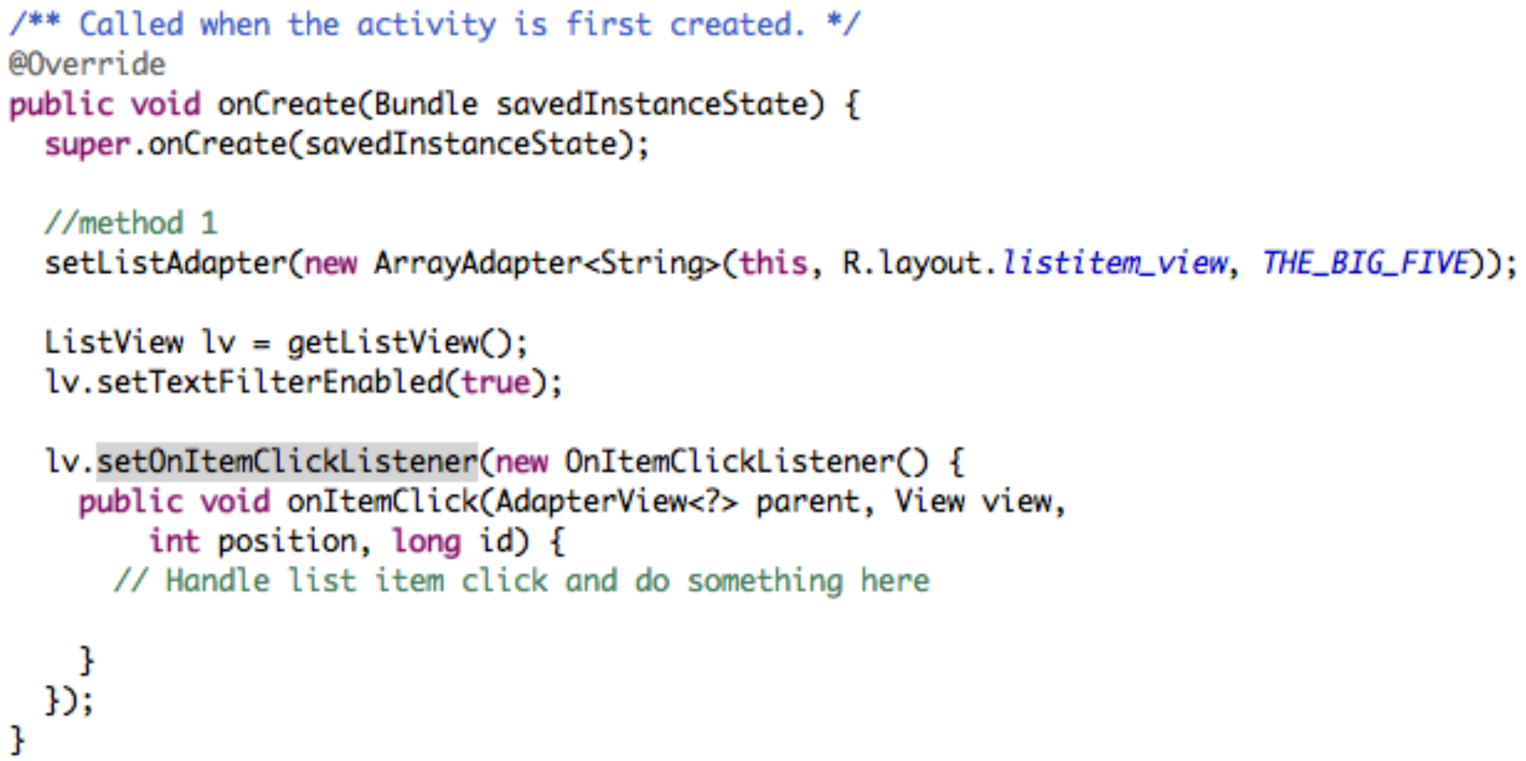
Note: This XML code means that each list item will essentially be a TextView, i.e. a simple text label. If we wanted each list item to also show an icon, we would need to modify this xml file to also include an ImageIcon and a layout of some sort.



Each list item, e.g. “Lion”, is simply a TextView

ListView Example, continued…

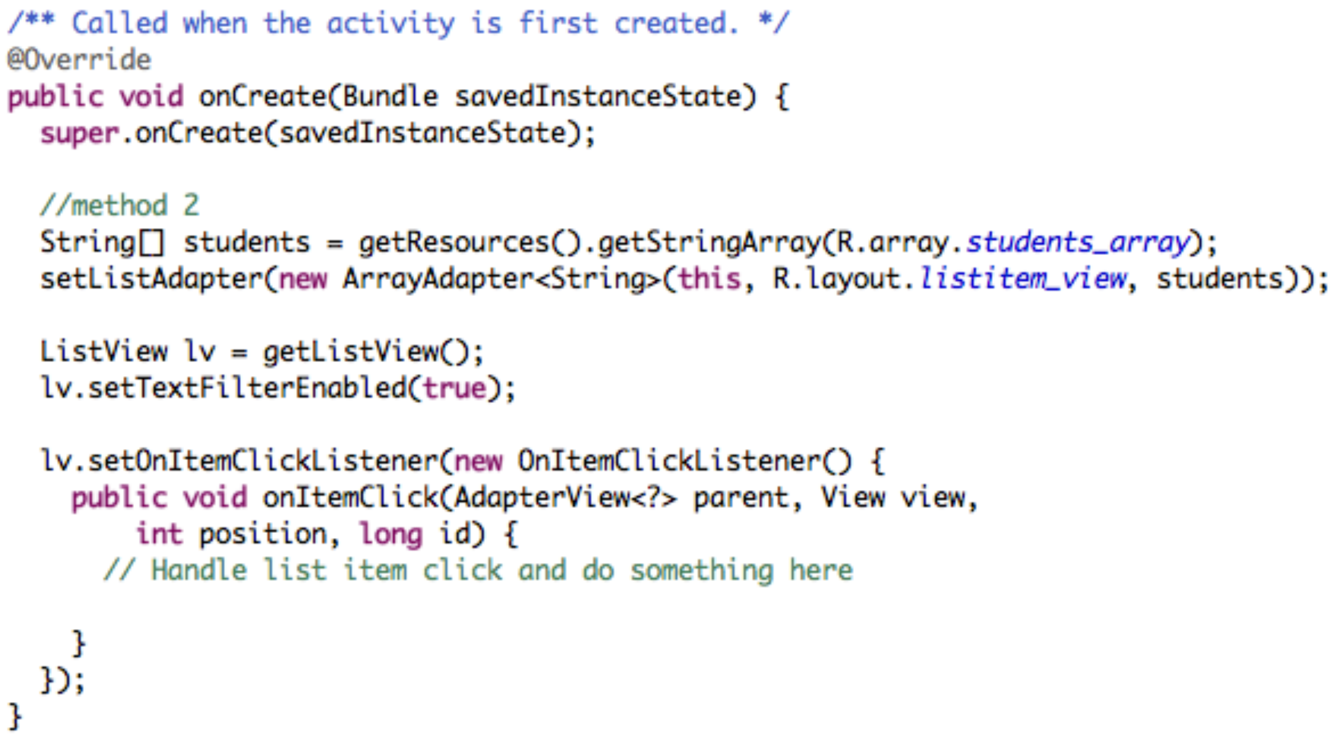
- Step 4: Now, establish the data binding in the onCreate() method of the ListActivity class
  - If data source is a hard-coded array, use the following:



```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //method 1
  setListAdapter(new ArrayAdapter<String>(this, R.layout.listitem_view, THE_BIG_FIVE));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // Handle list item click and do something here

    }
  });
}
```

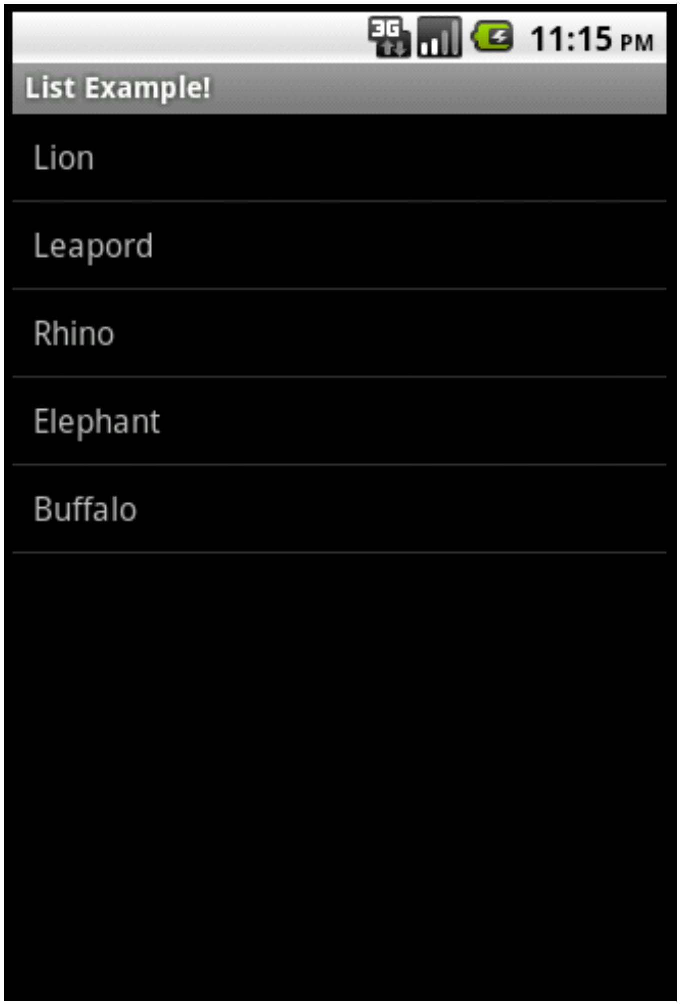
ListView Example, continued…

- Step 4 contd…
  - However, If data source is defined in an XML resource file, use the following:



```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //method 2
  String[] students = getResources().getStringArray(R.array.students_array);
  setListAdapter(new ArrayAdapter<String>(this, R.layout.listitem_view, students));

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);

  lv.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view,
        int position, long id) {
      // Handle list item click and do something here

    }
  });
}
```

The End Result!



List Example!

Lion

Leapord

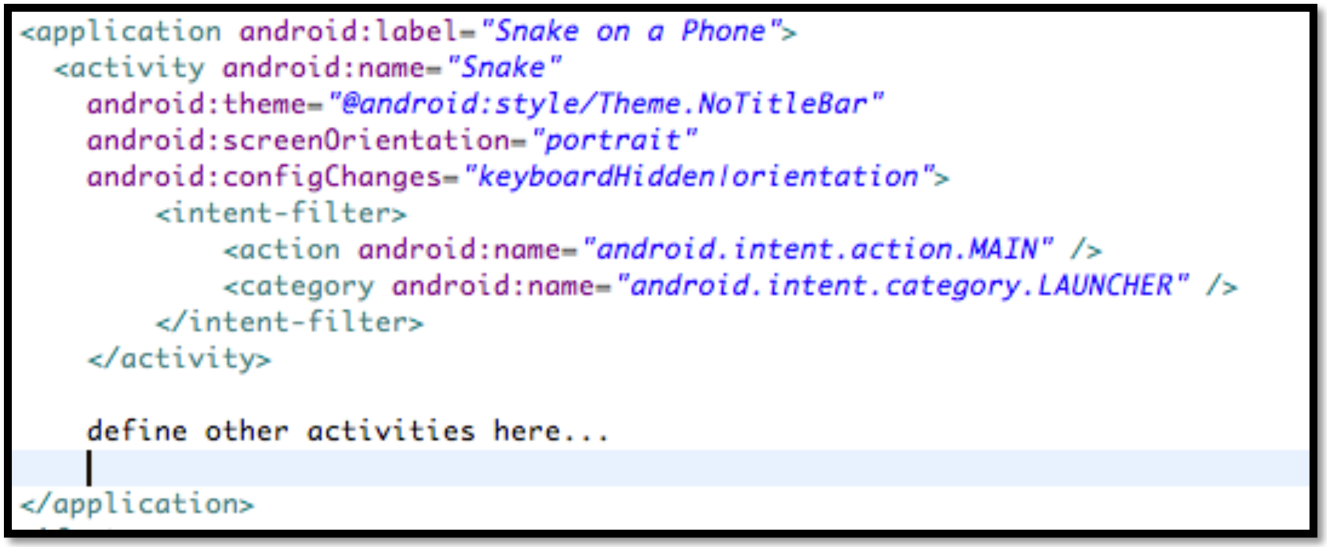
Rhino

Elephant

Buffalo

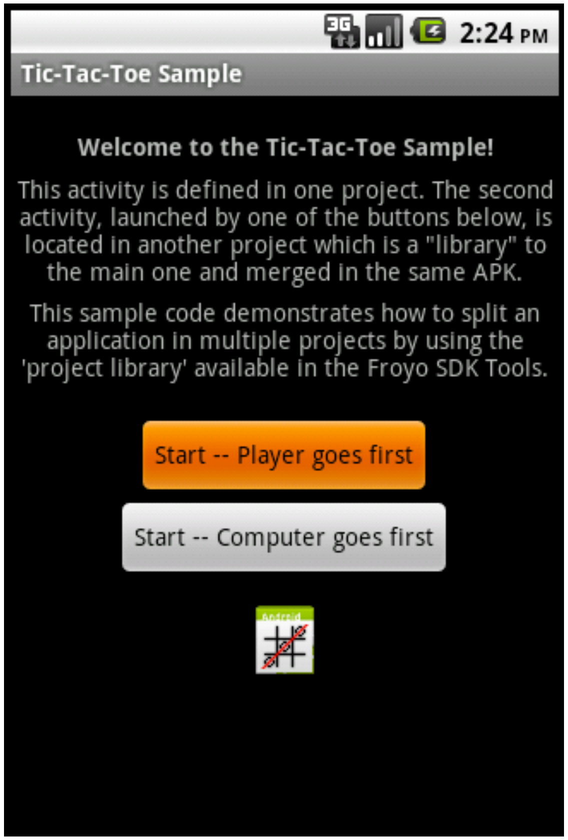
Multiple Activities

- An android application consists of multiple Activity objects
- Each Activity is like one “page” of the app
- Only one activity can be the *main* activity



```
<application android:label="Snake on a Phone">
  <activity android:name="Snake"
    android:theme="@android:style/Theme.NoTitleBar"
    android:screenOrientation="portrait"
    android:configChanges="keyboardHidden|orientation">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    define other activities here...

</application>
```

Multiple Activities, example:

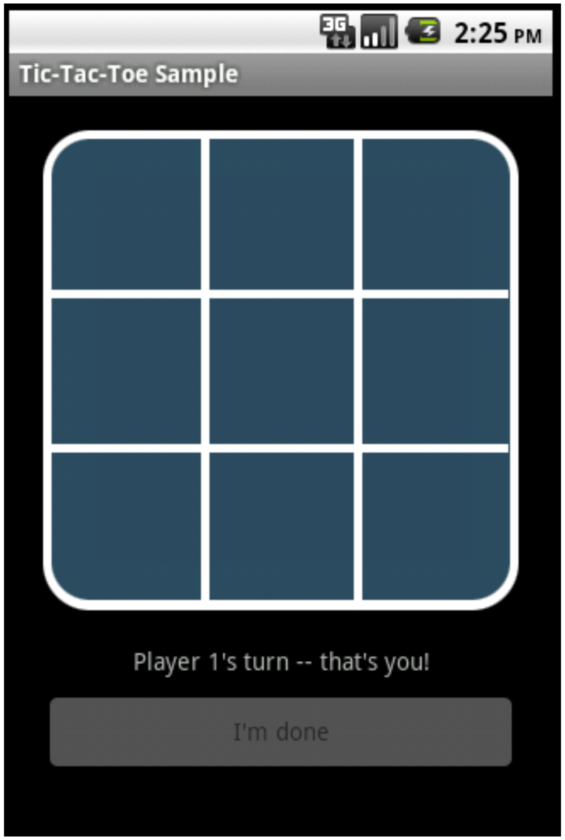


Welcome to the Tic-Tac-Toe Sample!

This activity is defined in one project. The second activity, launched by one of the buttons below, is located in another project which is a "library" to the main one and merged in the same APK.

This sample code demonstrates how to split an application in multiple projects by using the 'project library' available in the Froyo SDK Tools.

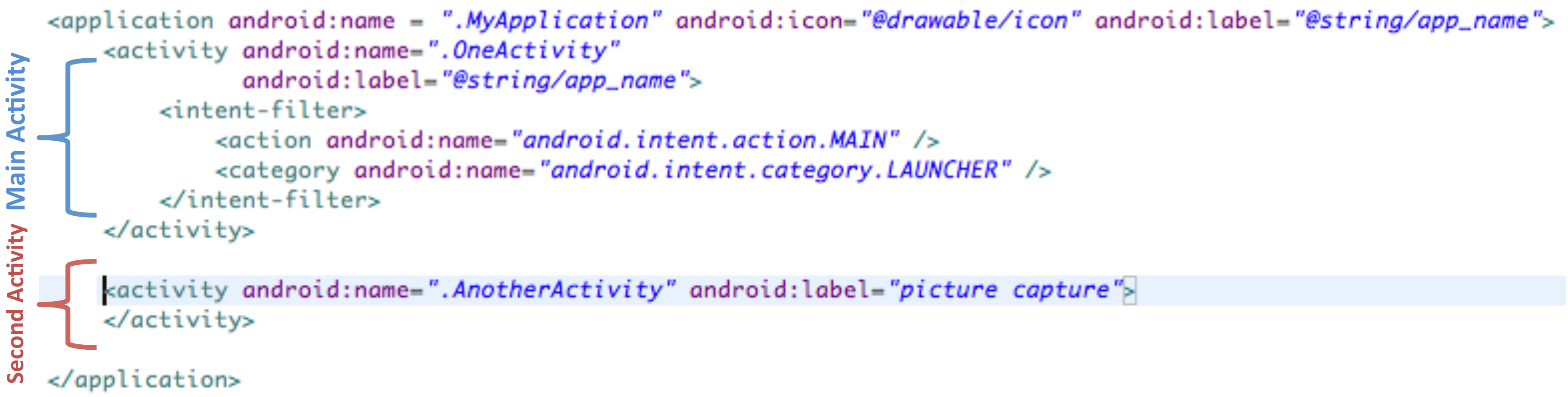
Main Activity (first thing you see when App starts)



Second Activity (clicking on a button on the Main Activity brings user to this one)

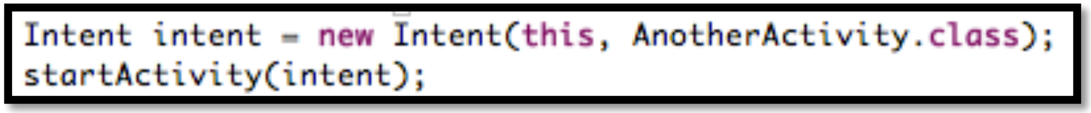
Switching between Activities

Step 1: Define all Activities in your App in the AndroidManifest.xml file



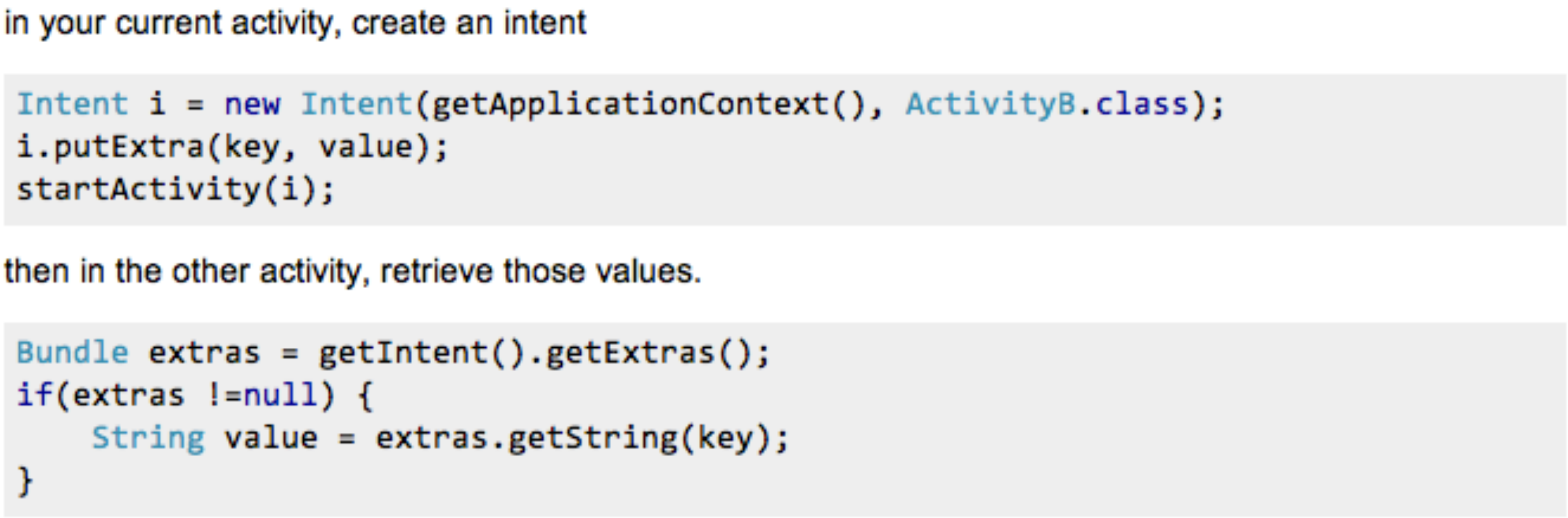
```
<application android:name = ".MyApplication" android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".OneActivity"
              android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name=".AnotherActivity" android:label="picture capture">
    </activity>

</application>
```

Step 2: Switch from Main Activity to the activity defined in **AnotherActivity.class**, using Intent objects.



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

Passing data between Activities



in your current activity, create an intent

```
Intent i = new Intent(getApplicationContext(), ActivityB.class);
i.putExtra(key, value);
startActivity(i);
```

then in the other activity, retrieve those values.

```
Bundle extras = getIntent().getExtras();
if(extras !=null) {
    String value = extras.getString(key);
}
```

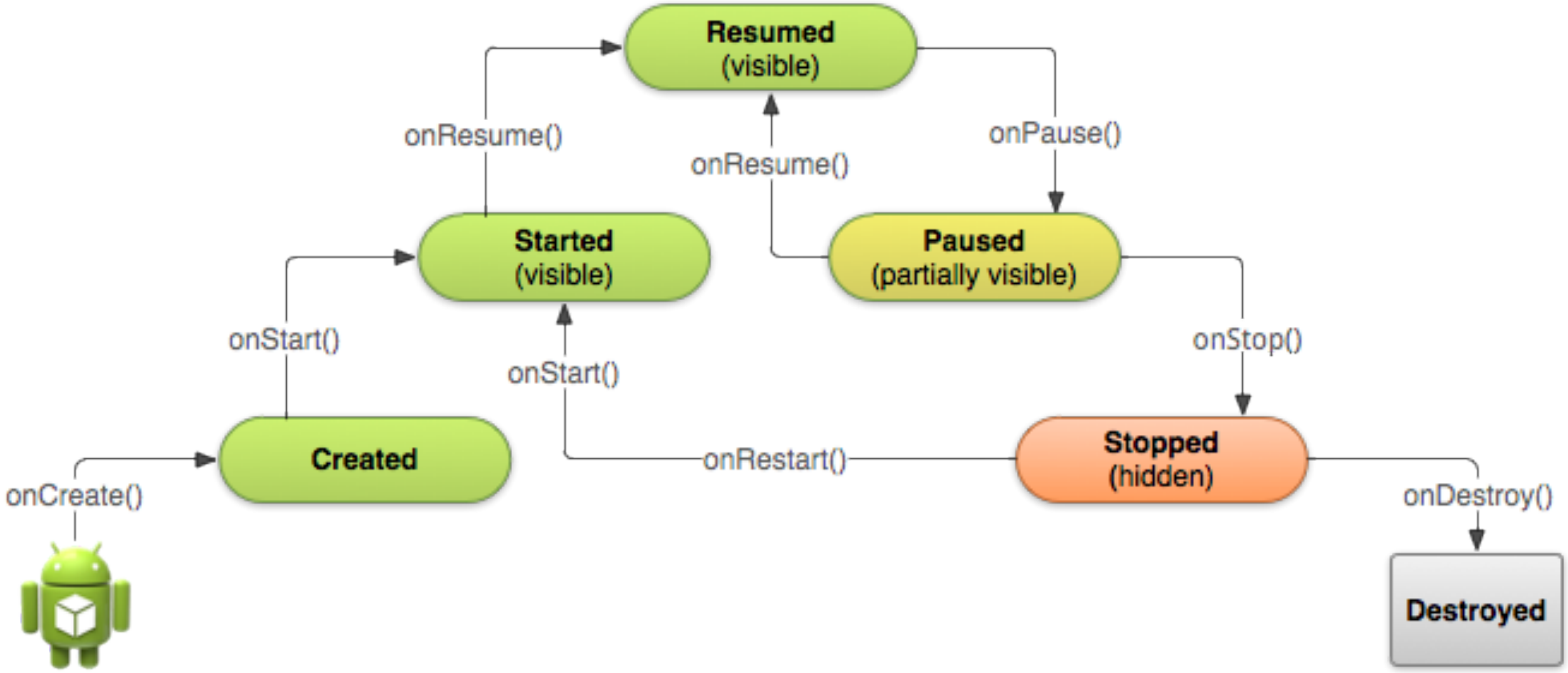
Note: you can use the putExtra method to add data in key value pairs to the Intent. The key must be a String object but the value can be any of the following: integer, integer[], float, float[], double, double[], String, String[], etc… primitive data types.

Then, you fetch that data in the second activity using the .getExtras().getString(key) approach.

Other ways to exchange data between Activities

- Intent approach is best for *primitive* data types that don't need to last forever (i.e. they are *not persistent*)
- For *primitive* types that need to last forever (i.e. *persistent* objects), use Preferences
- For *non-primitive* types that are *not persistent*:
  - Public Static Fields
  - Maintain global application state in the Application class (all Activity objects have access to this).
- For non-primitive types that are *persistent*:
  - Use ContentProvider, SQL Database on the phone, Files, etc.

The Activity lifecycle



Activity States

- Resumed
  - Activity is currently active
- Paused
  - Activity is partially hidden (obscured by menu, dialog box, etc.)
- Stopped
  - Activity is hidden (new Activity started, user changed apps, etc.)

Lifecycle Callback Methods

- onCreate()
  - Initialize anything that should happen only once
  - Define UI, init class variables, etc.
- onDestroy()
  - Called when your Activity is destroyed permanently (killed by the system)
  - Usually, most resources should be cleaned up before onDestroy() in onStop()

Lifecycle Callback Methods (cont.)

- onPause()
  - Called when you app is interrupted
  - Stop expensive CPU operations, pause video playback, release system resources
- onResume()
  - Called every time your Activity comes into view
  - Initialize components released in onPause()

Lifecycle Callback Methods (cont.)

- onStart()
  - Called every time your app comes back into view
  - (Re)start important processes like a remote connection
- onStop()
  - Called when new Activity or App is switched to
  - Write data to database, close potential memory leaks