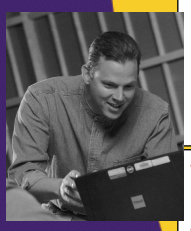
Widgets: Spinners (Combo Boxes)

Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/android-tutorial/

For live Android training, please see courses at http://courses.coreservlets.com/.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this Android tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - Android development, JSF 2, servlets/JSP, Ajax, jQuery, Java 6 programming, custom mix of topics
  - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, etc.) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, GWT, RESTful and SOAP-based Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- Switching from one Activity to another
- Spinners with choices set in XML
- Spinners with choices set in Java

General Approach for Widget Examples

Widget Lectures Combined in Single Project

- **Main screen**
  - Lets user choose screens on various Widget topics
- **Other screens**
  - Correspond to separate lectures.
    - One screen for lecture on Buttons, another for lecture on Spinners, another for number input, etc.
- **Separate layout files**
  - main.xml, buttons.xml, spinners.xml, etc. See next slide.
- **Separate Java classes**
  - WidgetActivity.java, ButtonActivity.java, SpinnerActivity.java, etc.
- **Shared strings file**
  - strings.xml has separate sections for each lecture, but same file

Layout Files for Widget Lectures

- **Separate layout files for each Activity**
  - res/layout/main.xml
    - Gives layout for main screen. Loaded with setContentView(R.layout.main);
  - res/layout/buttons.xml
    - Gives layout for screen on Button and related Widgets. Loaded with setContentView(R.layout.buttons);
  - res/layout/spinners.xml
    - Gives layout for screen on Spinners (i.e., combo boxes). Loaded with setContentView(R.layout.spinners);
- **Two common layout attributes**
  - android:layout_width, android:layout_height
    - match_parent (fill up space in enclosing View)
    - wrap_content (use natural size)

Strings File for Widget Lectures (res/values/strings.xml)

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

    <!--  Initial screen -->
    <string name="app_name">...</string>
    <string name="show_buttons_button_label">...</string>
    <string name="show_spinners_button_label">...</string>

    <!-- Buttons example -->
    <!-- Shown in earlier lecture -->

    <!-- Spinners example -->
    <!-- Shown in this lecture -->

    ...

</resources>
```


Switching Activities: Summary

- **Switches Activities with Intents**
  - Main screen has buttons to navigate to other Activities
  - Return to original screen with phone's "back" button
- **Syntax required to start new Activity**
  - Java
    - Intent newActivity = new Intent(this, NewActivity.class);
    - startActivity(newActivity);
  - XML
    - Requires entry in AndroidManifest.xml (which is part of downloadable Eclipse project for Widgets)
  - More details
    - Code shown on next few slides
    - Even more information given in later lecture on Intents

Switching Activities: Details

- **Java (InitialActivity.java)**

```
Intent newActivity = new Intent(this, NewActivity.class);
startActivity(newActivity);
```

- **XML (AndroidManifest.xml)**

```
<activity android:name=".NewActivity"
          android:label="@string/new_app_name">
  <intent-filter>
    <action
        android:name="android.intent.action.VIEW" />
    <category
        android:name="android.intent.category.DEFAULT" />
  </intent-filter>
</activity>
```

The intent-filter part stays the same. Just copy and paste.

Switching Activities: WidgetsInitialActivity.java

```
public class WidgetsInitialActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    private void goToActivity
                (Class<? extends Activity> activityClass) {
        Intent newActivity = new Intent(this, activityClass);
        startActivity(newActivity);
    }

    public void showSpinners(View clickedButton) {
        goToActivity(SpinnerActivity.class);
    }

    ...
}
```

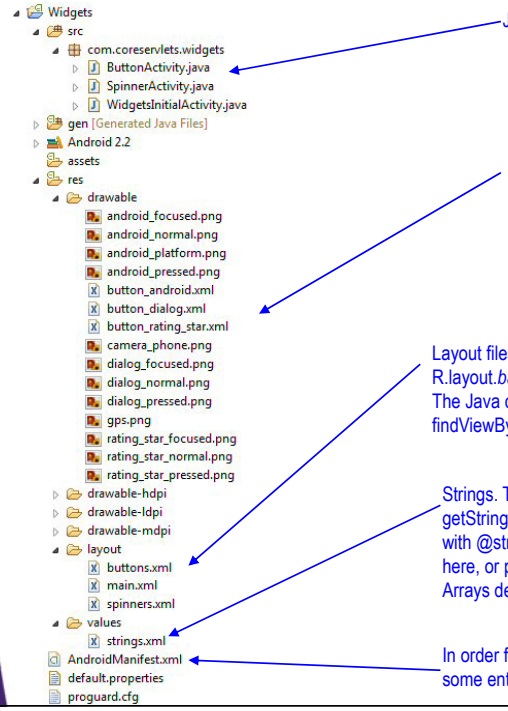
If you have never seen wildcards in generics before, this just means that I will pass in a subclass of Activity (as with SpinnerActivity.class at bottom).

Switching Activities: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.coreservlets.widgets"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".WidgetsInitialActivity"
                  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=".SpinnerActivity"
                  android:label="@string/spinner_app_name">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Most parts of this file were created automatically when the Android project was made in Eclipse. To switch Activities yourself, cut and paste this code from the downloadable source, and change only android:name and android:label.

Overall Widget Project Layout



Java code

Images and XML files that refer to sets of images. The layout files will refer to these images via @drawable/*base_file_name* (e.g., @drawable/gps). See ImageButton examples in lecture on buttons.

Layout files. The Java code will refer to the overall layouts via R.layout.*base_file_name* (R.layout.main, R.layout.spinners, etc.). The Java code will refer to specific GUI elements with findViewById(R.id.*element_id*).

Strings. The Java code will refer to these via getString(R.string.*string_name*). The layout files will refer to these with @string/*string_name*. You can also define arrays of strings here, or put the arrays in a separate file typically called arrays.xml. Arrays defined here are used in the first Spinner example.

In order for one Activity to start another Activity in the same project, you need some entries in here. See upcoming slide.

Spinner Approach 1: Choices Specified in XML

Spinner with Predefined Choices

- **Idea**
  - A combo box (drop down list of choices)
    - Similar purpose to a RadioGroup: to let the user choose among a fixed set of options
- **Main Listener types**
  - AdapterView.OnItemSelectedListener
  - AdapterView.OnItemClickedListener
    - The first is more general purpose, since it will be invoked on programmatic changes and keyboard events as well as clicks.

Spinner (Continued)

- **Key XML attributes**
  - android:id
    - You need a Java reference to assign an event handler
  - android:prompt
    - The text shown at the top of Spinner when user clicks to open it.
      - Since text is *not* shown when the Spinner is closed, the string used for the prompt is typically also displayed in a TextView above the Spinner.
  - android:entries
    - An XML entry defining an array of choices.
      Can be in strings.xml or a separate file (e.g., arrays.xml)

```
<string-array name="some_name">
    <item>choice 1</item>
    <item>choice 2</item>
    …
</string-array>
```


OnItemSelectedListener

- **onItemSelected**
  - Invoked when the an entry is selected. Invoked once when Spinner is first displayed, then again for each time the user selects something.
  - Arguments
    - AdapterView: the Spinner itself
    - View: the row of the Spinner that was selected
    - int: the index of the selection. Pass this to the Spinner's getItemAtPosition method to get the text of the selection.
    - long: The row id of the selected item
- **onNothingSelected**
  - Invoked when there is now nothing displayed. This cannot happen due to normal user interaction, but only when you programmatically remove an entry.

XML: Layout File Entry (Part of res/layout/spinners.xml)

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/spinner1_prompt" />
<Spinner
    android:id="@+id/spinner1"
    android:prompt="@string/spinner1_prompt"
    android:entries="@array/spinner1_entries"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Same text used twice, since the text is hidden when the Spinner is closed.

An array of entries. If you have lots of arrays, you typically put them in arrays.xml. However, here, it makes more sense to keep the array of entries in strings.xml with the spinner prompt and the spinner message template.

XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="spinner1_prompt">
   Current Android Vendors (Choices from XML)
</string>
<string-array name="spinner1_entries">
    <item>Acer</item>
    <item>Dell</item>
    <item>HTC</item>
    <item>Huawei</item>
    <item>Kyocera</item>
    <item>LG</item>
    <item>Motorola</item>
    <item>Nexus</item>
    <item>Samsung</item>
    <item>Sony Ericsson</item>
    <item>T-Mobile</item>
    <item>Neptune</item>
</string-array>
<string name="spinner_message_template">
    You selected \'%s\'.
</string>
```

The event handler method will use String.format, this template, and the current selection to produce a message that will be shown in a Toast when a Spinner selection is made.

Java (Relevant Parts)

```
public class SpinnerActivity extends Activity {
    private String mItemSelectedMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.spinners);
        mItemSelectedMessageTemplate =
                getString(R.string.spinner_message_template);
        Spinner spinner1 = (Spinner)findViewById(R.id.spinner1);
        spinner1.setOnItemSelectedListener(new SpinnerInfo());
        ...  // Code for spinner2 shown later
    }

    private void showToast(String text) {
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    // Continued on next slide with the SpinnerInfo inner class
```


Java (Relevant Parts, Continued)

```
    private class SpinnerInfo implements OnItemSelectedListener {
        private boolean isFirst = true;

       @Override
        public void onItemSelected(AdapterView<?> spinner, View selectedView,
                                   int selectedIndex, long id) {
            if (isFirst) {
                isFirst = false;
            } else {
                String selection =
                        spinner.getItemAtPosition(selectedIndex).toString();
                String message =
                        String.format(mItemSelectedMessageTemplate, selection);
                showToast(message);
            }
        }

        @Override
        public void onNothingSelected(AdapterView<?> spinner) {
            // Won't be invoked unless you programmatically remove entries
        }
    }
```

Don't want the Toast when the screen is first displayed, so ignore the first call to onItemSelected. Other calls are due to user interaction.

Results (Emulator)

Spinner Approach 2: Choices Specified in Java

Spinner with Choices Computed by Java Code

- **Idea**
  - A combo box (drop down list of choices)
    - Same general purpose as previous example. However, here you want to programmatically compute the options to be displayed, possibly based on earlier user interaction.
- **Main Listener types**
  - AdapterView.OnItemSelectedListener
  - AdapterView.OnItemClickedListener
    - These are same as in previous Spinner example

Spinner (Continued)

- **Key XML attributes**
  - android:id
    - You need a Java reference to specify the entries and to assign an event handler.
  - android:prompt
    - The text shown at the top of Spinner when user clicks to open it.
      - Since this text is *not* shown when the Spinner is closed, the string used for the prompt is typically also displayed in a TextView above the Spinner.
  - android:entries
    - Not used in this version. Java will compute the entries.

Creating Spinner Entries Programmatically

- **Get reference to the Spinner**

```
Spinner spinner = (Spinner)findViewById(R.id.spinner_id);
```

- **Make an ArrayAdapter**

```
List<String> entries = …; // Can also use String[]
ArrayAdapter<String> spinnerAdapter =
   new ArrayAdapter<String>(this,
                  android.R.layout.simple_spinner_item,
                  entries);
```

Predefined entry in Android distribution

- **Specify the drop down View resource**

```
spinnerAdapter.setDropDownViewResource
      (android.R.layout.simple_spinner_dropdown_item);
```

- **Set the adapter for the Spinner**

```
spinner.setAdapter(spinnerAdapter);
```


XML: Layout File Entry (Part of res/layout/spinners.xml)

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/spinner2_prompt" />
<Spinner
    android:id="@+id/spinner2"
    android:prompt="@string/spinner2_prompt"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Same text used twice, since the text is hidden when the Spinner is closed.

android:entries is *not* used. Instead of having fixed choices, the Java code will compute the options.

XML: Strings File Entries (Part of res/values/strings.xml)

```
<string name="spinner2_prompt">
    Future Android Vendors (Choices from Java)
</string>
```


Java (Relevant Parts)

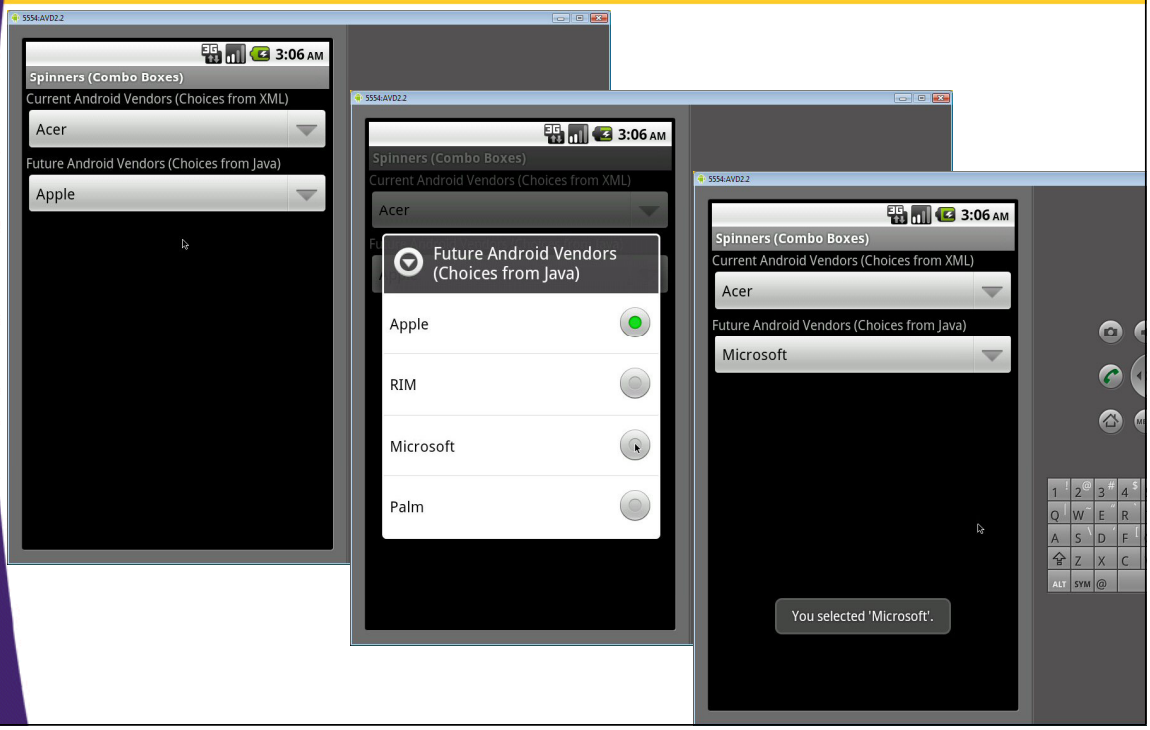
```
public class SpinnerActivity extends Activity {
    private String mItemSelectedMessageTemplate;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        // General code and code for spinner1 shown earlier
        List<String> futureAndroidVendors =
                getFutureAndroidVendors();
        ArrayAdapter<String> spinner2Adapter =
                new ArrayAdapter<String>(this,
                                android.R.layout.simple_spinner_item,
                                futureAndroidVendors);
        spinner2Adapter.setDropDownViewResource
             (android.R.layout.simple_spinner_dropdown_item);
        spinner2.setAdapter(spinner2Adapter);
        spinner2.setOnItemSelectedListener(new SpinnerInfo());
    }
```

Java (Relevant Parts, Continued)

```
private List<String> getFutureAndroidVendors() {
    String[] vendorArray = { "Apple", "RIM",
                             "Palm", "Microsoft" };
    List<String> vendorList = Arrays.asList(vendorArray);
    Collections.shuffle(vendorList);
    return(vendorList);
}
```

The last argument to the ArrayAdapter<String> constructor on previous page can be any List<String> or String[]. I am randomizing the order of the elements to demonstrate that you can have Java compute the entries instead of having a fixed set of choiuces (in which case you would define the entries in the XML file as with approach 1).

Results (Emulator)



Wrap-Up

Summary

- **Spinner with fixed entries**
  - Define array in strings.xml.
  - Use android:prompt and android:entries in layout file. Also assign id with android:id
  - Java gets ref and calls setOnItemSelectedListener
- **Spinner with computed entries**
  - XML uses android:prompt and android:id
  - Java gets ref, makes ArrayAdapter with a List<String> or String[], uses some predefined resource names
- **Switching Activities**
  - Intent newActivity = new Intent(this, NewActivity.class);
  - startActivity(newActivity);
  - Also requires entry in AndroidManifest.xml

Questions?