Application Resources

Sang Shin
Michèle Garoche
www.javapassion.com
“Learn with Passion!”

Disclaimer

- Portions of this presentation are modifications based on work created and shared by the Android Open Source Project
  - http://code.google.com/policies.html
- They are used according to terms described in the Creative Commons 2.5 Attribution License
  - http://creativecommons.org/licenses/by/2.5/

Topics

- What is a resource?
- Externalizing resources
- Default vs. Alternative resources
- Providing resources
- Providing alternative resources
- Accessing resources
  - in Code
  - in XML
- Localization

What is a Resource?

What is a Resource?

- Any static data that can be externalized from code
  - Layouts
  - Strings
  - Images
  - Video and audio files
  - Menu definitions
  - Animation
  - etc.

Externalizing Resources

Why Externalizing Resources?

- Allows you to maintain them independently from code
- Allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes
  - Increasingly becoming important as more Android-powered devices become available with different configurations.

Default vs. Alternative Resources

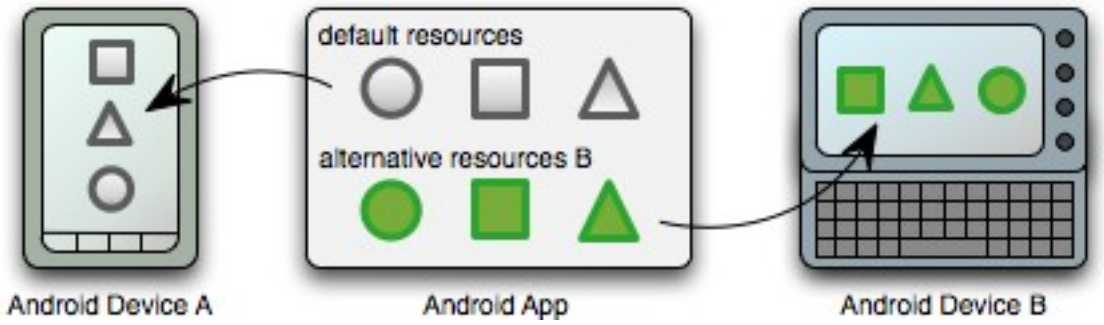
Default vs. Alternative Resources

- For any type of resource, you can specify default and multiple alternative resources for your application
- Default resources are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration
- Alternative resources are those that you've designed for use with a specific configuration.

Default vs. Alternative Resources

- Two device configurations, both using default resources

- Two device configurations, one using alternative resources.



Providing Resources

Grouping Resources under /res directory

- You should place each type of resource in a specific subdirectory of your project's *res/* directory. For example, here's the file hierarchy for a simple project:

```
Myproject/
    src/
        MyActivity.java
    res/
        drawable/
            icon.png
        layout/
            main.xml
            info.xml
        values/
            strings.xml
```

Resource Sub-directories under /res

- layout/
- values/
- anim/
  - XML files that define tween animations
- color/
- drawable/
- menu/
- raw/
- xml/

Providing Alternative Resources

Why Alternative Resources?

- Almost every application should provide alternative resources to support specific device configurations.
  - Alternative drawable resources for different screen densities
  - Alternative string resources for different languages.
- At runtime, Android automatically detects the current device configuration and loads the appropriate resources

How to specify Alternative Resources?

- Create a new directory in res/ named in the form <resources_name>-<config_qualifier>
  - <resources_name> is the directory name of the corresponding default resources
  - <config_qualifier> is a name that specifies a configuration for which these resources are to be used
- Save your alternative resources in this new directory
  - The alternative resource files must be named exactly the same as the default resource files.

Example: Default & Alternative Resources

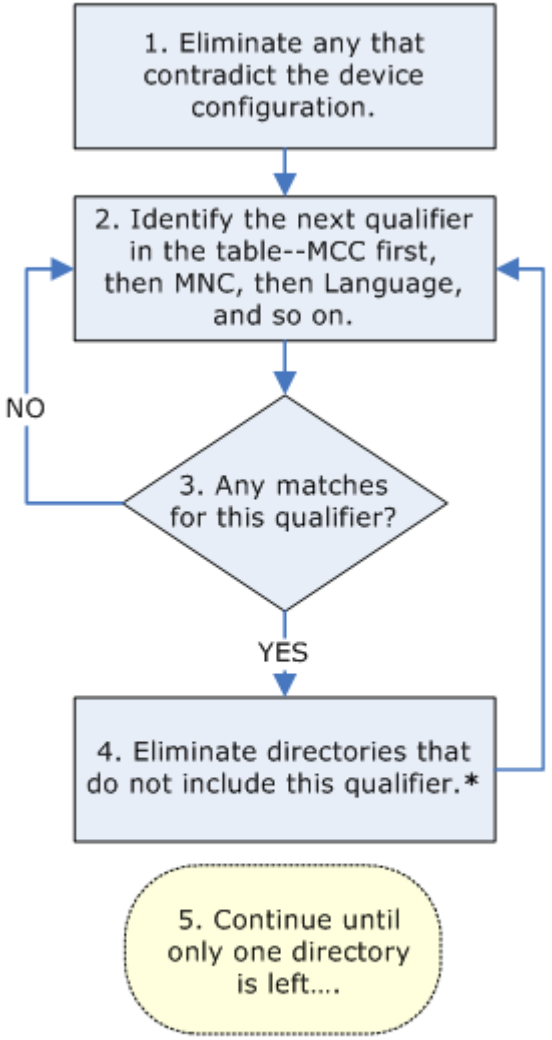
res/
drawable/
icon.png
background.png
drawable-hdpi/
icon.png
background.png

- The *hdpi* qualifier indicates that the resources in that directory are for devices with a high-density screen.
- While the images in each *drawable* directory are sized for a specific screen density, the filenames are the same.

Alternative Resource Qualifier Names

- Language and region
  - en, fr, en-rUS, fr-rFR, fr-rCA
- Screen size
  - small, normal, large
- Screen orientation
  - port, land
- Screen density
  - ldpi, mdpi, hdpi, nodpi
- Mobile country code (MCC)
  - mcc310, mcc310-mnc004

How Android Finds Best-matching Resource



* If the qualifier is the screen density, Android selects a "best" match and the process is done.

MCC: Mobile Country Code
MNC: Mobile Network Code

Accessing Resources

Resource ID

- Once you provide a resource in your application, you can apply it by referencing its resource ID.
  - All resource IDs are defined in your project's R class, which the *aapt* tool automatically generates.
- For each type of resource, there is an R subclass
  - *R.drawable* for all *drawable* resources

Two ways to access a resource

- In code: Using an static integer from a sub-class of your R class, such as:
  - *R.string.hello* (string is the resource type and hello is the resource name)
- In XML: Using a special XML syntax that also corresponds to the resource ID defined in your R class, such as:
  - *@string/hello* (string is the resource type and hello is the resource name)

Accessing Resources: In Code

Accessing Resources In Code

- You can use a resource in code by passing the resource ID as a method parameter.
- For example, you can set an ImageView to use the *res/drawable/myimage.png* resource using setImageResource():

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

Example: Accessing Resources in Code

```
// Load a background for the current screen from a drawable resource
getWindow().setBackgroundDrawableResource(R.drawable.my_background_image) ;

// Set the Activity title by getting a string from the Resources object, because
//  this method requires a CharSequence rather than a resource ID
getWindow().setTitle(getResources().getText(R.string.main_title));

// Load a custom layout for the current screen
setContentView(R.layout.main_screen);

// Set a slide in animation by getting an Animation from the Resources object
mFlipper.setInAnimation(AnimationUtils.loadAnimation(this,
        R.anim.hyperspace_in));

// Set the text on a TextView object using a resource ID
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello_message);
```

Accessing Android Platform Resources

- Android contains a number of built-in resources, such as styles, themes, and layouts. To access these resource, qualify your resource reference with the android package name.

```
// Android provides a layout resource you can use for list items in a ListAdapter.
// In this example, simple_list_item_1 is a layout resource defined by the
// platform for items in a ListView. You can use this instead of creating your
// own layout for list items.
setListAdapter(
        new ArrayAdapter<String>(this,
                                 android.R.layout.simple_list_item_1,
                                 myarray));
```

Accessing Resources: In XML

Accessing Resources from XML

- You can define values for some XML attributes and elements using a reference to an existing resource.
- You will often do this when creating layout files, to supply strings and images for your widgets.

```
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/submit" />
```

Example: Accessing Resources from XML

- If you have the following resource file that includes a color resource and a string resource:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

- You can use these resources in the following layout file to set the text color and text string:

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="@string/hello" />
```

Example: Accessing Resources from XML

- To reference a system resource, you would need to include the package name.

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@android:color/secondary_text_dark"
    android:text="@string/hello" />
```

Localization

Need for Localization

- Your application should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your application will be used

Localization Guidelines

- Use the Android resource framework to separate the localized aspects of your application as much as possible from the code
  - Place most or all of the contents of your application's user interface into resource files
- The behavior of the user interface, on the other hand, is driven by your Java code.
  - For example, if users input data that needs to be formatted or sorted differently depending on locale, then you would use Java to handle the data programmatically

Create Default Resources

- Put the application's default text in a file with the following location and name:
  - res/values/strings.xml (required directory)
  - The text strings in res/values/strings.xml should use the default language, which is the language that you expect most of your application's users to speak.
- Put language-specific resources in res/values-<language-code>/strings.xml file
  - res/values-fr/strings.xml
  - res/values-ja/strings.xml

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"