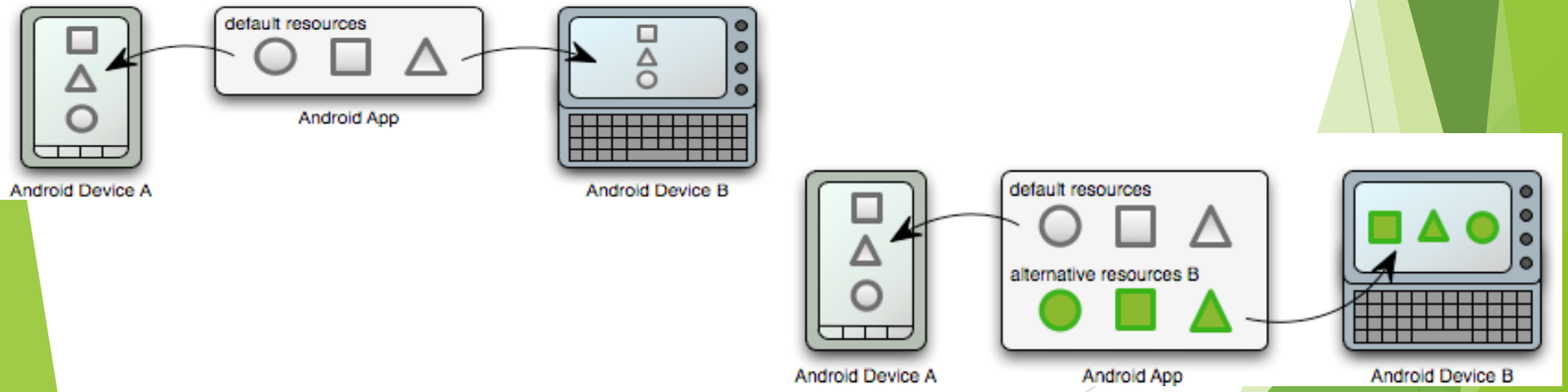
Resources & Assets

Externalizing Resources

- Android supports the externalization of resources ranging from simple values such as strings and colors to more complex resources like images (drawables), animations, and themes.
- By externalizing resources, they become easier to maintain, update, and manage.
  This also lets you easily define alternative resource values to support different hardware and internationalization.



Creating Resources

- Application resources are stored under the res/ folder of your project hierarchy.
  In this folder, each of the available resource types can have a subfolder containing its resources.
- 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
```

Creating Resources

- There are seven primary resource types that have different folders: simple values, drawables, layouts, animations, XML, styles, and raw resources.
- When your application is built, these resources will be compiled and included in your application package.
- This process also creates an R class file that contains references to each of the resources you include in your project.

Note: In all cases, the resource fi lenames should contain only lowercase letters, numbers, and the period (.) and underscore (_) symbols.

Creating Simple Values

- Supported simple values include strings, colors, dimensions, and string or integer arrays.
- All simple values are stored within XML files in the res/values folder.
- Within each XML fi le, you indicate the type of value being stored using tags.

Example – Simple Values

```
<?xml version=”1.0” encoding=”utf-8”?>
<resources>
    <string name=”Company”>SpymekTech Solutions Pvt. Ltd.</string>
    <color name=”app_background”>#FF0000FF</color>
    <dimen name=”default_border”>5px</dimen>
    <array name=”city”>
        <item>Pune</item>
        <item>Mumbai</item>
        <item>Nagpur</item>
    </array>
    <array name=”pin”>
        <item>411001</item>
        <item>400001</item>
        <item>421001</item>
    </array>
</resources>
```

Strings

- String resources are specifi ed using the string tag as shown in the following XML snippet:

  <string name=”stop_message”>Stop.</string>

- Android supports simple text styling, so you can use the HTML tags , <i>, and <u> to apply bold, italics, or underlining to parts of your text strings as shown in the example below:

  <string name=”stop_message”>Stop.</string>

- String.format
  - Can accept only those resource strings as an argument which doesn’t have any format tags associated.
- Instead use Html.fromHtml to method to convert this back into a styled character sequence.

Colors

- Use the color tag to define a new color resource.
- The color is specified with an RGB value and optional alpha channel.
- E.g.

```
<color name=”opaque_blue”>#00F</color>
<color name=”transparent_green”>#7700FF00</color>
```

Dimensions

- Dimensions are most commonly referenced within style and layout resources.
- To specify a dimension resource, use the dimen tag, specifying the dimension value, followed by an identifier describing the scale of your dimension:
  - px Screen pixels
  - in Physical inches
  - pt Physical points
  - mm Physical millimeters
  - dp Density-independent pixels relative to a 160-dpi screen
  - sp Scale-independent pixels

Dimension - Example

```
<?xml version="1.0" encoding="utf-8"?>
    <resources>
        <dimen name="textview_height">25dp</dimen>
        <dimen name="textview_width">150dp</dimen>
        <dimen name="ball_radius">30dp</dimen>
        <dimen name="font_size">16sp</dimen>
    </resources>
```

Drawables

- Drawable resources include bitmaps and NinePatch (stretchable PNG) images.
- They are stored as individual files in the res/drawable folder.
- The resource identifier for a bitmap resource is the lowercase filename without an extension.
- The preferred format for a bitmap resource is PNG, although JPG and GIF files are also supported.

Animations

- Android supports two types of animation.
  - Tweened animations can be used to rotate, move, stretch, and fade a View;
  - frame-by-frame animations to display a sequence of drawable images.

Animations - Tweened Animations

- Each tweened animation is stored in a separate XML fi le in the project's res/anim folder.
- An animation can be defined for changes in alpha (fading), scale (scaling), translate (moving), or rotate (rotating).
- Each of these animation types supports attributes to define what the sequence will do:

Alpha	fromAlpha and toAlpha	Float from 0 to 1
Scale	fromXScale/toXScale	Float from 0 to 1
	fromYScale/toYScale	Float from 0 to 1
	pivotX/pivotY	String of the percentage of graphic width/height from 0% to 100%
Translate	fromX/toX	Float from 0 to 1
	fromY/toY	Float from 0 to 1
Rotate	fromDegrees/toDegrees	Float from 0 to 360
	pivotX/pivotY	String of the percentage of graphic width/height from 0% to 100%

Animations - Tweened Animations

- You can create a combination of animations using the set tag. An animation set contains one or more animation transformations and supports various additional tags and attributes to customize when and how each animation within the set is run.
- The following list shows some of the set tags available:
  - **duration** Duration of the animation in milliseconds.
  - **startOffset** Millisecond delay before starting this animation.
  - **fillBefore** True to apply the animation transformation before it begins.
  - **fillAfter** True to apply the animation transformation after it begins.
  - **interpolator** Interpolator to set how the speed of this effect varies over time.

Animations - Tweened Animations - Example

```
<?xml version=”1.0” encoding=”utf-8”?>
<set xmlns:android=”http://schemas.android.com/apk/res/android”
android:interpolator=”@android:anim/accelerate_interpolator”>
<rotate
android:fromDegrees=”0” android:toDegrees=”360”
android:pivotX=”50%” android:pivotY=”50%”
android:startOffset=”500” android:duration=”1000” />
<scale
android:fromXScale=”1.0”  android:toXScale=”0.0”
android:fromYScale=”1.0”    android:toYScale=”0.0”
    android:pivotX=”50%” android:pivotY=”50%”  android:startOffset=”500”
    android:duration=”500” />
<alpha
android:fromAlpha=”1.0” android:toAlpha=”0.0”
android:startOffset=”500” android:duration=”500” />
</set>
```

Frame-by-Frame Animations

- Frame-by-frame animations let you create a sequence of drawables, each of which will be displayed for a specified duration, on the background of a View.
- Because frame-by-frame animations represent animated drawables, they are stored in the res/drawble folder.

Frame-by-Frame Animations - Example

```
<animation-list
xmlns:android=”http://schemas.android.com/apk/res/android”
android:oneshot=”false”>
<item android:drawable=”@drawable/rocket1” android:duration=”500”
    />
<item android:drawable=”@drawable/rocket2” android:duration=”500”
    />
<item android:drawable=”@drawable/rocket3” android:duration=”500”
    />
</animation-list>
```

Using Resources

- The resources can be used directly from your application code and can also be referenced from within other resources (e.g., a dimension resource might be referenced in a layout definition).

Using Resources in Code

- 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 example, R.drawable for all drawable resources)
  - for each resource of that type, there is a static integer (for example, R.drawable.icon).
- A resource ID is always composed of:
  - The *resource type*: Each resource is grouped into a "type," such as string, drawable, and layout.
  - The *resource name*, which is either: the filename, excluding the extension; or the value in the XML android:name attribute, if the resource is a simple value (such as a string).

Using Resources in Code

- There are two ways you can 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

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);
```

- you can retrieve resources using methods in Resources. You can get an instance of Resources with Context.getResources().

Caution: You should never modify the R.java file by hand—it is generated by the aapt tool when your project is compiled. Any changes are overridden next time you compile..

Accessing Resources in Code - Example

```
// 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 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.
- For example, if you add a Button to your layout, you should use a string resource for the button text:

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

Accessing Resources from XML

```
<?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" />
```

Accessing System Resources from XML

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

```
<?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" />
```

Accessing Array Resources

```
Animation tranOut;
tranOut = AnimationUtils.loadAnimation(this,
    R.anim.spin_shrink_fade);
String[] stringArray;
stringArray = myResources.getStringArray(R.array.string_array);
int[] intArray = myResources.getIntArray(R.array.integer_array);
```

Referring to Styles in the Current Theme

- Android provides a shortcut to let you use styles from the currently applied theme.
- use ?android: rather than @ as a prefix to the resource you want to use.

```
<EditText
android:id=”@+id/myEditText”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”@string/stop_message”
android:textColor=”?android:textColor”
/>
```

Providing Alternative Resources

- To specify configuration-specific alternatives for a set of 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.
  - *<qualifier>* is a name that specifies an individual configuration for which these resources are to be used.
- You can append more than one *<qualifier>*. Separate each one with a dash.
- Save the respective alternative resources in this new directory.
- The resource files must be named exactly the same as the default resource files.

Alternate Resources - Example

res/

drawable/

icon.png

background.png

drawable-hdpi/

icon.png

background.png

Note: The images in each of these drawable directories are sized for a specific screen density, but the filenames are exactly the same.
This way, the resource ID that you use to reference the icon.png or background.png image is always the same, but Android selects the version of each resource that best matches the current device

Qualifiers

Qualifier	Values
MCC and MNC	Examples: mcc310 mcc310-mnc004 mcc208-mnc00 etc.
Language and region	Examples: en fr en-rUS fr-rFR fr-rCA etc.
Screen size	small normal large
Screen aspect	long notlong

Qualifiers

Qualifier	Values
Dock mode	car desk
Night mode	night notnight
Screen pixel density (dpi)	ldpi mdpi hdpi nodpi
Touchscreen type	notouch stylus finger
Keyboard availability	keysexposed keyssoft
Primary text input method	nokeys qwerty 12key

Qualifiers

Qualifier	Values
Navigation key availability	navexposed navhidden
Primary non-touch navigation method	nonav dpad trackball wheel
System Version (API Level)	Examples: v3 v4 v7 etc.

Qualifier name Rules

- You can specify multiple qualifiers for a single set of resources, separated by dashes.
  For example, drawable-en-rUS-land applies to US-English devices in landscape orientation.
- The qualifiers must be in the order listed in table earlier.
  For example:
  - Wrong: drawable-hdpi-port/
  - Correct: drawable-port-hdpi/
- Alternative resource directories cannot be nested.
- Values are case-insensitive. The resource compiler converts directory names to lower case before processing to avoid problems on case-insensitive file systems.
- Only one value for each qualifier type is supported.

- Android 1.5 (and lower) does not support the following configuration qualifers:
  - Density ldpi, mdpi, ldpi, and nodpi
  - Screen size small, normal, and large
  - Screen aspect long and notlong

How Android Finds the Best-matching Resource

- Android selects which alternative resource to use at runtime, depending on the current device configuration.
- Android uses the following logic to select best alternative resources:
  1. Eliminate resource files that contradict the device configuration.
  2. Pick the (next) highest-precedence qualifier in the list.
  3. Do any of the resource directories include this qualifier?
     1. If No, return to step 2 and look at the next qualifier.
     2. If Yes, continue to step 4.
  4. Eliminate resource directories that do not include this qualifier.
  5. Go back and repeat steps 2, 3, and 4 until only one directory remains.

Handling Runtime Changes

- When device configurations changes during runtime, Android restarts the running Activity (onDestroy() is called, followed by onCreate()).
- When configuration changes, restarting your application and restoring significant amounts of data can be costly and create a poor user experience.
  In such a situation, you have two options:
  - Retain an object during a configuration change
    Allow your Activity to restart when a configuration changes, but carry a stateful Object to the new instance of your Activity.

Handling Runtime Changes

- Handle the configuration change yourself

  Prevent the system from restarting your Activity during certain configuration changes and receive a callback when the configurations do change, so that you can manually update your Activity as necessary.

Retaining an Object During a Configuration Change

- In a situation where restarting the activity is costly, you can alleviate the burden of reinitializing your Activity by retaining a stateful Object when your Activity is restarted due to a configuration change.
- To retain an Object during a runtime configuration change:
  - Override the onRetainNonConfigurationInstance() method to return the Object you would like to retain.
  - When your Activity is created again, call getLastNonConfigurationInstance() to recover your Object.
- Android calls onRetainNonConfigurationInstance() between onStop() and onDestroy() when it shuts down your Activity due to a configuration change.

Saving an Object when activity restarts

```
@Override
    public Object onRetainNonConfigurationInstance() {
       final MyDataObject data = collectMyLoadedData();
       return data;
    }
```

Then retrieve the data when your Activity starts again:

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

       final MyDataObject data = (MyDataObject) getLastNonConfigurationInstance();
       if (data == null) {
          data = loadMyData();
       }
       ...
    }
```

Handling the Configuration Change Yourself

- You can declare that your Activity handles the configuration change itself, which prevents the system from restarting your Activity.
- To declare that your Activity handles a configuration change, edit the appropriate <activity> element in your manifest file to include the android:configChanges attribute with a string value that represents the configuration that you want to handle.

```
<activity android:name=".MyActivity"
          android:configChanges="orientation|keyboardHidden"
          android:label="@string/app_name">
```

Handling the Configuration Change Yourself

- When one of the mentioned configurations change, MyActivity is not restarted. Instead, the Activity receives a call to onConfigurationChanged().
  - This method is passed a Configuration object that specifies the new device configuration.
- At the time this method is called, your Activity's Resourcesobject is updated to return resources based on the new configuration.

Handling the Configuration Change Yourself - Example

```
@Override
    public void onConfigurationChanged(Configuration newConfig) {
        super.onConfigurationChanged(newConfig);

        // Checks the orientation of the screen
        if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
            Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
        } else if (newConfig.orientation ==
    Configuration.ORIENTATION_PORTRAIT){
            Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
        }
```

Continued..

```
// Checks whether a hardware keyboard is available
    if (newConfig.hardKeyboardHidden ==
Configuration.HARDKEYBOARDHIDDEN_NO) {
        Toast.makeText(this, "keyboard visible",
Toast.LENGTH_SHORT).show();
    } else if (newConfig.hardKeyboardHidden ==
Configuration.HARDKEYBOARDHIDDEN_YES) {
        Toast.makeText(this, "keyboard hidden",
Toast.LENGTH_SHORT).show();
    }
}
```

More Resource Types

- Bool
  - XML resource that carries a boolean value.
- Color
  - XML resource that carries a color value (a hexadecimal color).
- Dimension
  - XML resource that carries a dimension value (with a unit of measure).

More Resource Types

- ID
  - XML resource that provides a unique identifier for application resources and components.
- Integer
  - XML resource that carries an integer value.
- Integer Array
  - XML resource that provides an array of integers.
- Typed ArrayXML resource that provides a TypedArray (which you can use for an array of drawables).

Bool

- XML file saved at res/values-small/bools.xml:<?xml

version="1.0" encoding="utf-8"?>
<resources>
    <bool name="screen_small">true</bool>
    <bool name="adjust_view_bounds">true</bool>
</resources>This application code retrieves the boolean:

```
Resources res = getResources();
boolean screenIsSmall = res.getBoolean(R.bool.screen_small);
```

This layout XML uses the boolean for an attribute:

```
<ImageView
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android:src="@drawable/logo"
    android:adjustViewBounds="@bool/adjust_view_bounds" />
```

Integer Array

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <integer-array name="bits">
        <item>4</item>
        <item>8</item>
        <item>16</item>
        <item>32</item>
    </integer-array>
</resources>
```

This application code retrieves the integer array:

```
Resources res = getResources();
int[] bits = res.getIntArray(R.array.bits);
```

Typed Array

```
<?xml version="1.0" encoding="utf-8"?>
   <resources>
      <array name="icons">
         <item>@drawable/home</item>
         <item>@drawable/settings</item>
         <item>@drawable/logout</item>
      </array>
      <array name="colors">
         <item>#FFFF0000</item>
         <item>#FF00FF00</item>
         <item>#FF0000FF</item>
      </array>
   </
```

Typed Array

- resources>This application code retrieves each array and then obtains the first entry in each array:

```
Resources res = getResources();
    TypedArray icons = res.obtainTypedArray(R.array.icons);
    Drawable drawable = icons.getDrawable(0);

    TypedArray colors = res.obtainTypedArray(R.array.icons);
    int color = colors.getColor(0,0);
```