Android: Resources, Animation and Intents

Tommy MacWilliam

Harvard University

February 22, 2011

Announcements

- Lecture videos available at: https://www.cs76.net/Lectures
- New section schedule: https://www.cs76.net/Sections
  - section every week here right after lecture
  - office hours weeks projects are due Tuesday/Wednesday nights
  - walkthroughs on Thursdays after project released
  - seminars on interesting mobile dev topics on non-walkthrough weeks
  - online labs

Today

- Resources
- Styles
- Animation
- Intent Filters

Section Feedback

- http://tommymacwilliam.com/e76/feedback
  - let me know how I'm doing!
- I don't like long surveys either, so give me feedback via an anonymous (I promise) 140-character tweet!

Resources

- remember, resources are anything in our project that isn't code
- all resources will be placed in a special directory called `res`
  - this directory contains subfolders to organize your resources by type

Resources

- `anim`: animations (which we'll take a look at later)
- `color`: colors used for different states of UI elements
- `drawable`: PNG, JPG, GIF, etc.
- `layout`: layouts for activities (which we saw last week)
- `menu`: app menus (i.e. Options, Context, etc.)
- `raw`: any file needed in its raw for (i.e. plaintext)
- `xml`: configuration XML files

Accessing Resources

- resources can be accessed from XML and Java
  - XML: `@<type>/<name>`
  - Java: `R.<type>.<name>`
- `<type>`: `anim`, `drawable`, etc.
- `<name>`: filename (without the extension)

Resources

- `values`: predefined strings, integers, colors, etc.
  - filenames are arbitrary, can contain arbitrary XML elements
  - each element must be a child of `<resources>`
  - from Java: `R.<element>.<name>`
  - from XML: `@<element>/<name>`

Alternative Resources

- unlike the iPhone, there are many different types of Android devices
  - which all have different hardware specs, screen resolutions, etc.
- all of the folders in the `res` directory can be device-specific
  - create a new folder called `<name>-<qualifier 1>-<qualifier 2>-...`
  - can have any number of qualifiers, which match devices that match all qualifiers

Alternative Resources

- resource qualifiers
  - language: `en`, `en-rUS`, `es`, etc. (any valid ISO 639-1 code)
    - http://www.loc.gov/standards/iso639-2/php/code_list.php
  - screen size: `small`, `normal`, `large`, `xlarge`
  - screen aspect ratio: `long`, `notlong`
  - screen orientation: `port`, `land`

Alternative Resources

- resource qualifiers
  - pixel density: `ldpi, mdpi, hdpi, xhdpi, nodpi`
  - text input: `nokeys, qwerty, 12key`
  - navigation: `nonav, dpad, trackball, wheel`

Picking the Best Resource

- must be specified in the order they were presented here
- 4-step process
  - eliminate any contradictions (device is `en`, app has config for `es`)
  - iterate through qualifiers (all possible qualifiers in the predefined order)
  - check if any folder matches the qualifier
  - if match, eliminate directories that do not include the qualifier (until only one directory is left)

Alternative Resources

- example time!

Styles

- define a custom look and feel for UI elements
  - the Android equivalent of CSS
  - makes View XML much less verbose
  - can apply the same style to multiple Views
- stored in XML files located in `res/values/`

Styles

- `<style>` defines multiple `<item>`s
- `<item>`'s `name` attribute is equivalent to the View attribute
  - value equivalent to the value of the View attribute
- `<Something android:layout_width="fill_parent" />`
  - becomes:

```
<item name="android:layout_width">
    fill_parent
</item>
```

Styles

- style can inherit from platform styles and themes via the `parent` attribute
- a style can also extend an already defined style
  - `name="MyStyle.Something"` will automatically inherit from the style with `name="MyStyle"`

Styles

- example time!

Tween Animations

- despite its name, unrelated to Justin Bieber
- equivalent to CSS3 transitions, can be used to animate any View object
  - admittedly a bit more verbose than CSS3, though
- saved in `res/anim`
- `<set>` defines properties to be animated concurrently

Tween Animations

- transitions
  - `<alpha>`: fade a View's opacity in/out
  - `<scale>`: resize a View
  - `<translate>`: vertical/horizontal motion
  - `<rotate>`: rotate a View
- all transitions must be children of a `<set>`, but a `<set>` can contain `<set>`s

Tween Animations

- attributes in the form `android:from<Something>` and `android:to<Something>`
  - `<alpha>: fromAlpha, toAlpha`
  - `<scale>: from{X,Y}Scale, to{X,Y}Scale`
  - `<translate>: from{X,Y}, to{X,Y}, from{X,Y}Delta, to{X,Y}Delta`
  - `<rotate>: fromDegrees, toDegrees`
  - `<all>: duration, startOffset`

Applying Animations

- first, we load a new `Animation` via the `AnimationUtils.loadAnimation`
  - recall we can access the animation via `R.anim.<filename>`
- call `startAnimation` on the View we want to animate
- `AnimationListener` provides callbacks to detect when an animation is complete

Tween Animations

- example time!

Interpolators

- just like in CSS3, the interpolator determines how the animation is applied
  - built-in interpolators given by `@android:anim/<interpolator>`
  - `accelerate_decelerate_interpolator`, `bounce_interpolator`, `linear_interpolator`, `overshoot_interpolator`
- custom interpolators can also be defined
  - `<bounceInterpolator>`, etc. are XML elements, attributes changeable to customize animation

Interpolators

- example time!

Frame Animation

- used to animate `drawable` elements
- animation consists of multiple `<item>`s, each a child of an `<animation-list>`
  - each `<item>` must have an `android:drawable` and `android:duration`
  - remember, we can access `drawable` resources via `@drawable/<name>`

Intents

- remember, Activities, Broadcast Receivers, and Services are triggered via Intents
  - last week:

```
Intent i = new Intent(Context
packageContext, Class<?> cls);
startActivity(i);
```

- Intent objects simply contain a description of what action is to be performed
  - has an action, data, and a category
- we can use Intents to trigger (private) actions within our own app and (public) actions within other apps

Intents

- `Intent i = new Intent(<action>, <URI>);`
- action: the action to be performed
  - `ACTION_MAIN`: show the initial Activity
  - `ACTION_DIAL`: present a dialer for a phone call
  - `ACTION_EDIT`: display editable data to the user
  - `ACTION_VIEW`: view content (web page, etc.)
  - `ACTION_WEB_SEARCH`: search the web for data
- manage actions with `setAction()` and `getAction()`

Intents

- `<URI>` contains data for the action (phone number to call, etc.) formatted as a URI
  - `scheme://host/path/segments`
  - `getScheme(), getHost(), getPathSegments()`
- category: additional information about the component that should respond
  - `CATEGORY_BROWSABLE`: component can be invoked by the browser to display content
  - `CATEGORY_LAUNCHER`: component can be shown as the initial activity from the launcher
- manage categories with `addCategory()` and `removeCategory()`

Bundles

- we can pass additional data to Activities with bundles
- sending data: `intent.putExtra("key", "value");`
- retrieving passed data:

```
Bundle bundle = getIntent().getExtras();
bundle.getString("key");
```

Intent Filters

- we can also allow other apps to open Activities within our app via Intent Filters
- we register these in `AndroidManifest.xml`
  - generally not created using Java
  - also where we tell Android what Activities exist in our app

Intent Filters

- `<intent-filter>` must be a child of an `<activity>` and can contain `<action>`, `<data>`, and `<category>` elements
  - `<action>`: action to be performed
  - `<data>`: format through which data is passed to the activity
    - accessed via `getIntent().getData()`
  - `<category>`: component properties

Intent Filters

- we can also use Intent Filters to open our app's Activities from within our app
  - the other syntax is kinda ugly
- if we give our `<action>` a unique `android:name` attribute, we can just pass that to the Intent constructor

Intent Filters

- example time!