Session #5: Loaders and Responsive Layouts

March 12th, 2015

Overview

- Survey
- Final Project & Demo Details
- Loaders
- Responsive Layout
- Final Project Ideas?

Survey

Take the survey!

http://goo.gl/aYmmFJ

Why Loaders?

1) Simple data loading

Convenient asynchronous data loading for Activities and Fragments

2) **Management not needed**

Exists beyond the lifecycle of an Activity or Fragment using LoaderManager

Loader Implementation

1) Extend LoaderManager.LoaderCallbacks
2) Define your Loader ID
3) Implement
    a) onCreateLoader
    b) onLoadFinished
    c) onLoaderReset

CursorAdapter

Bind data from a database Cursor to an AdapterView class (ListView, GridView, etc.)

newView(...) - inflate your view from XML

bindView(...) - pull data out of your cursor and populate your view

ViewHolder Pattern

ViewHolder holds references to Views in an object. The object is attached to the view it represents.

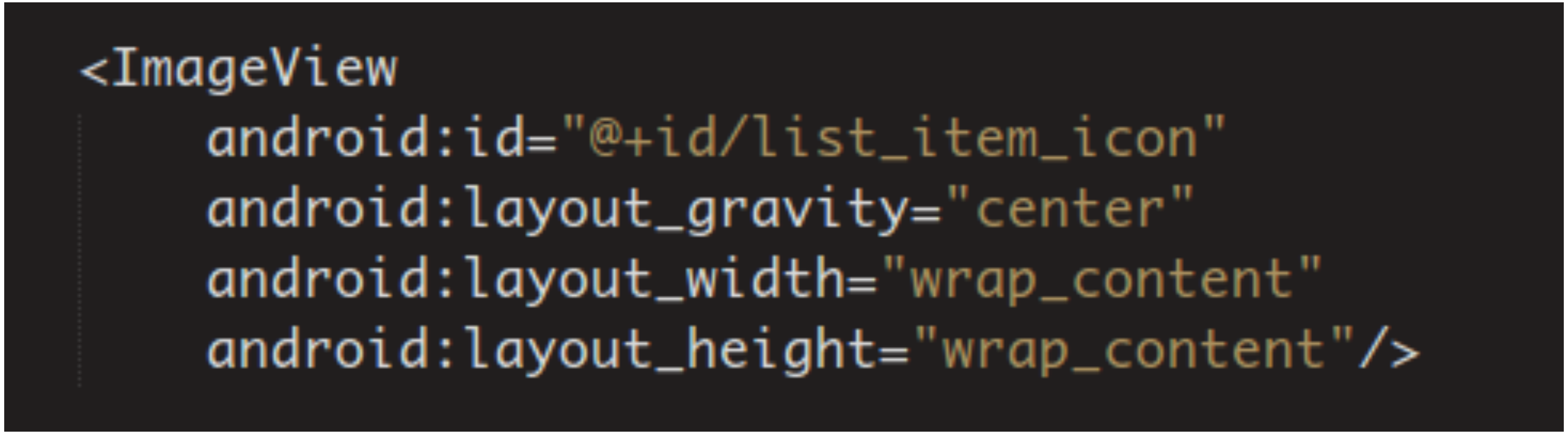
Rich & Responsive Layouts

Views & ViewGroups

Views are the basic building blocks of Android user interfaces (ie. EditText, Spinner, ImageView, etc.)

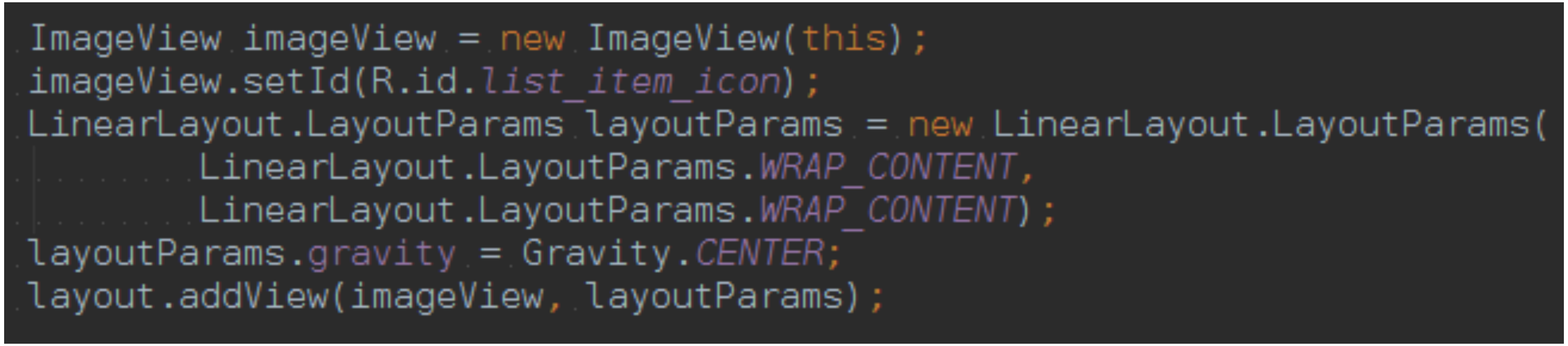
ViewGroups are <u>views</u> that can contain child views (ie. RelativeLayout, LinearLayout, etc)

View Definition (XML)



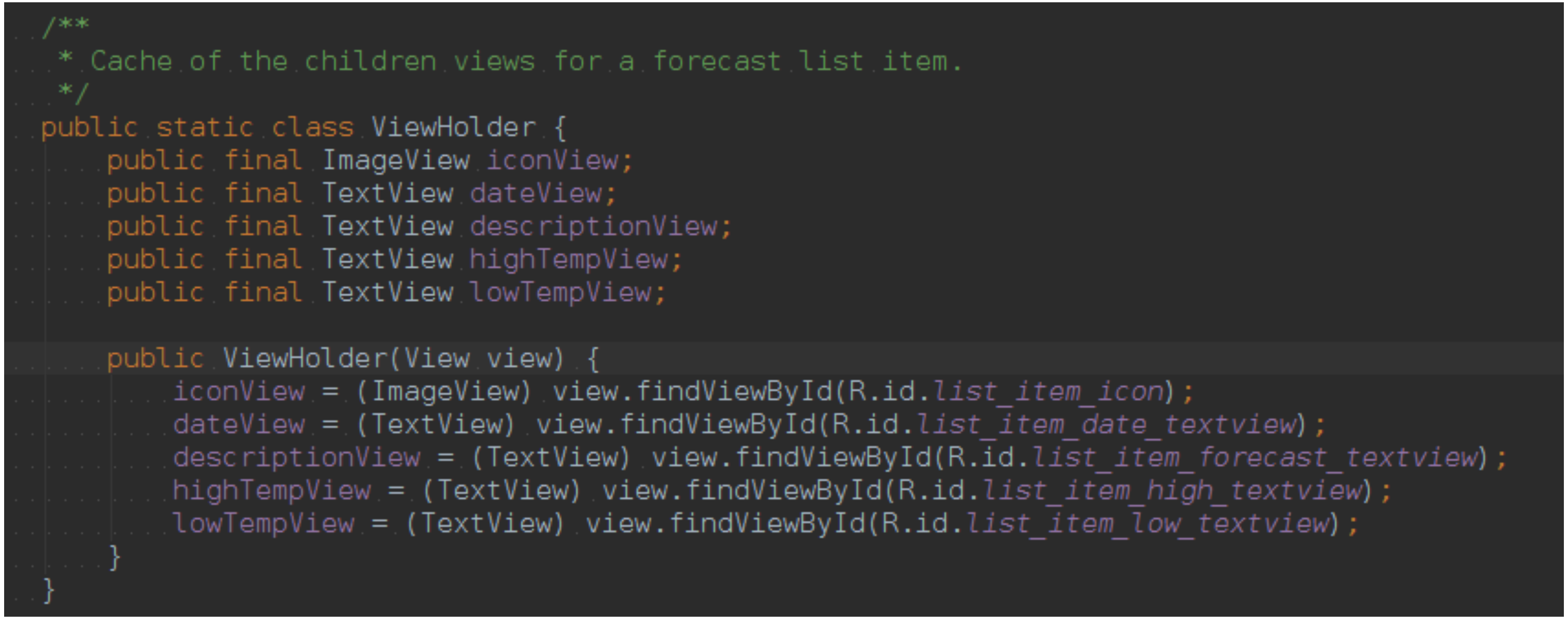
```
<ImageView
    android:id="@+id/list_item_icon"
    android:layout_gravity="center"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

View Definition (Java)



```
ImageView imageView = new ImageView(this);
imageView.setId(R.id.list_item_icon);
LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams(
        LinearLayout.LayoutParams.WRAP_CONTENT,
        LinearLayout.LayoutParams.WRAP_CONTENT);
layoutParams.gravity = Gravity.CENTER;
layout.addView(imageView, layoutParams);
```

ViewHolder Pattern Cont'd



```java
/**
 * Cache of the children views for a forecast list item.
 */
public static class ViewHolder {
    public final ImageView iconView;
    public final TextView dateView;
    public final TextView descriptionView;
    public final TextView highTempView;
    public final TextView lowTempView;

    public ViewHolder(View view) {
        iconView = (ImageView) view.findViewById(R.id.list_item_icon);
        dateView = (TextView) view.findViewById(R.id.list_item_date_textview);
        descriptionView = (TextView) view.findViewById(R.id.list_item_forecast_textview);
        highTempView = (TextView) view.findViewById(R.id.list_item_high_textview);
        lowTempView = (TextView) view.findViewById(R.id.list_item_low_textview);
    }
}
```

Responsive Layouts

Your apps have the possibility of running on hundreds of types of devices…

…countless screen sizes

…countless screen densities

Adopting responsive design patterns available in Android mitigate this opportunity

Layout Size Units

px - absolute pixels ← bad!

dip/dp - device independent pixels ← good!

Screen Densities vs. Screen Sizes

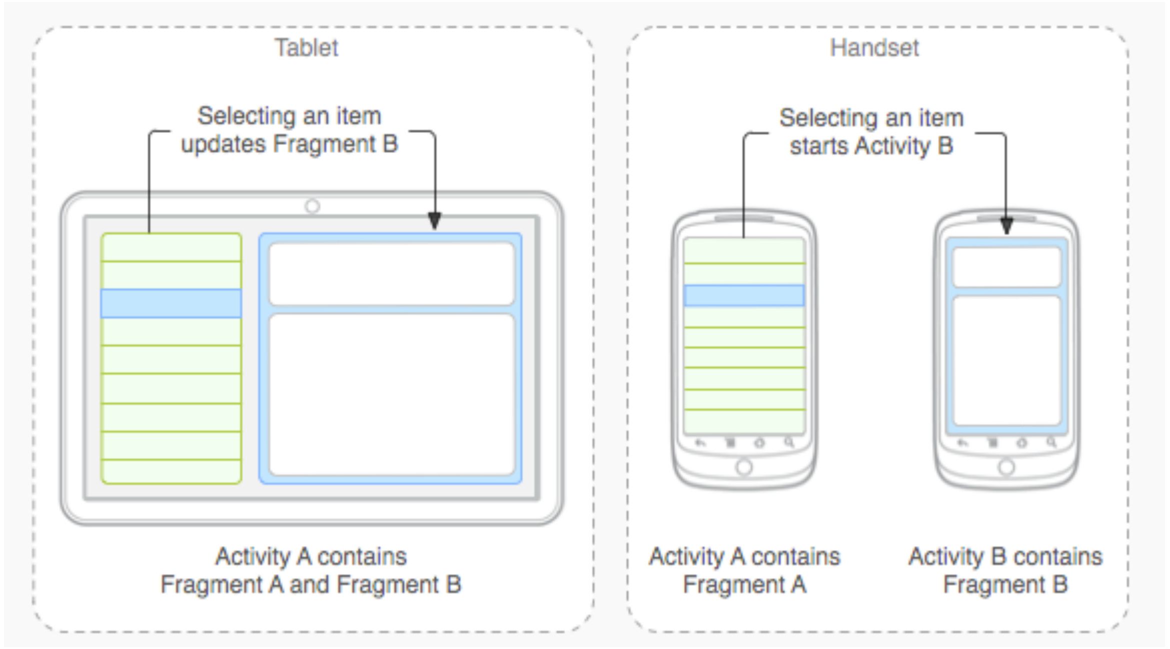
Density	Sizes
mdpi	360dp
hdpi	480dp
xhdpi	600dp
xxhdpi	720dp
xxxhdpi	

DP vs. PX

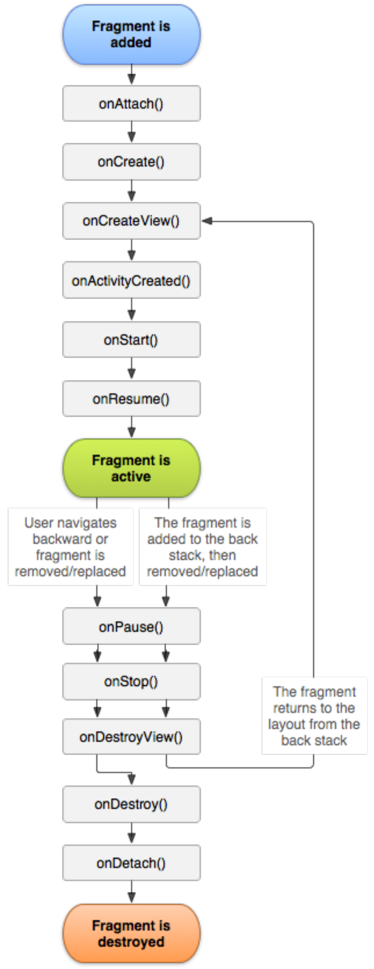
Pixels	Density	Multiplier	Actual Size
48px	mdpi	1x	48px
48px	hdpi	1.5x	72px
48px	xhdpi	2x	96px
48px	xxhdpi	3x	144px
48px	xxxhdpi	4x	192px

Fragments

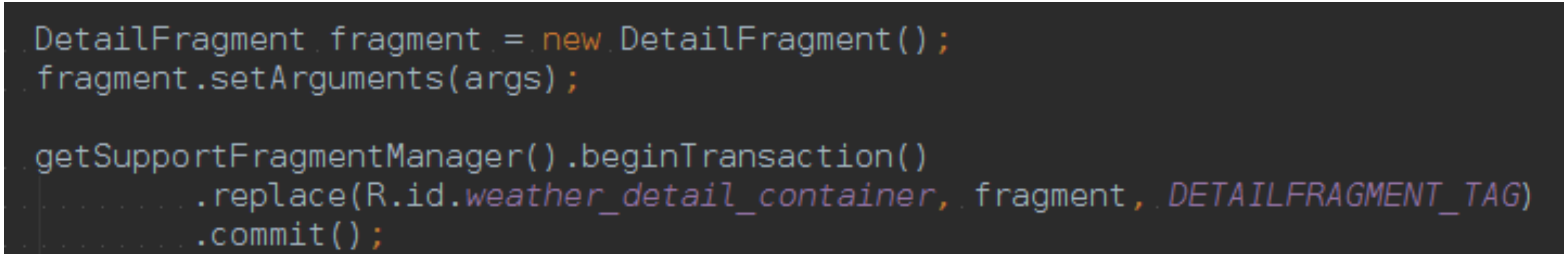
Modularize your user interface


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Fragment Lifecycle



Fragment Transactions



```
DetailFragment fragment = new DetailFragment();
fragment.setArguments(args);

getSupportFragmentManager().beginTransaction()
        .replace(R.id.weather_detail_container, fragment, DETAILFRAGMENT_TAG)
        .commit();
```

Final Project Ideas?