Advanced Android Application Development

4 – Rich and Responsive Layouts

Contents

- **Fundamental Android design principles**
- **Supporting localization and variable screen sizes**
- **Optimizing tablet UIs using Fragments**
- **Accessibility Features**
- **Custom views**

Recap on Views and ViewGroups

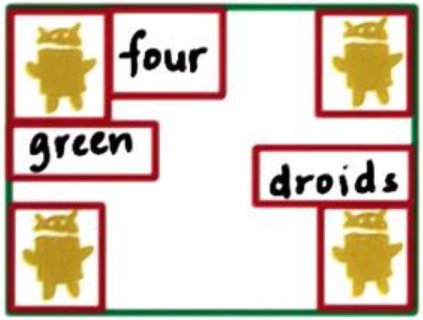
- **Android Design Building Blocks**: http://www.androiddocs.com/design/building-blocks/index.html
- **View Class:** http://developer.android.com/reference/android/view/View.html
- **View Group:** http://developer.android.com/reference/android/view/ViewGroup.html

View Groups

FrameLayout

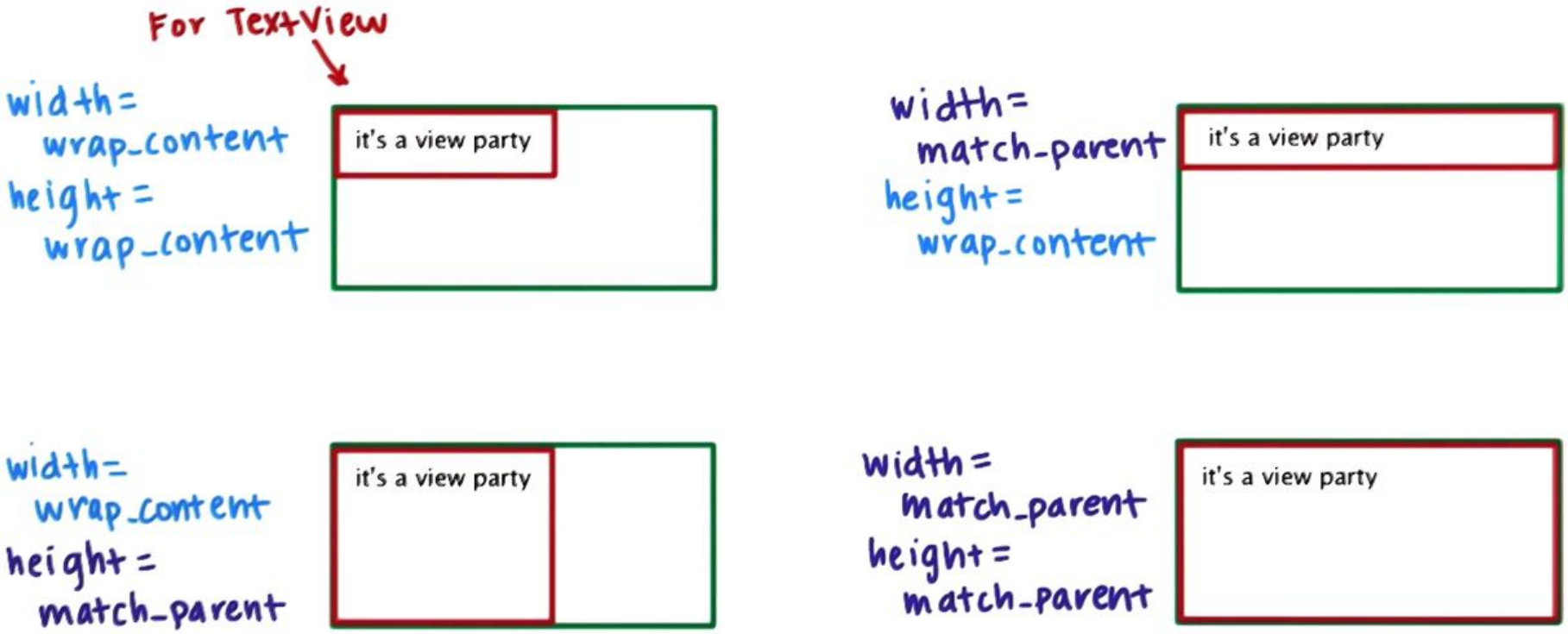
LinearLayout

RelativeLayout



GridLayout

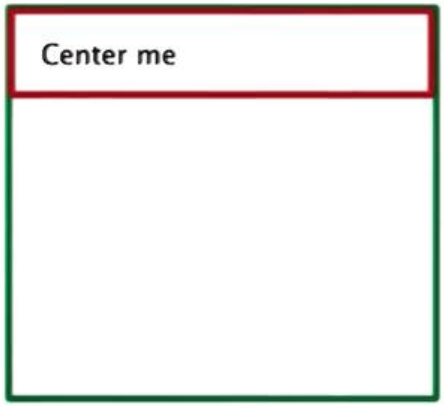
View Width/Height



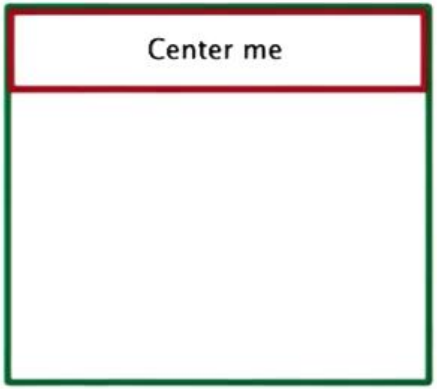
Gravity

For this TextView

No gravity set

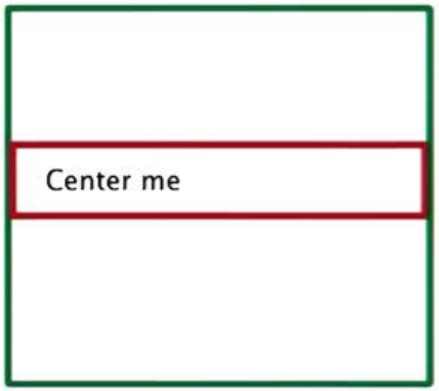


gravity = center



Center within TextView

layout_gravity = center

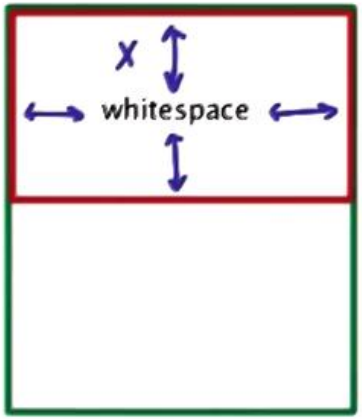


center within parent

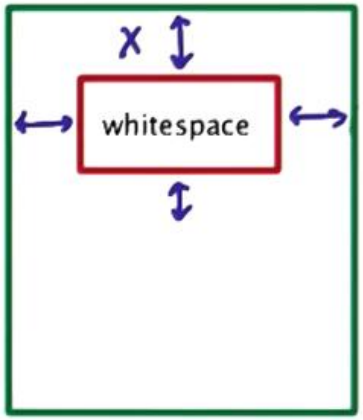
Padding vs. Margin

padding = X

For this TextView



layout_margin = X



View Visibility

For Image View

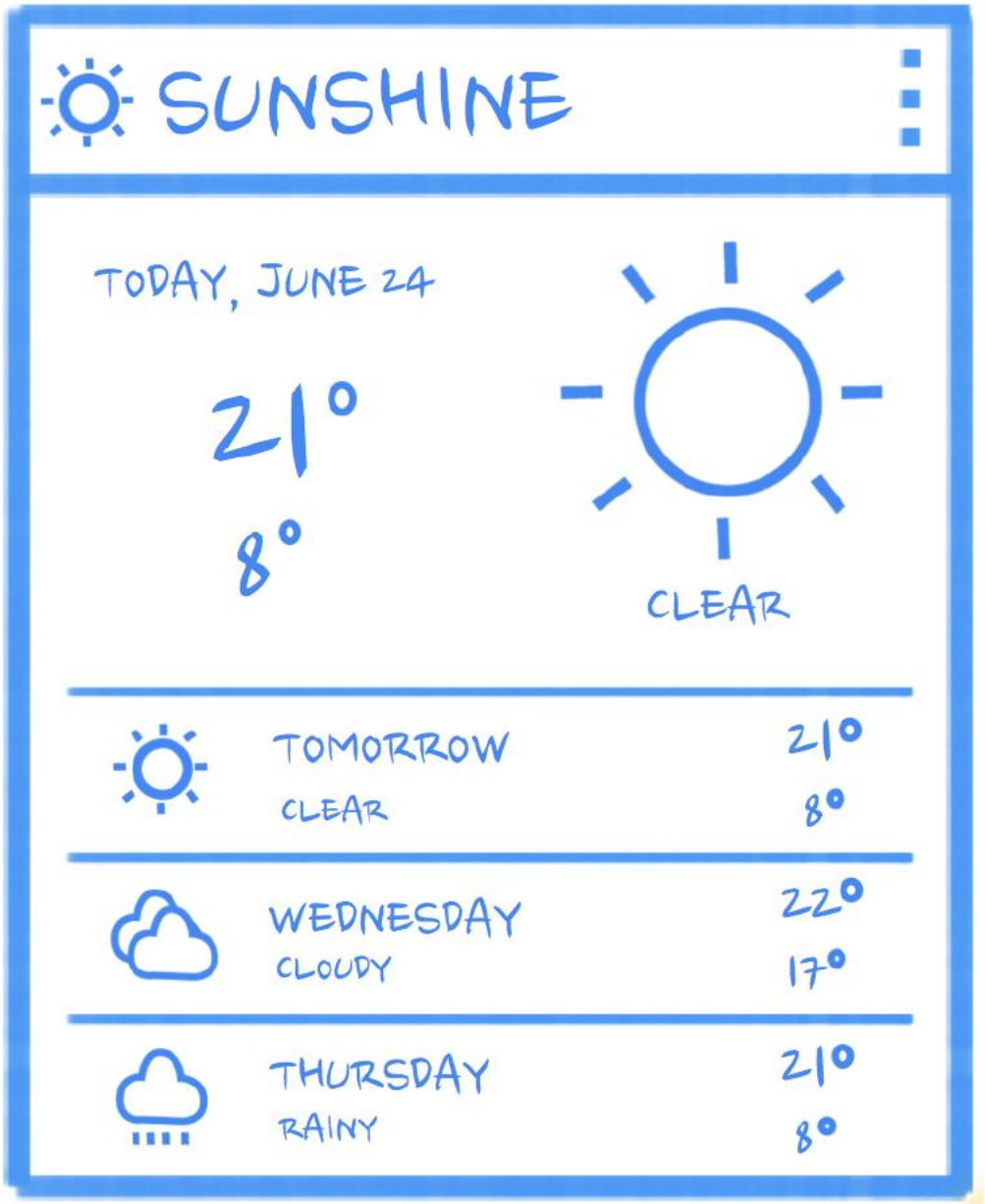
visible

invisible

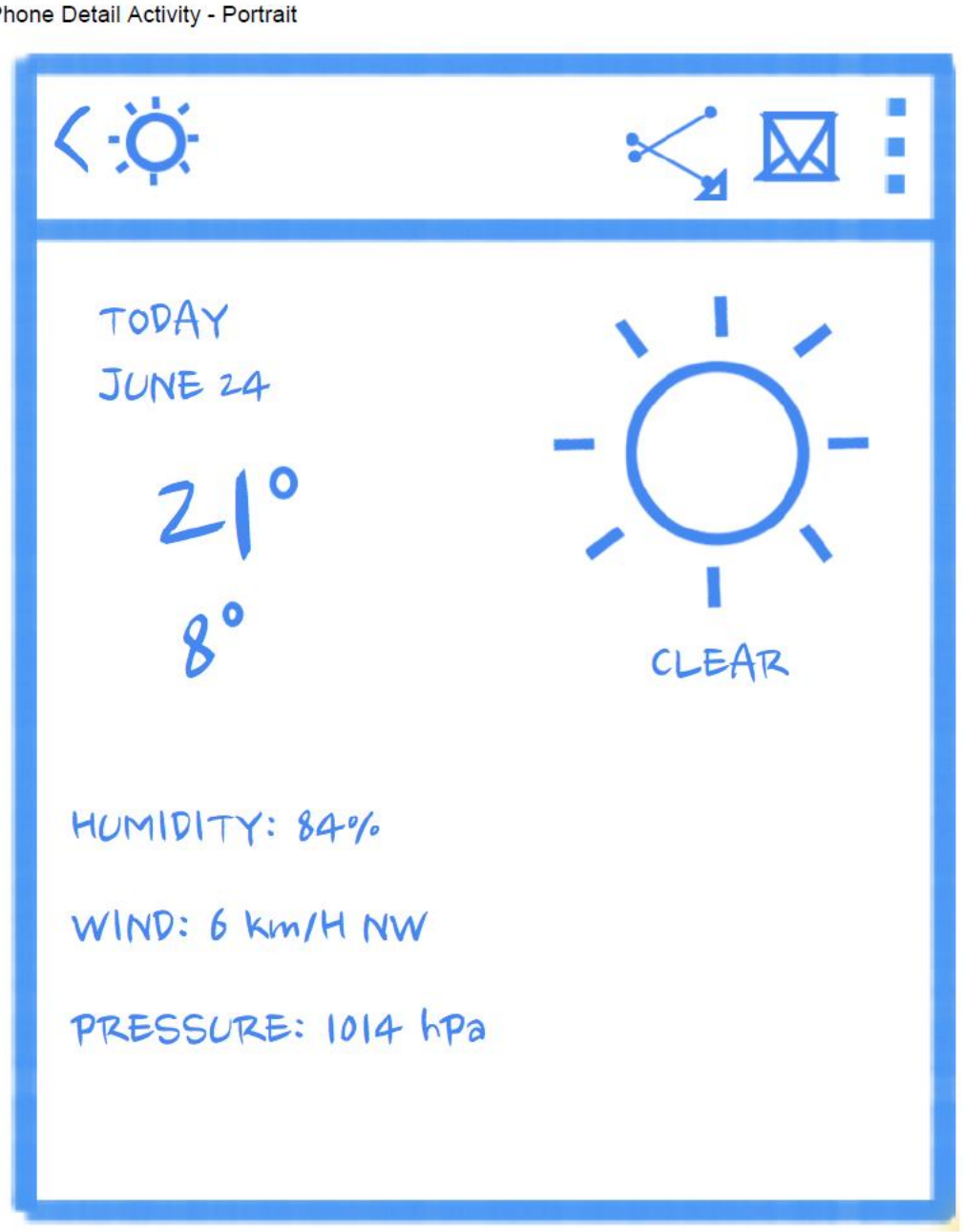
gone

Wireframe

Phone Main Activity - Portrait

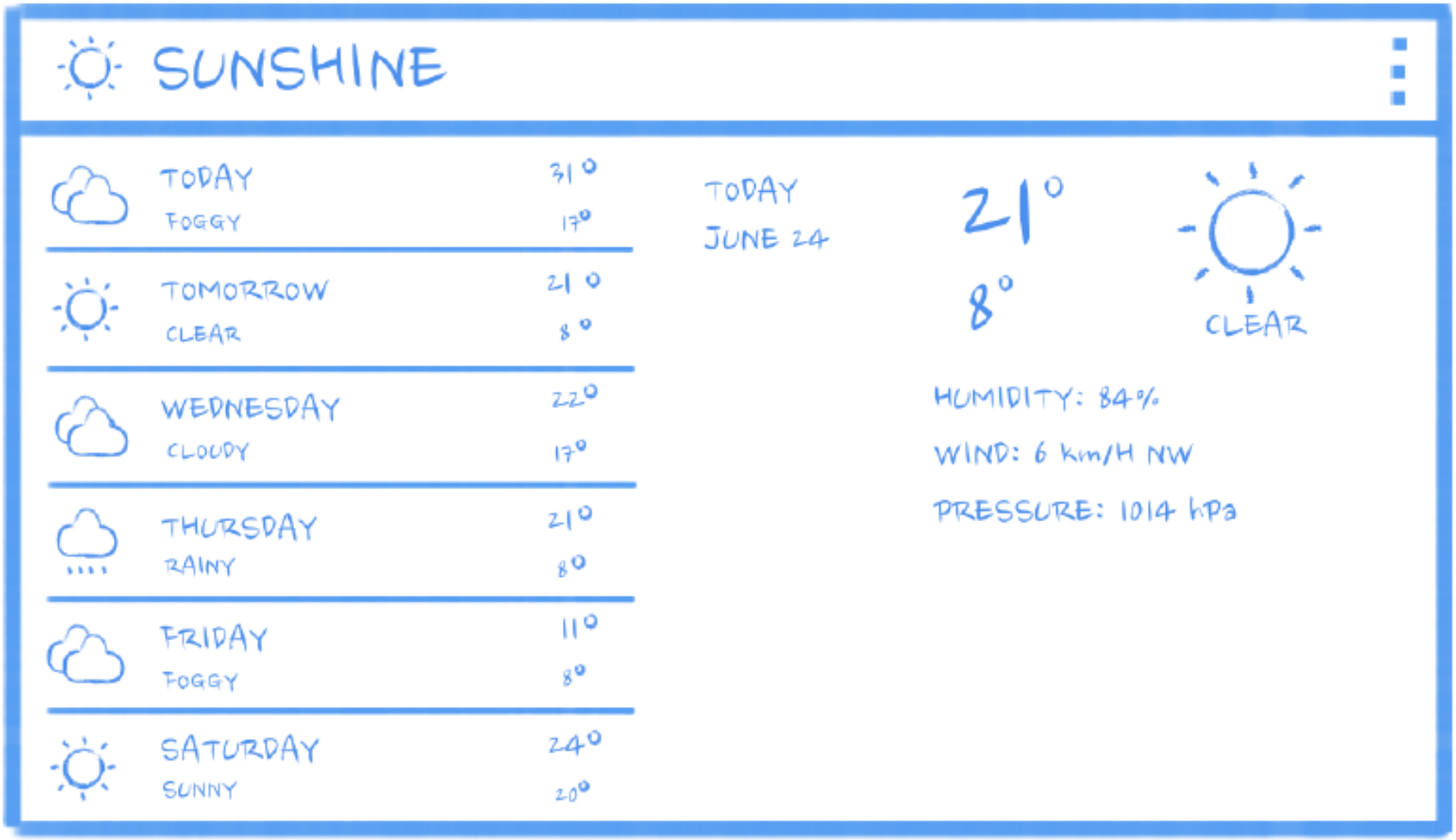


Wireframe


Phone Detail Activity - Portrait
TODAY
JUNE 24
21°
8°
CLEAR
HUMIDITY: 84%
WIND: 6 km/H NW
PRESSURE: 1014 hPa

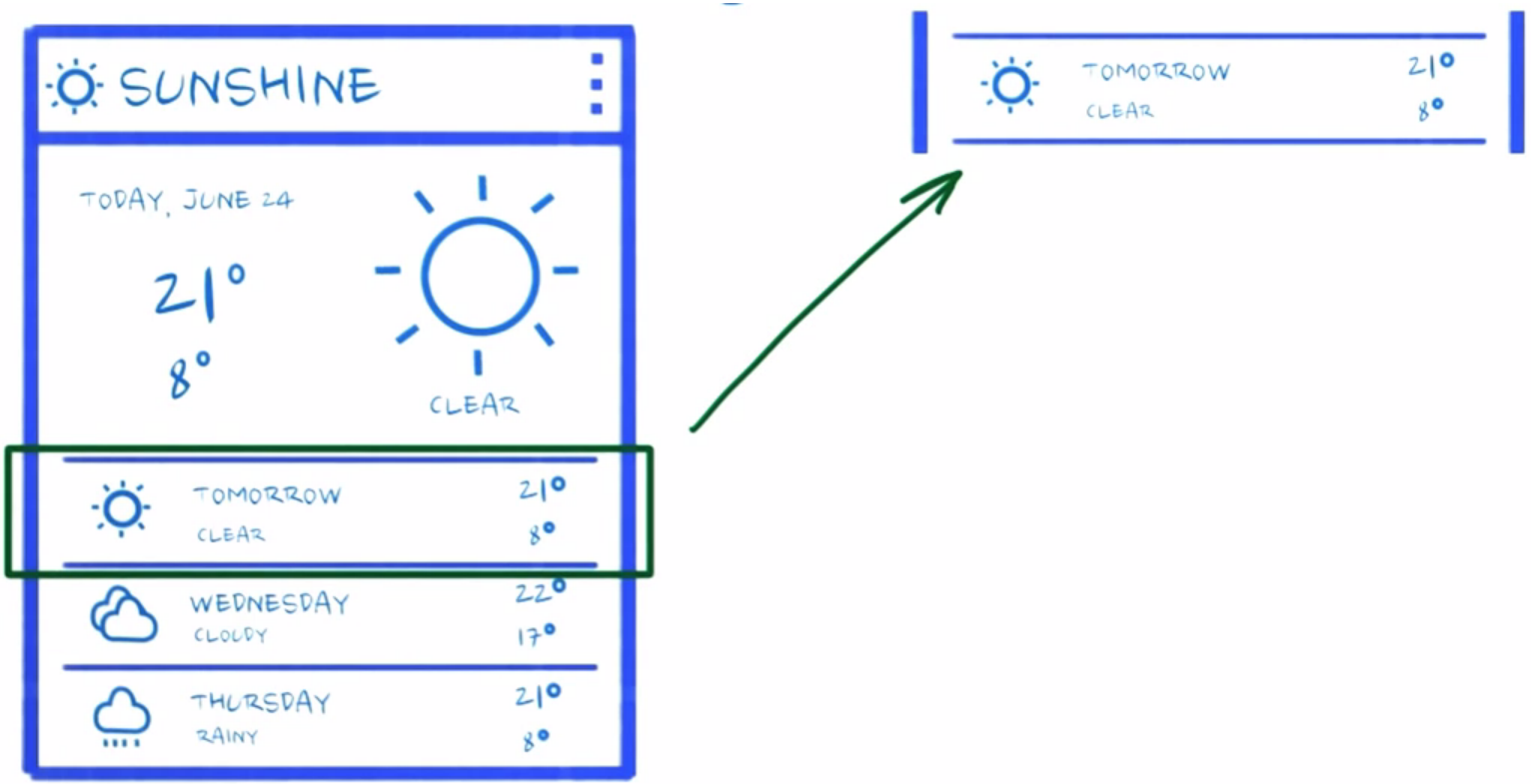
Wireframe

Tablet Main Activity - Landscape

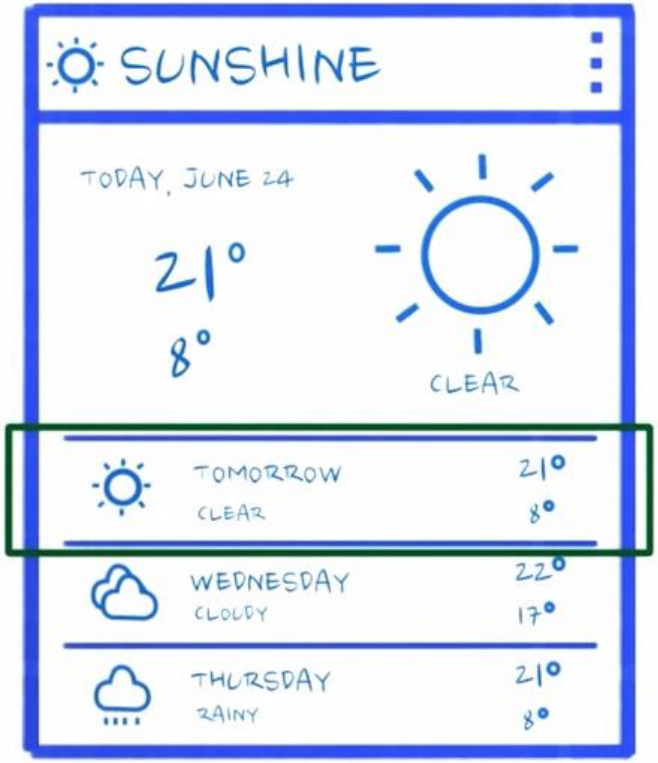


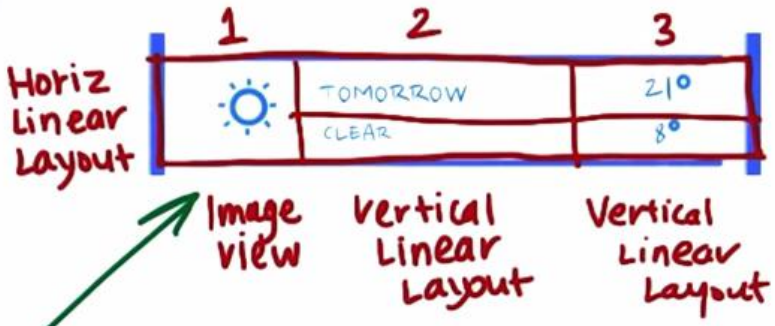
Visual Mocks, Redlines and Assets

Building List Items

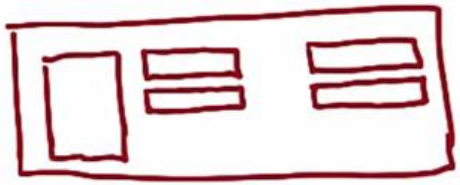


Building List Items Solution

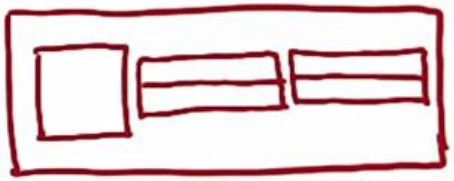




☑ Modify list_item_forecast.xml to build this layout



Relative Layout

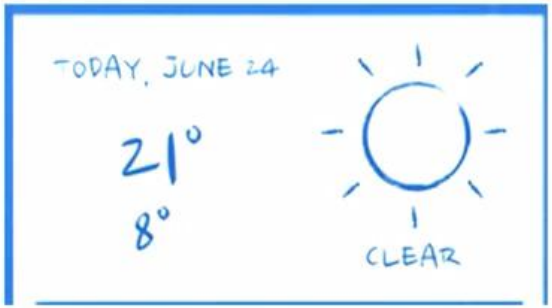


Linear Layout

Instructor Notes

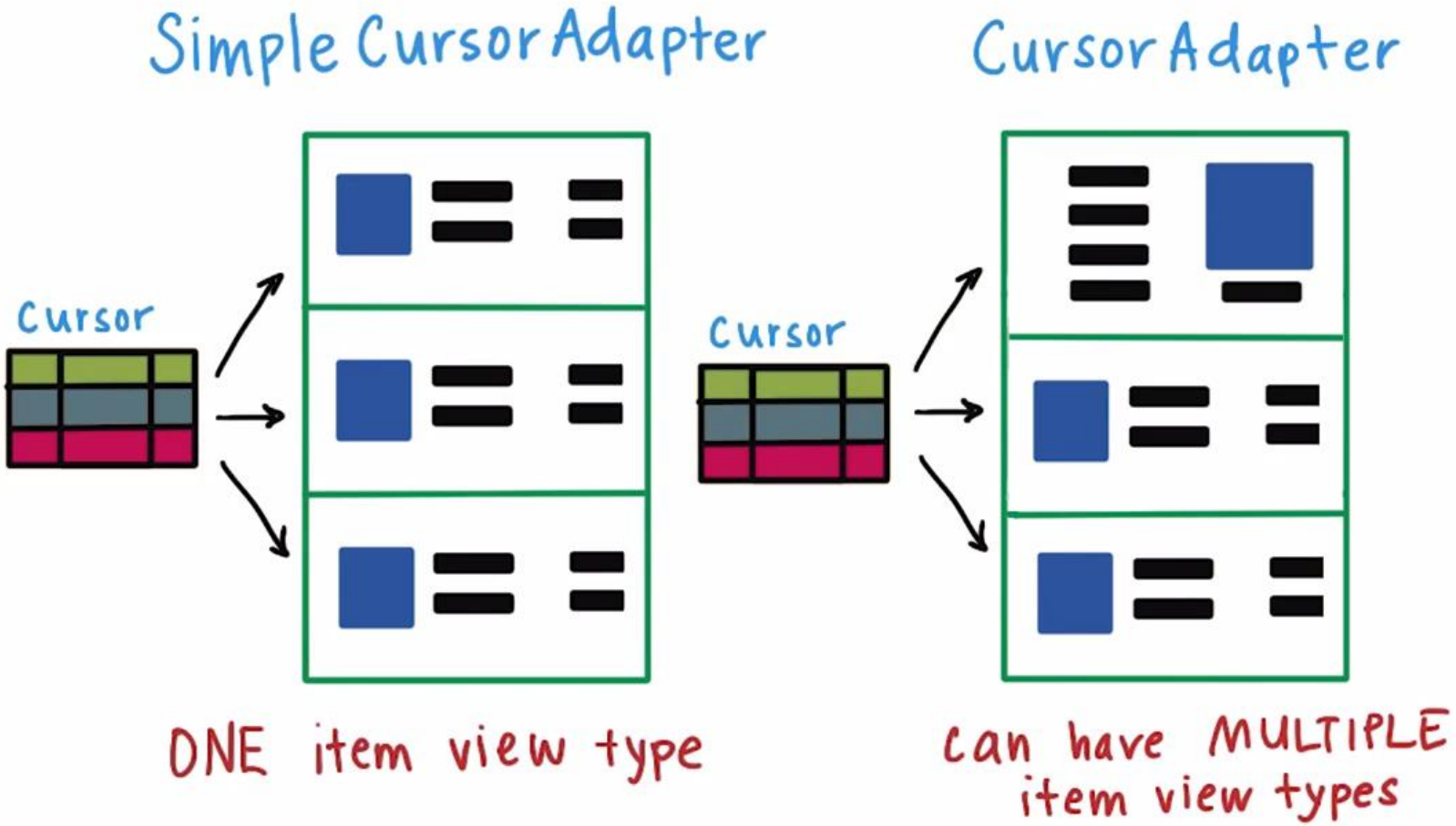
- **Assign an ID to any view that will be bound to data:**
  - `list_item_date_textview`
  - `list_item_forecast_textview`
  - `list_item_high_textview`
  - `list_item_low_textview`
  - `list_item_icon`
- **Hint: You should use Layout Weight and can read about it here.**

Building Today's List Items



- [x] Create new list_item_forecast_today.xml for Today item

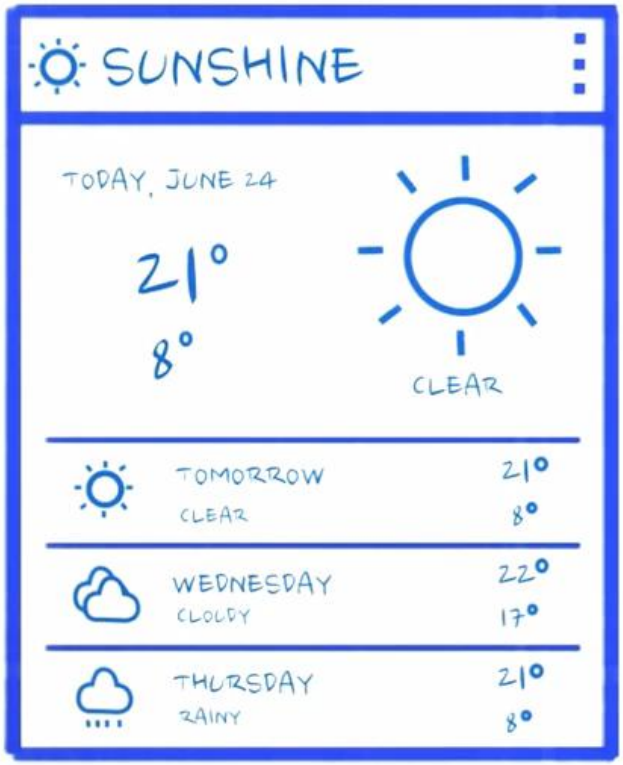
CursorAdapter



CursorAdapter

- **An abstract class**
- **Two abstract methods:**
  - **The `newView` method knows how to return a new list item layout, but doesn't contain data yet**
  - **The `bindView` method knows how to take an existing layout and update it with the data pointed to by the cursor.**

Friendly Date String



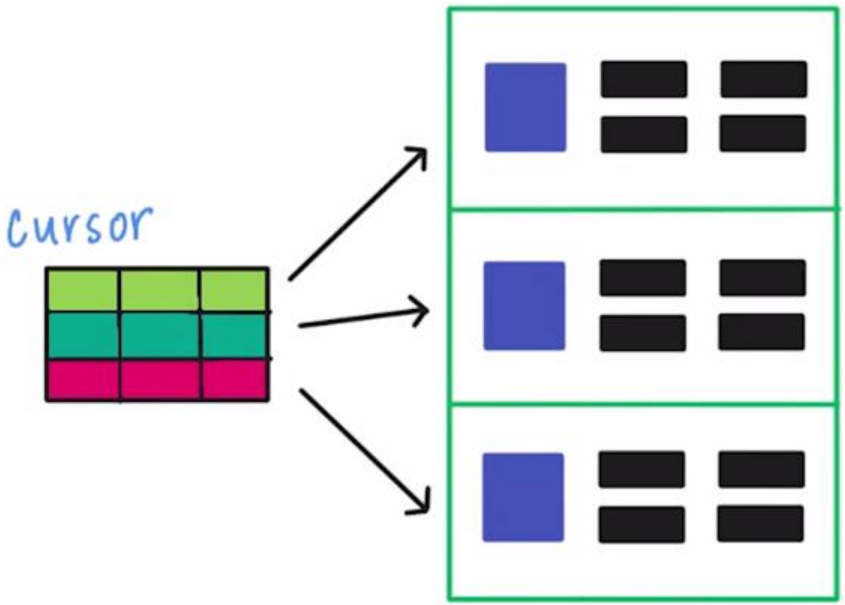
Friendly Date String

ForecastAdapter.java × | Utility.java × | ForecastFragment.java ×

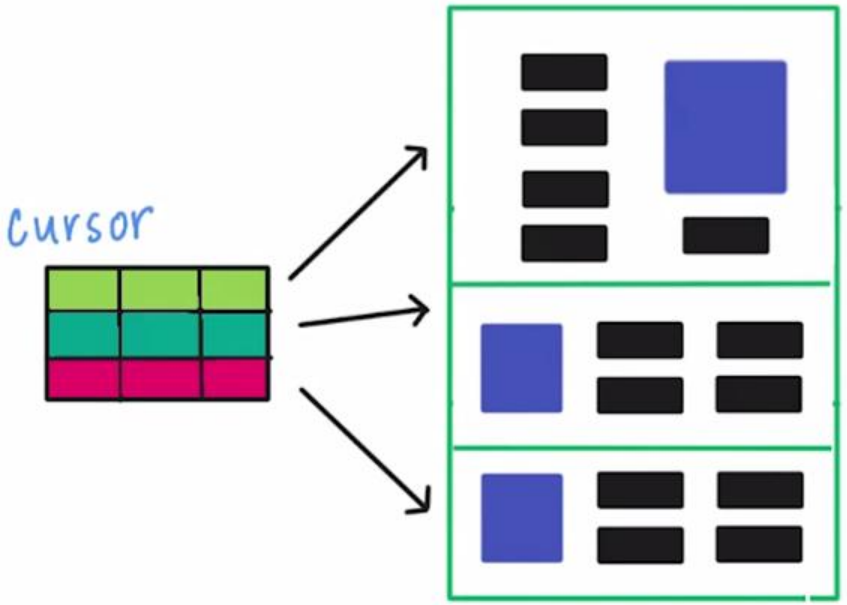
```
68      public static String getFriendlyDayString(Context context, long dateInMillis) {
69          // The day string for forecast uses the following logic:
70          // For today: "Today, June 8"
71          // For tomorrow:  "Tomorrow"
72          // For the next 5 days: "Wednesday" (just the day name)
73          // For all days after that: "Mon Jun 8"
74
75          Time time = new Time();
76          time.setToNow();
77          long currentTime = System.currentTimeMillis();
78          int julianDay = Time.getJulianDay(dateInMillis, time.gmtoff);
79          int currentJulianDay = Time.getJulianDay(currentTime, time.gmtoff);
80
81          // If the date we're building the String for is today's date, the format
82          // is "Today, June 24"
83          if (julianDay == currentJulianDay) {
84              String today = "Today";
85              int formatId = "%1$s, %2$s";
86              return String.format(context.getString(
87                      formatId,
88                      today,
89                      getFormattedMonthDay(context, dateInMillis)));
90          } else if ( julianDay < currentJulianDay + 7 ) {
91              // If the input date is less than a week in the future, just return the day name.
92              return getDayName(context, dateInMillis);
93          } else {
94              // Otherwise, use the form "Mon Jun 3"
```

Item View Types

ONE item view type



MULTIPLE item view types

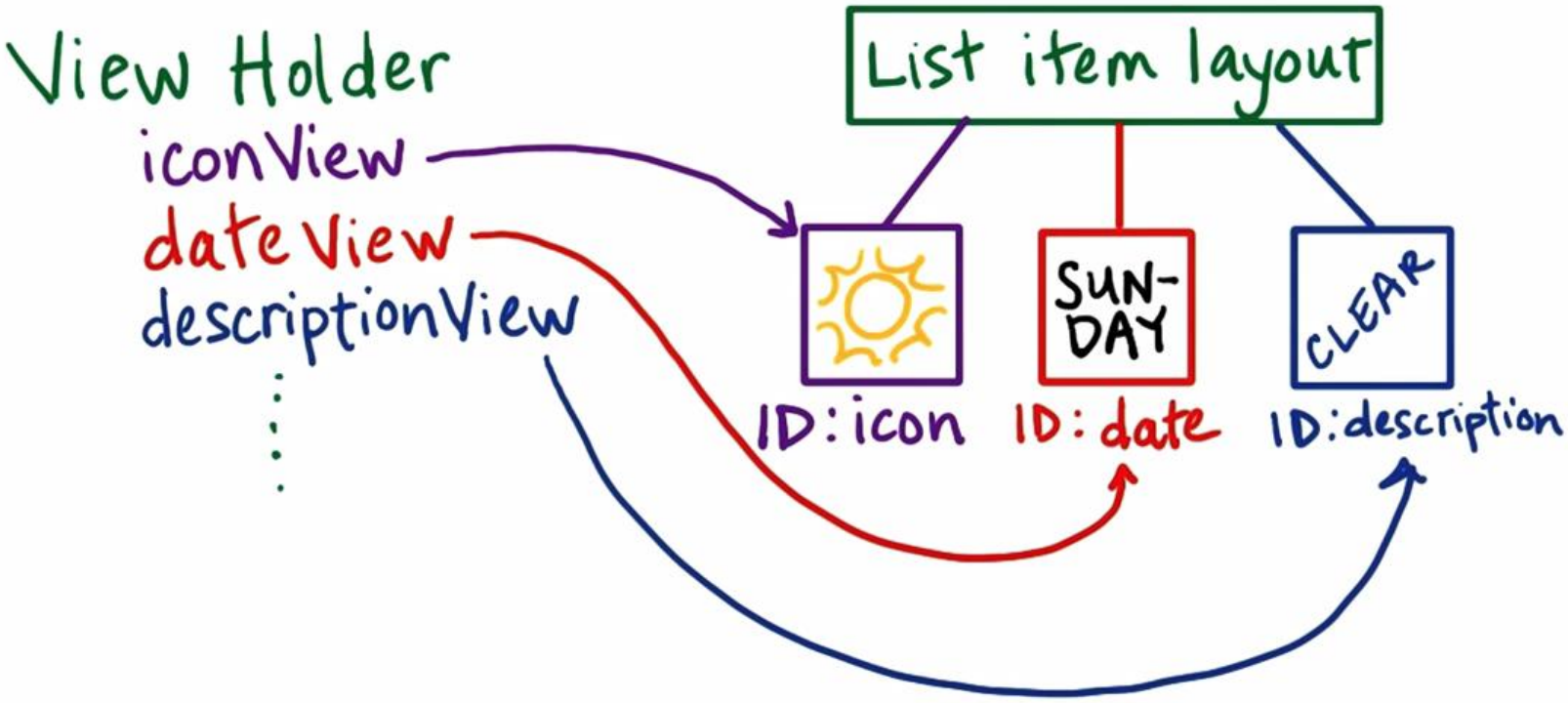


Item View Types

ForecastAdapter.java

```
21      private static final int VIEW_TYPE_COUNT = 2;
22      private static final int VIEW_TYPE_TODAY = 0;
23      private static final int VIEW_TYPE_FUTURE_DAY = 1;
24
25      @Override
26      public int getItemViewType(int position) {
27          return position == 0 ? VIEW_TYPE_TODAY : VIEW_TYPE_FUTURE_DAY;
28      }
29
30      @Override
31      public int getViewTypeCount() { return VIEW_TYPE_COUNT; }
34      /*
35          Remember that these views are reused as needed.
36       */
37
38      @Override
39      public View newView(Context context, Cursor cursor, ViewGroup parent) {
40          // Choose the layout type
41          int viewType = getItemViewType(cursor.getPosition());
42          int layoutId = -1;
43          switch (viewType) {
44              case VIEW_TYPE_TODAY: {
45                  layoutId = R.layout.list_item_forecast_today;
46                  break;
47              }
48              case VIEW_TYPE_FUTURE_DAY: {
49                  layoutId = R.layout.list_item_forecast;
```

ViewHolder Pattern



ViewHolder Pattern

- http://developer.android.com/training/improving-layouts/smooth-scrolling.html
- http://developer.android.com/training/contacts-provider/display-contact-badge.html#ListView

ForecastAdapter.java

```
22      /**
23       * Cache of the children views for a forecast list item.
24       */
25      public static class ViewHolder {
26          public final ImageView iconView;
27          public final TextView dateView;
28          public final TextView descriptionView;
29          public final TextView highTempView;
30          public final TextView lowTempView;
31
32          public ViewHolder(View view) {
33              iconView = (ImageView) view.findViewById(R.id.list_item_icon);
34              dateView = (TextView) view.findViewById(R.id.list_item_date_textview);
35              descriptionView = (TextView) view.findViewById(R.id.list_item_forecast_textview);
36              highTempView = (TextView) view.findViewById(R.id.list_item_high_textview);
37              lowTempView = (TextView) view.findViewById(R.id.list_item_low_textview);
38          }
39      }
```

Formatting Strings

DON'T:

```
textview.setText(temperature + "°");
```

DO:

```
<string name="welcome">Hi <xliff:g id="name">%1$s</xliff:g>!
    You have <xliff:g id="count">%2$d</xliff:g> new messages.</string>
```

```
<!-- Date format for displaying day of week and date (i.e. Mon Jun 1) [CHAR LIMIT=20] -->
<string name="format_full_friendly_date"><xliff:g id="day">%1$s</xliff:g>, <xliff:g id="date">%2$

<!-- Strings for formatting weather-related data -->
<!-- Temperature format [CHAR LIMIT=5 -->
<string name="format_temperature"><xliff:g id="temp">%1.0f</xliff:g>\u00B0</string>
```

Formatting Strings

strings.xml × | ForecastAdapter.java × | Utility.java ×

```
41  static String formatTemperature(Context context, double temperature, boolean isMetric) {
42      double temp;
43      if ( !isMetric ) {
44          temp = 9*temperature/5+32;
45      } else {
46          temp = temperature;
47      }
48      return context.getString(R.string.format_temperature, temp);
49  }
50
51  static String formatDate(long dateInMilliseconds) {
52      Date date = new Date(dateInMilliseconds);
53      return DateFormat.getDateInstance().format(date);
54  }
55
56  // Format used for storing dates in the database.  ALso used for converting those strings
57  // back into date objects for comparison/processing.
58  public static final String DATE_FORMAT = "yyyyMMdd";
59
60  /**
61   * Helper method to convert the database representation of the date into something to display
62   * to users.  As classy and polished a user experience as "20140102" is, we can do better.
63   *
64   * @param context Context to use for resource localization
65   * @param dateInMillis The date in milliseconds
66   * @return a user-friendly representation of the date.
```

Details Screen

Details Screen

- [ ] Build up layout XML
- [ ] Update DetailFragment to populate new views

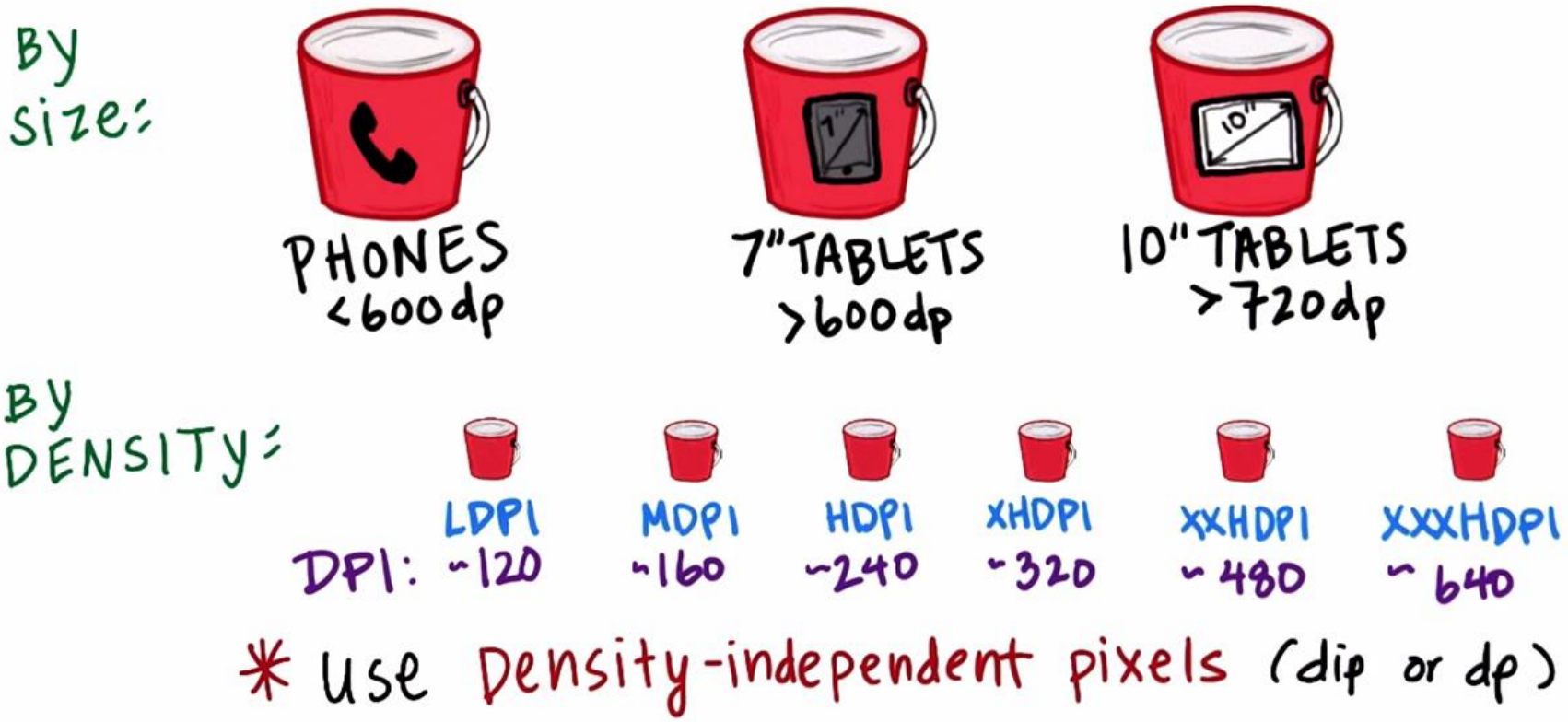
Responsive Design

- **Tablet App Quality:** http://developer.android.com/distribute/essentials/quality/tablets.html
- **Supporting Tablets and Handsets:** http://developer.android.com/guide/practices/tablets-and-handsets.html

- **Responsive design means designing your app by keeping in mind that it'll be used across a range of different device screen sizes.**
- **Apps adapt using multi-pane layouts.**

Splitting Devices into Buckets

- **Configuration examples:** http://developer.android.com/guide/practices/screens_support.html#ConfigurationExamples

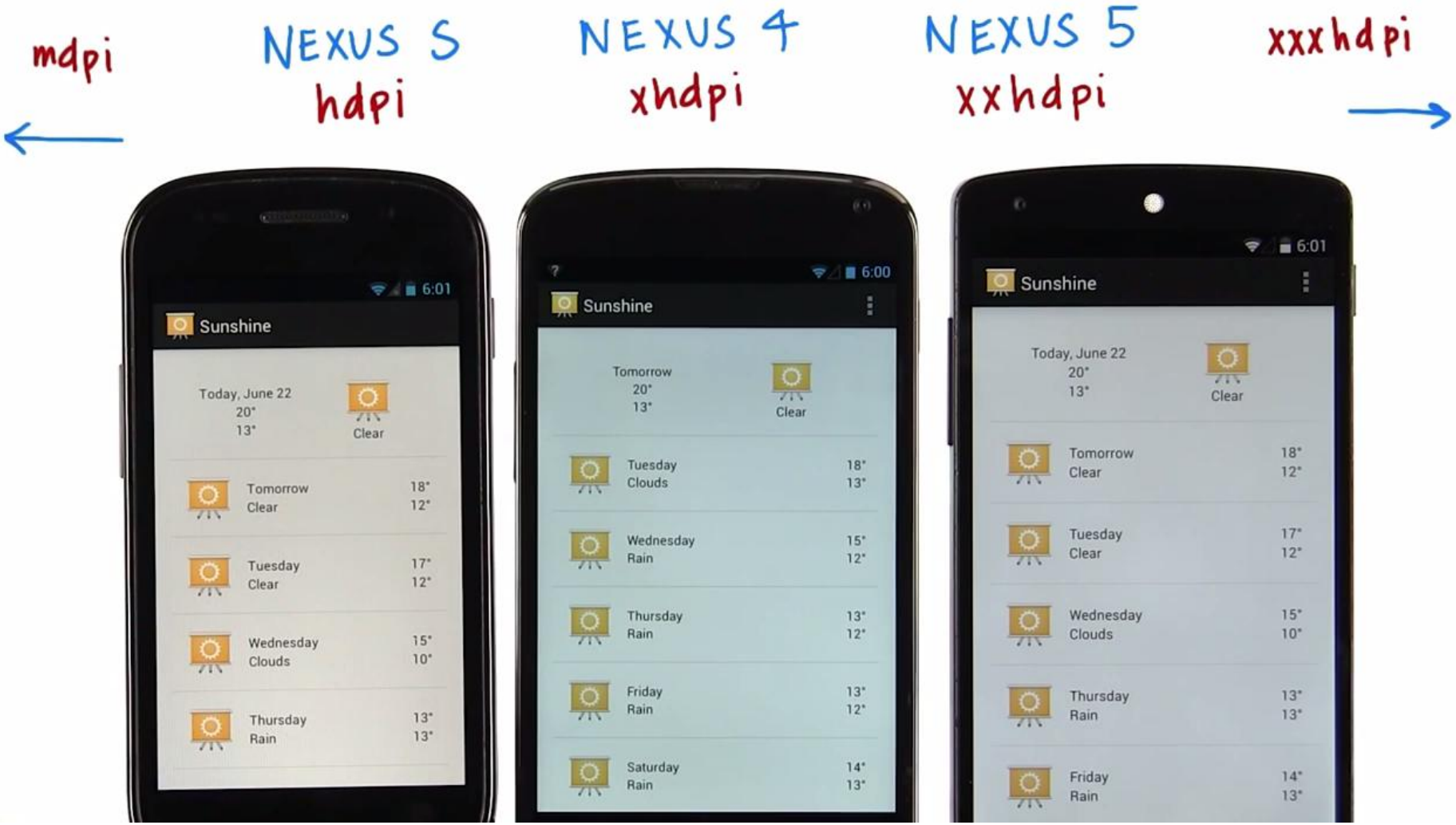


Resource Folder Qualifiers

- **Android allows you to create alternative versions of every resource by placing them into folders with different qualifiers.**
- **Android activities are destroyed and recreated whenever the device configuration changes.**
- **http://developer.android.com/guide/practices/screens_support.html**

```
res/
    values/
        strings.xml
    layout/
        main_activity.xml
    values-fr/
    values-fr-rCA/
    layout-desk/
    layout-stylus/
    drawable-xhdpi/
    layout-land/
    layout-sw720dp/
```

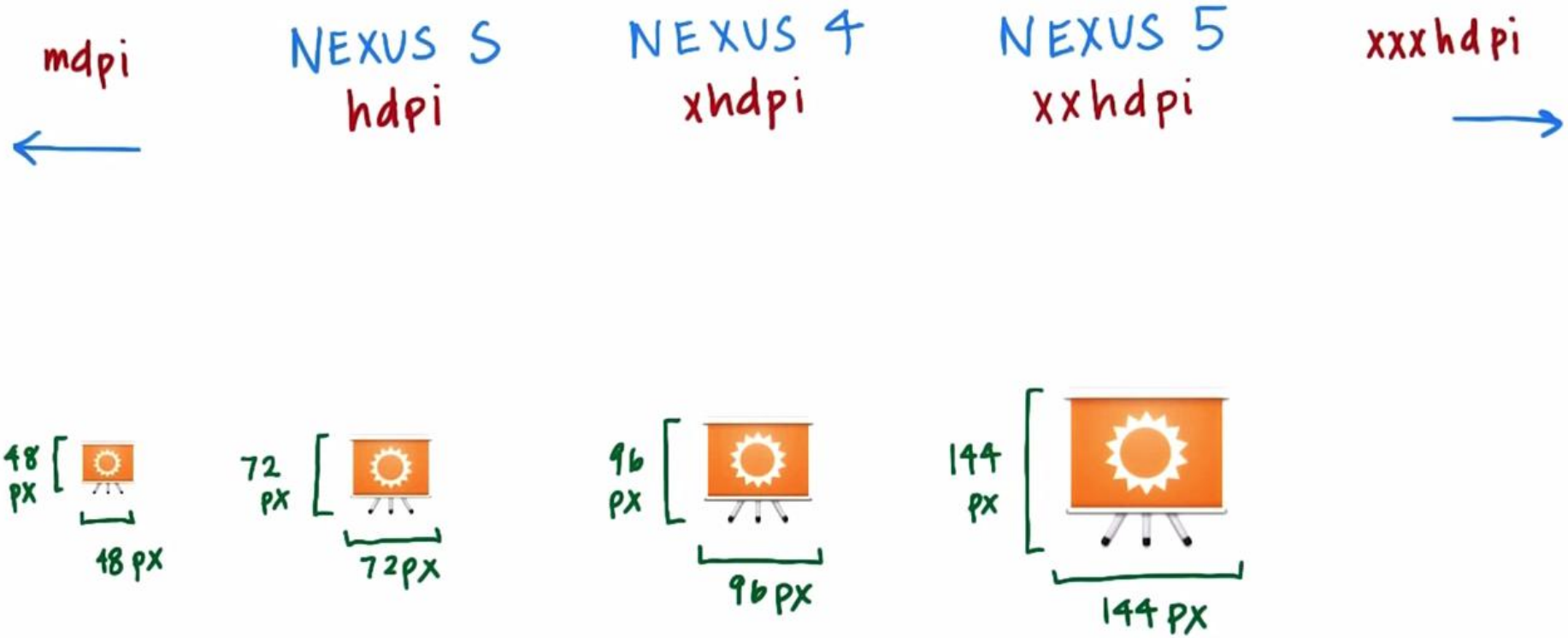
Images for Different Densities



Devices and Displays

- http://developer.android.com/design/style/devices-displays.html

Images for Different Densities



Adding Images to the App

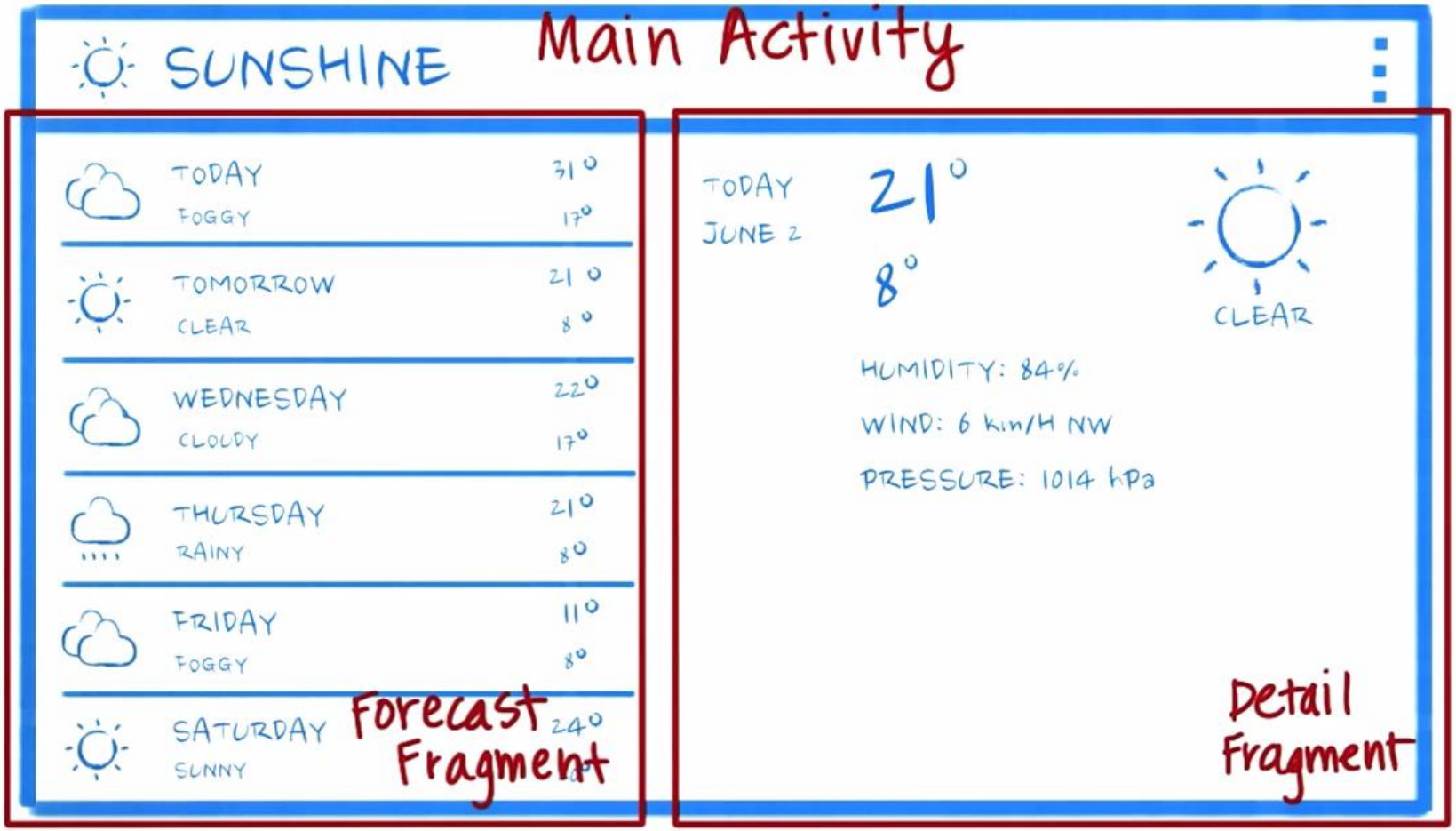
- ☐ Download Sunshine assets
- ☐ Update launcher icon
- ☐ Integrate w/app in forecast list and forecast detail screen

weather condition code → icon

Building a Flexible UI

- http://developer.android.com/training/basics/fragments/fragment-ui.html

Tablet UX Mocks



Why Do We Need Fragments?

- **They allow us to fully modularize our activities, including the life cycle events.**
- **Android didn't always support embedding activities within other activities.**

Why We Don't Only Use Fragments

How Fragments Work

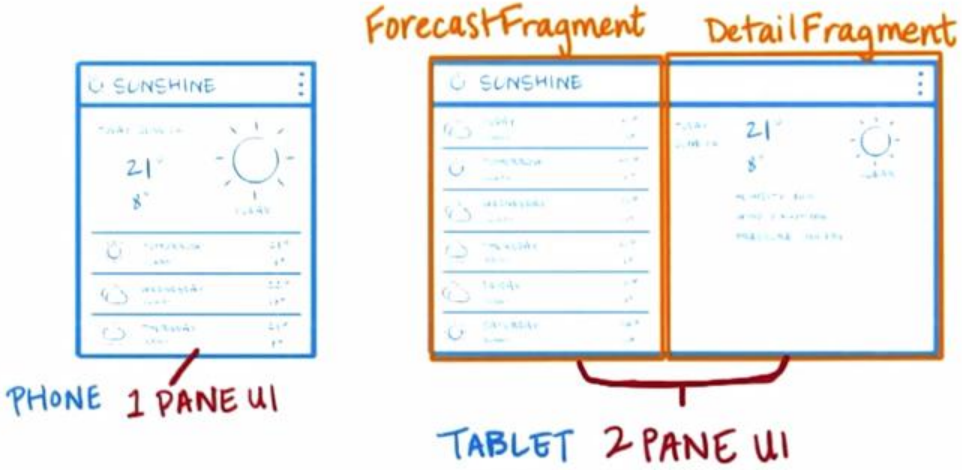
Try the Fragment Manager

Fragments with No UI

Sunshine Resource Folders

- http://developer.android.com/guide/practices/screens_support.html#ConfigurationExamples

Which layout folders should contain an activity_main.xml file?



- ☐ layout
- ☐ layout-land
- ☐ layout-sw600dp
- ☐ layout-sw720dp

Smallest Width (SW) Qualifier

Directory structure:

res/layout
- main.xml
- detail.xml
- item.xml

res/layout-sw600dp
- detail.xml
- item.xml

res/layout-sw 720dp
- item.xml

Smallest Width (SW) Qualifier

Directory structure:

```
res/layout
    main.xml
    detail.xml
    item.xml
res/layout-sw600dp
    detail.xml
    item.xml
res/layout-sw 720dp
    item.xml
```

Smallest Width (SW) Qualifier

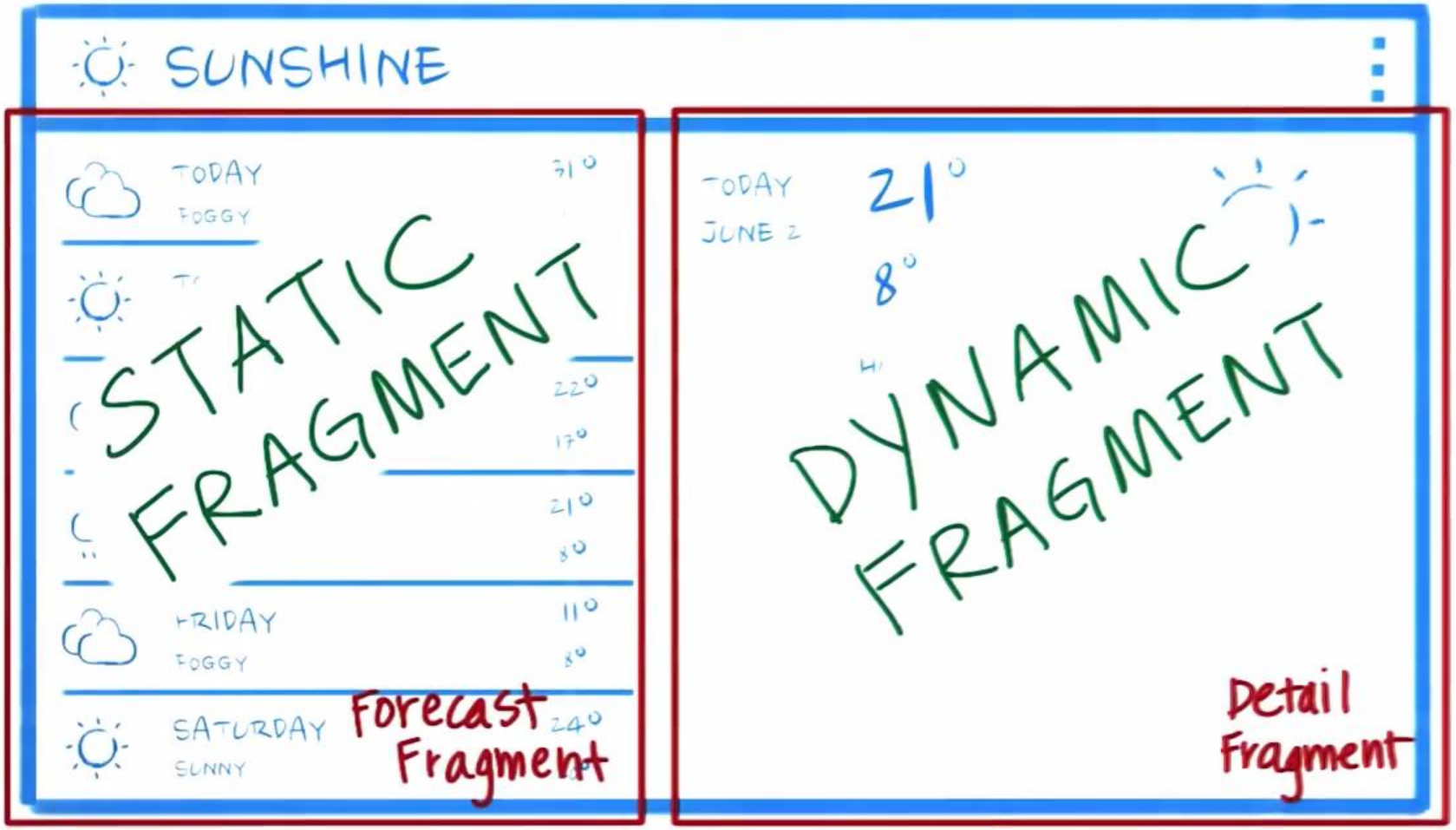
Directory structure:

res/layout
- main.xml
- detail.xml
- item.xml

res/layout-sw600dp
- detail.xml
- item.xml

res/layout-sw 720dp
- item.xml

Build 2-Pane Tablet UI



Build 2-Pane Tablet UI

1. **Remove values-w820dp folder**
2. **Make two activity_main.xml files**
3. **Make the base activity_main.xml and activity_detail match the 2-pane UI xml**
4. **Update DetailActivity to add the detail fragment**
5. **Fix Tags and onResume in MainActivity**
6. **Update MainActivity so that it sets up the layout correctly and knows whether it's one or two pane mode**
7. **Put placeholder data in the detail_fragment xml**
8. **Modify the DetailFragment to handle the case where it received no Uri**

Create the 2-Pane UI Layout

Sunshine-Version-2-5.09_two_pane_ui > app > src > main > res > layout-sw600dp > activity_main.xml

layout\activity_main.xml × | sw600dp\activity_main.xml ×

```
9
10      <!--
11      This layout is a two-pane layout for the Items master/detail flow.
12      -->
13
14      <fragment
15          android:id="@+id/fragment_forecast"
16          android:name="com.example.android.sunshine.app.ForecastFragment"
17          android:layout_width="0dp"
18          android:layout_height="match_parent"
19          android:layout_weight="2"
20          tools:layout="@android:layout/list_content" />
21
22      <FrameLayout
23          android:id="@+id/weather_detail_container"
24          android:layout_width="0dp"
25          android:layout_height="match_parent"
26          android:layout_weight="4" />
27
28  </LinearLayout>
```

Update the 1-Pane UI Layout

Sunshine-Version-2-5.09_two_pane_ui › app › src › main › res › layout › activity_main.xml

layout\activity_main.xml × | sw600dp\activity_main.xml ×

```
1  <fragment xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:id="@+id/fragment_forecast"
4      android:name="com.example.android.sunshine.app.ForecastFragment"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:layout_marginLeft="16dp"
8      android:layout_marginRight="16dp"
9      tools:context="com.example.android.sunshine.app.ForecastFragment"
10     tools:layout="@android:layout/list_content" />
```

Remove ForecastFragment

- **Since the fragment is already inside the main activity layout we can just remove this so we don't dynamically add it again.**

Sunshine-Version-2-5.09_two_pane_ui › app › src › main › java › com › example › android › sunshine › app › MainActivity

MainActivity.java × | layout\activity_main.xml × | sw600dp\activity_main.xml ×

```
34      @Override
35      protected void onCreate(Bundle savedInstanceState) {
36          super.onCreate(savedInstanceState);
37          mLocation = Utility.getPreferredLocation(this);
38
39          setContentView(R.layout.activity_main);
40
41      }
42
43      @Override
44      public boolean onCreateOptionsMenu(Menu menu) {
45          // Inflate the menu; this adds items to the action bar if it is present.
46          getMenuInflater().inflate(R.menu.main, menu);
47          return true;
48      }
```

Modify the Activity Detail Layout

- **We change the frame layout ID to be weather detail container so that it matches the container view id in the two pane UI case.**

Sunshine-Version-2-5.09_two_pane_ui › app › src › main › res › layout › activity_detail.xml

MainActivity.java × | layout\activity_main.xml × | sw600dp\activity_main.xml × | activity_detail.xml ×

```
1  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:id="@+id/weather_detail_container"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      tools:context="com.example.android.sunshine.app.DetailActivity"
7      tools:ignore="MergeRootFrame" />
```

Update the Detail Activity

- **Since we changed the name of the container, we should also update the detail activity. This is only used in one pane mode.**

Sunshine-Version-2-5.09_two_pane_ui › app › src › main › java › com › example › android › sunshine › app › DetailActivity

MainActivity.java × | layout\activity_main.xml × | DetailActivity.java × | sw600dp\activity_main.xml × | activity_detail.xml ×

```
24
25 public class DetailActivity extends ActionBarActivity {
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.activity_detail);
31         if (savedInstanceState == null) {
32             getSupportFragmentManager().beginTransaction()
33                     .add(R.id.weather_detail_container, new DetailFragment())
34                     .commit();
35         }
36     }
```

Fix Tags and onResume in MainActivity

- **Create `the private static final String DETAILFRAGMENT_TAG = "DFTAG"` in MainActivity.**
- **We no longer need the `FORCASTFRAGMENT` tag since we're no longer explicitly creating the ForcastFragment.**
- **We need to update on resume to have this line instead:**

```
ForecastFragment ff =
  (ForecastFragment)getSupportFragmentManager()
  .findFragmentById(R.id.fragment_forecast);
```

Update the Main Activity

- **So we dynamically add the detail fragment.**

Sunshine-Version-2-5.09_two_pane_ui › app › src › main › java › com › example › android › sunshine › app › MainActivity

MainActivity.java × | layout\activity_main.xml × | DetailActivity.java × | sw600dp\activity_main.xml × | activity_detail.xml ×

```
35    protected void onCreate(Bundle savedInstanceState) {
36        super.onCreate(savedInstanceState);
37        mLocation = Utility.getPreferredLocation(this);
38
39        setContentView(R.layout.activity_main);
40        if (findViewById(R.id.weather_detail_container) != null) {
41            // The detail container view will be present only in the large-screen layouts
42            // (res/layout-sw600dp). If this view is present, then the activity should be
43            // in two-pane mode.
44            mTwoPane = true;
45            // In two-pane mode, show the detail view in this activity by
46            // adding or replacing the detail fragment using a
47            // fragment transaction.
48            if (savedInstanceState == null) {
49                getSupportFragmentManager().beginTransaction()
50                        .replace(R.id.weather_detail_container, new DetailFragment(), DETAILFRAGMENT
51                        .commit();
52            }
53        } else {
54            mTwoPane = false;
55        }
56    }
```

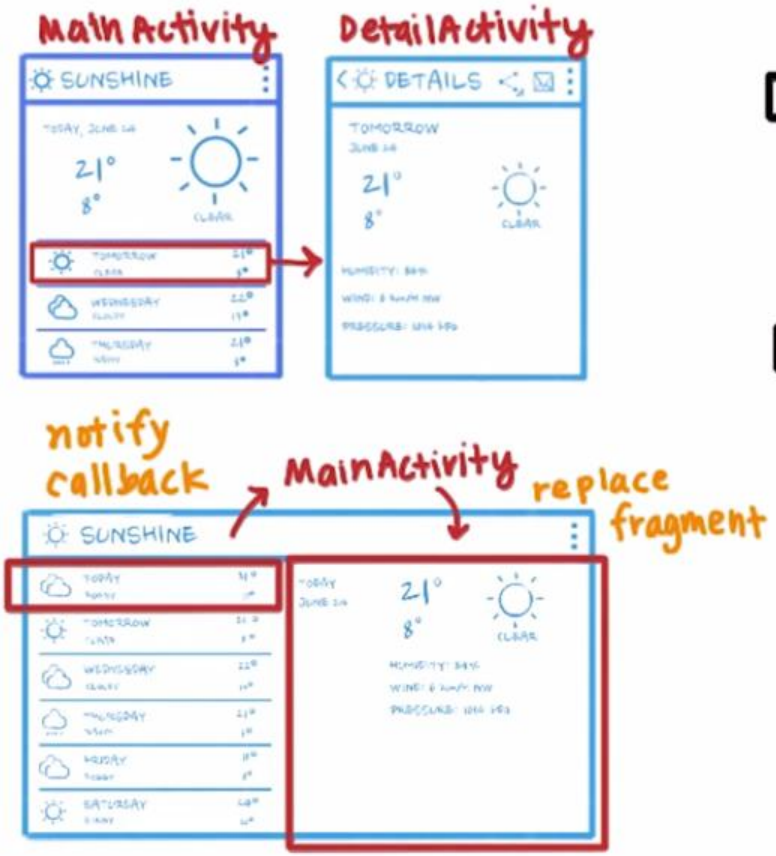
- **Put placeholder data in the detail_fragment xml:**
  - If you want to see something in the detail fragment to check that it's loading correctly, add some dummy data.
- **Modify the DetailFragment to handle the case where it received no Uri:**
  - If DetailFragment is created without a uri (as in intent.data() == null), it should not try to create a loader.

Update Fragment Detail Layout

TWO-PANE TABLET UI

- [ ] Create activity_main.xml in layout-sw600dp folder, update other layout files
- [ ] In MainActivity, dynamically add DetailFragment (but not ForecastFragment)
- [ ] Display hardcoded data in DetailFragment even without incoming intent

Handle List Item Click



- Use Callback to notify activity of list item selection
- On phone, launch DetailActivity
  on tablet, replace DetailFragment

Add the `Callback` Interface

- **In the `ForecastFragment` class, we add the `Callback` interface.**

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › ForecastFragment

ForecastFragment.java ×

```
76      /**
77       * A callback interface that all activities containing this fragment must
78       * implement. This mechanism allows activities to be notified of item
79       * selections.
80       */
81      public interface Callback {
82          /**
83           * DetailFragmentCallback for when an item has been selected.
84           */
85          public void onItemSelected(Uri dateUri);
86      }
```

Modify the List Item Click Behavior

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › ForecastFragment

ForecastFragment.java ×

```
117     public View onCreateView(LayoutInflater inflater, ViewGroup container,
118                              Bundle savedInstanceState) {
119         // The CursorAdapter will take data from our cursor and populate the ListView.
120         mForecastAdapter = new ForecastAdapter(getActivity(), null, 0);
121
122         View rootView = inflater.inflate(R.layout.fragment_main, container, false);
123
124         // Get a reference to the ListView, and attach this adapter to it.
125         ListView listView = (ListView) rootView.findViewById(R.id.listview_forecast);
126         listView.setAdapter(mForecastAdapter);
127
128         // We'll call our MainActivity
129         listView.setOnItemClickListener((adapterView, view, position, l) → {
133                 // CursorAdapter returns a cursor at the correct position for getItem(), or null
134                 // if it cannot seek to that position.
135                 Cursor cursor = (Cursor) adapterView.getItemAtPosition(position);
136                 if (cursor != null) {
137                     String locationSetting = Utility.getPreferredLocation(getActivity());
138                     ((Callback) getActivity())
139                             .onItemSelected(WeatherContract.WeatherEntry.buildWeatherLocationWithDat
140                                     locationSetting, cursor.getLong(COL_WEATHER_DATE)
141                             ));
142                 }
143         });
145         return rootView;
```

Implement the `Callback` Interface

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › MainActivity

ForecastFragment.java × | MainActivity.java ×

```
123     @Override
124     public void onItemSelected(Uri contentUri) {
125         if (mTwoPane) {
126             // In two-pane mode, show the detail view in this activity by
127             // adding or replacing the detail fragment using a
128             // fragment transaction.
129             Bundle args = new Bundle();
130             args.putParcelable(DetailFragment.DETAIL_URI, contentUri);
131
132             DetailFragment fragment = new DetailFragment();
133             fragment.setArguments(args);
134
135             getSupportFragmentManager().beginTransaction()
136                     .replace(R.id.weather_detail_container, fragment, DETAILFRAGMENT_TAG)
137                     .commit();
138         } else {
139             Intent intent = new Intent(this, DetailActivity.class)
140                     .setData(contentUri);
141             startActivity(intent);
142         }
143     }
144 }
```

Modify the `onCreate` Method

- to read the data URI from the incoming intent
- take the URI and set it as an arguments in the new `DetailFragment`

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › DetailActivity

ForecastFragment.java × | MainActivity.java × | DetailActivity.java ×

```
28    protected void onCreate(Bundle savedInstanceState) {
29        super.onCreate(savedInstanceState);
30        setContentView(R.layout.activity_detail);
31
32        if (savedInstanceState == null) {
33            // Create the detail fragment and add it to the activity
34            // using a fragment transaction.
35
36            Bundle arguments = new Bundle();
37            arguments.putParcelable(DetailFragment.DETAIL_URI, getIntent().getData());
38
39            DetailFragment fragment = new DetailFragment();
40            fragment.setArguments(arguments);
41
42            getSupportFragmentManager().beginTransaction()
43                    .add(R.id.weather_detail_container, fragment)
44                    .commit();
```

Modify the `onCreateView` Method

- **read the arguments that the fragment was initialized with**
- **read the data URI and store that in the member variable `mUri`**

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › DetailFragment

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java ×

```
100     public View onCreateView(LayoutInflater inflater, ViewGroup container,
101                              Bundle savedInstanceState) {
102
103         Bundle arguments = getArguments();
104         if (arguments != null) {
105             mUri = arguments.getParcelable(DetailFragment.DETAIL_URI);
106         }
107
108         View rootView = inflater.inflate(R.layout.fragment_detail, container, false);
109         mIconView = (ImageView) rootView.findViewById(R.id.detail_icon);
110         mDateView = (TextView) rootView.findViewById(R.id.detail_date_textview);
111         mFriendlyDateView = (TextView) rootView.findViewById(R.id.detail_day_textview);
112         mDescriptionView = (TextView) rootView.findViewById(R.id.detail_forecast_textview);
113         mHighTempView = (TextView) rootView.findViewById(R.id.detail_high_textview);
114         mLowTempView = (TextView) rootView.findViewById(R.id.detail_low_textview);
```

Modify the `onCreateLoader` Method

- **remove some code that relied on incoming intent, and switch it to use `mUri` instead.**
- **Whether this `DetailFragment` is in the main activity or the `DetailActivity`, it should have mUri set.**

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › DetailFragment

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java ×

```
163     @Override
164     public Loader<Cursor> onCreateLoader(int id, Bundle args) {
165         if ( null != mUri ) {
166             // Now create and return a CursorLoader that will take care of
167             // creating a Cursor for the data being displayed.
168             return new CursorLoader(
169                     getActivity(),
170                     mUri,
171                     DETAIL_COLUMNS,
172                     null,
173                     null,
174                     null
175             );
176         }
177         return null;
```

Fix Navigation Bug

- **If you navigate to the settings activity, and then hit the up button, the detail fragment is blank.**
- **That's because a brand new instance of main activity got launched, and for a brand new instance, nothing is selected yet.**

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › SettingsActivity

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java × | SettingsActivity.java ×

```
51      @TargetApi(Build.VERSION_CODES.JELLY_BEAN)
52      @Override
53      public Intent getParentActivityIntent() {
54          return super.getParentActivityIntent().addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
55      }
```

Detect that the Location Has Changed

- **If the user went to Settings and entered in a new location, when they come back to the main activity.**

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › MainActivity

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java × | SettingsActivity.java ×

```
105     @Override
106     protected void onResume() {
107         super.onResume();
108         String location = Utility.getPreferredLocation( this );
109         // update the location in our second pane using the fragment manager
110             if (location != null && !location.equals(mLocation)) {
111             ForecastFragment ff = (ForecastFragment)getSupportFragmentManager().findFragmentById(R.i
112             if ( null != ff ) {
113                 ff.onLocationChanged();
114             }
115             DetailFragment df = (DetailFragment)getSupportFragmentManager().findFragmentByTag(DETAIL
116             if ( null != df ) {
117                 df.onLocationChanged(location);
118             }
119             mLocation = location;
120         }
121     }
```

Add a new method onLocationChanged

- create a new Uri with the new location
- update the nUri and then restart the loader

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › DetailFragment

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java × | SettingsActivity.java ×

```
152     void onLocationChanged( String newLocation ) {
153         // replace the uri, since the location has changed
154         Uri uri = mUri;
155         if (null != uri) {
156             long date = WeatherContract.WeatherEntry.getDateFromUri(uri);
157             Uri updatedUri = WeatherContract.WeatherEntry.buildWeatherLocationWithDate(newLocation,
158             mUri = updatedUri;
159             getLoaderManager().restartLoader(DETAIL_LOADER, null, this);
160         }
161     }
```

Sunshine-Version-2-5.10_selected_item › app › src › main › java › com › example › android › sunshine › app › ForecastFragment

ForecastFragment.java × | MainActivity.java × | DetailActivity.java × | DetailFragment.java × | SettingsActivity.java ×

```
154     // since we read the location when we create the loader, all we need to do is restart things
155     void onLocationChanged( ) {
156         updateWeather();
157         getLoaderManager().restartLoader(FORECAST_LOADER, null, this);
158     }
```

Activated List Item Style

- http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList

Show activated state on selected item

XML file saved at `res/drawable/button.xml` :

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>
```

This layout XML applies the state list drawable to a Button:

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

The StateList Drawable

Sunshine › app › src › main › res › drawable › touch_selector.xml

touch_selector.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <selector xmlns:android="http://schemas.android.com/ap
4
5      <!-- State when a row is being pressed, but hasn't
6      <item android:state_pressed="true"
7          android:drawable="@color/grey" />
8
9      <!-- When the view is "activated".  In SINGLE_CHOI
10         of a ListView -->
11     <item android:state_activated="true"
12         android:drawable="@color/sunshine_light_blue"
13
14     <!-- Default, "just hangin' out" state. -->
15     <item android:drawable="@android:color/transparent
16 </selector>
```

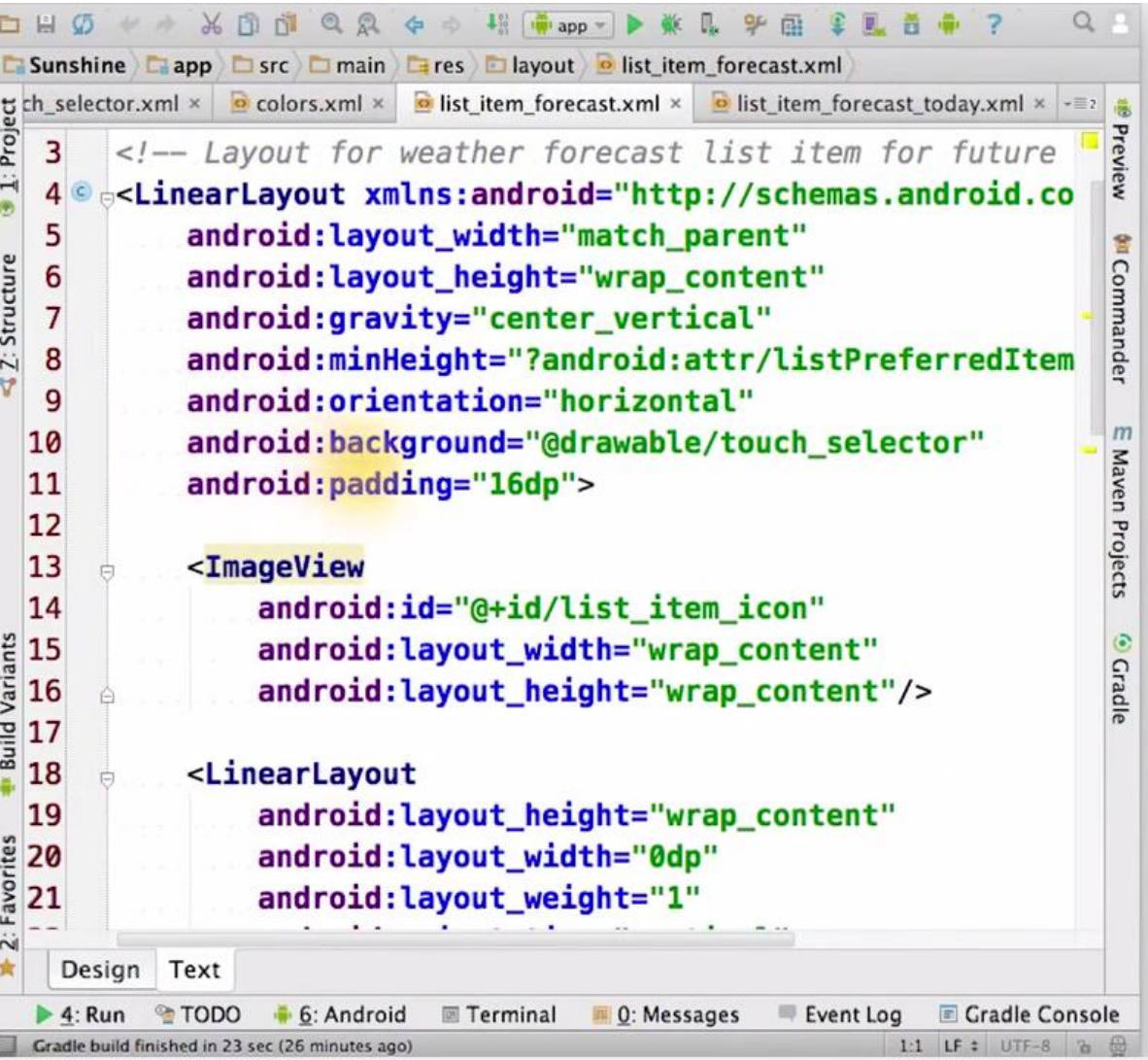
The StateList Drawable – v21

Sunshine › app › src › main › res › drawable-v21 › touch_selector.xml

touch_selector.xml | v21/touch_selector.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <selector xmlns:android="http://schemas.android.com/ap
4
5      <!-- State when a row is being pressed, but hasn't
6      <item android:state_pressed="true">
7          <ripple android:color="@color/grey" />
8      </item>
9
10     <!-- When the view is "activated".  In SINGLE_CHOI
11         of a ListView -->
12     <item android:state_activated="true"
13         android:drawable="@color/sunshine_light_blue"
14
15     <!-- Default, "just hangin' out" state. -->
16     <item android:drawable="@android:color/transparent
17 </selector>
```

Set the List Item Background



Define Colors

- The `@android` color notation is for a framework defined color.
- The other colors are ones that we defined in the `colors.xml` file.

Sunshine-Version-2-5.11_activated_item > app > src > main > res > values > colors.xml

touch_selector.xml × | list_item_forecast.xml × | list_item_forecast_today.xml × | colors.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <color name="grey">#cccccc</color>
5      <color name="sunshine_light_blue">#ff64c2f4</color>
6
7  </resources>
```

Define Colors

- The `@android` color notation is for a framework defined color.
- The other colors are ones that we defined in the `colors.xml` file.

Sunshine-Version-2-5.11_activated_item > app > src > main > res > values > colors.xml

touch_selector.xml × | list_item_forecast.xml × | list_item_forecast_today.xml × | colors.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3
4      <color name="grey">#cccccc</color>
5      <color name="sunshine_light_blue">#ff64c2f4</color>
6
7  </resources>
```

Styles

- http://developer.android.com/guide/topics/ui/themes.html
- **Styles are a way that you can group together attributes for a view.**
- **You can also have styles inherit other styles by specifying the parent.**

Style Example

For example, by using a style, you can take this layout XML:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

And turn it into this:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

Defining Styles

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

1-Pane vs. 2-Pane Mode

Sunshine-Version-2-5.11_activated_item › app › src › main › res › layout › fragment_main.xml

values\styles.xml × | sw600dp\styles.xml × | fragment_main.xml ×

```
12      <ListView
13          style="@style/ForecastListStyle"
14          android:id="@+id/listview_forecast"
15          android:layout_width="match_parent"
16          android:layout_height="match_parent" />
17
18  </FrameLayout>
```

Sunshine-Version-2-5.11_activated_item › app › src › main › res › values › styles.xml

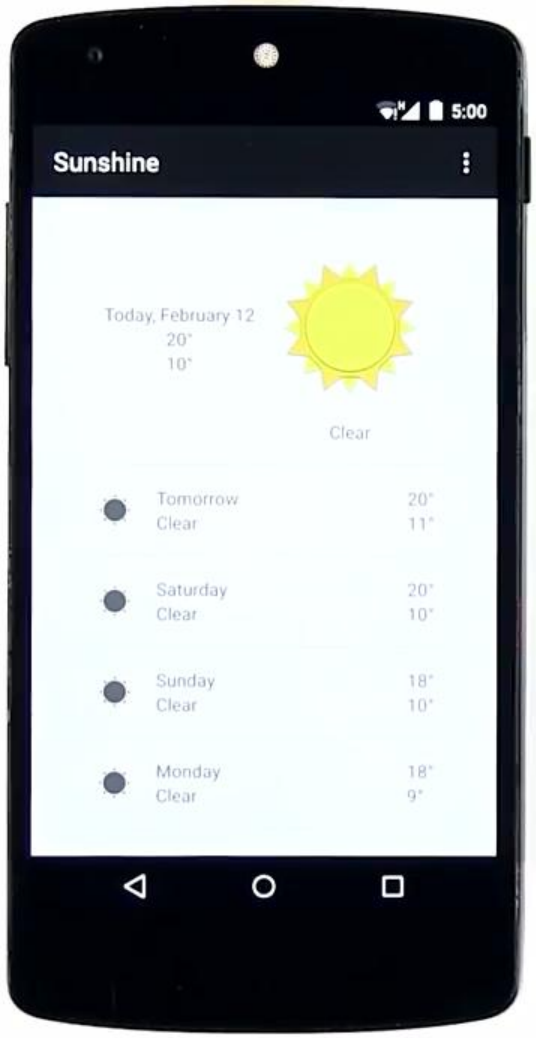
values\styles.xml × | sw600dp\styles.xml × | fragment_main.xml ×

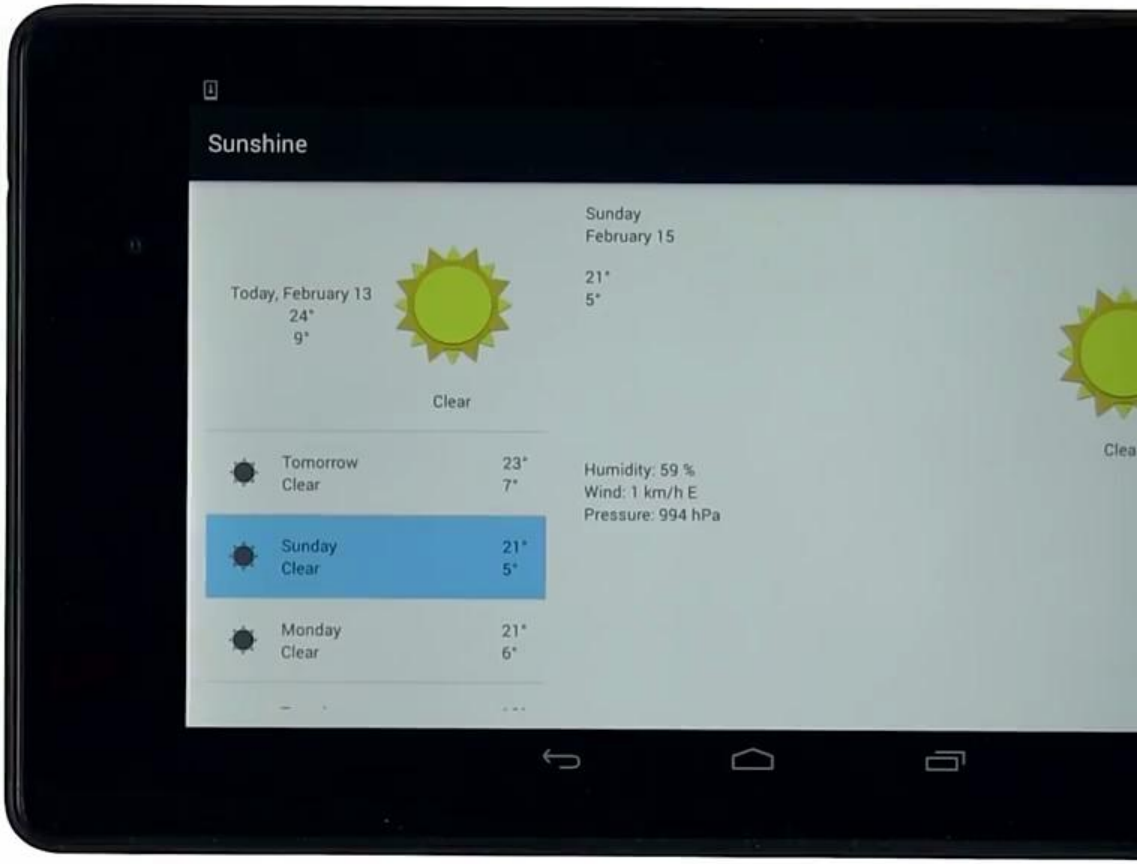
```
8       <style name="ForecastListStyle">
9           <!-- Here it's a stub.  In screens of sufficient width, this style includes modifications
10          for two-pane layout -->
11      </style>
```

Sunshine-Version-2-5.11_activated_item › app › src › main › res › values-sw600dp › styles.xml

values\styles.xml × | sw600dp\styles.xml × | fragment_main.xml ×

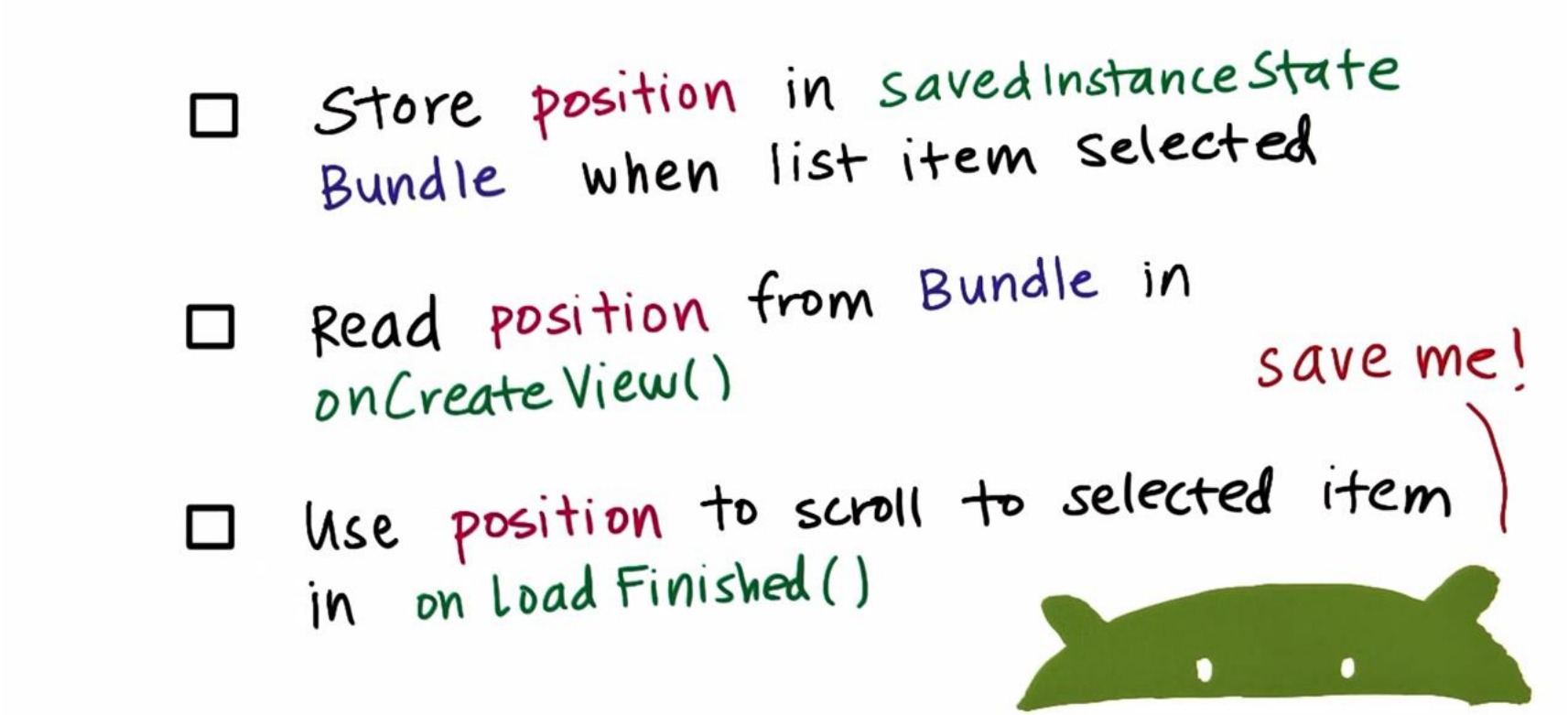
```
1   <resources>
2
3       <style name="ForecastListStyle">
4           <item name="android:choiceMode">singleChoice</item>
5       </style>
6
7   </resources>
```


5:00
Sunshine
Today, February 12
20°
10°
Clear
Tomorrow
Clear
20°
11°
Saturday
Clear
20°
10°
Sunday
Clear
18°
10°
Monday
Clear
18°
9°


Sunshine
Today, February 13
24°
9°
Clear
Sunday
February 15
21°
5°
Tomorrow
Clear
23°
7°
Sunday
Clear
21°
5°
Monday
Clear
21°
6°
Humidity: 59 %
Wind: 1 km/h E
Pressure: 994 hPa

Restore Scroll Position on Rotation

- **On the tablet, there's a bug where if you select an item near the bottom of the list and then you rotate the device, the selected item is no longer scrolled into view.**



Restore Scroll Position on Rotation

- **In the forecast fragment, we create a position variable.**
- **Whenever an item is clicked, we update the position.**
- **Then, in the `onSaveInstanceState` method, we store the position in the bundle.**
- **If the app gets killed, then we can restore the position from the save state bundle. (`onCreateView` method)**
- **We wait for the `onLoadFinished` callback to happen when the cursor is swapped. Then, we can tell the list view to set selection on that position, and that position will be scrolled into view.**

Alternate Detail Layout

Which layout folders would we need to define fragment_detail.xml ?

Wide Detail Fragment

Which layout folders would we need to define fragment_detail.xml ?

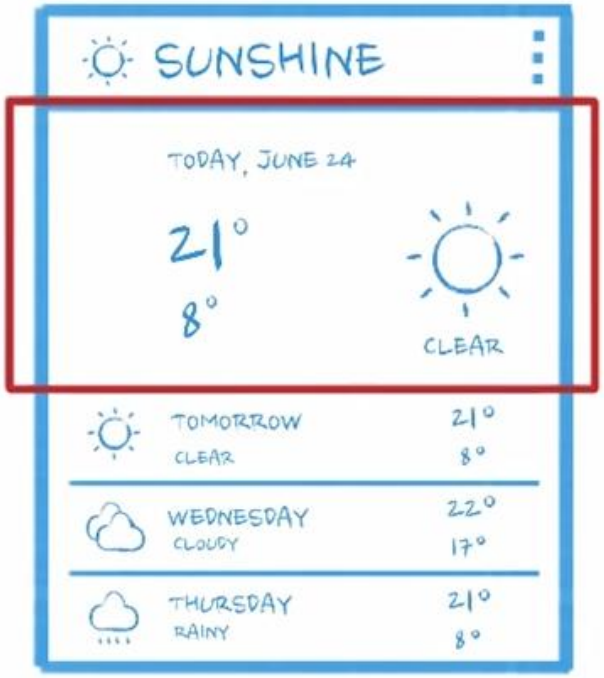
- [x] layout
- [x] layout-land
- [x] layout-sw600dp
- [ ] layout-sw720dp

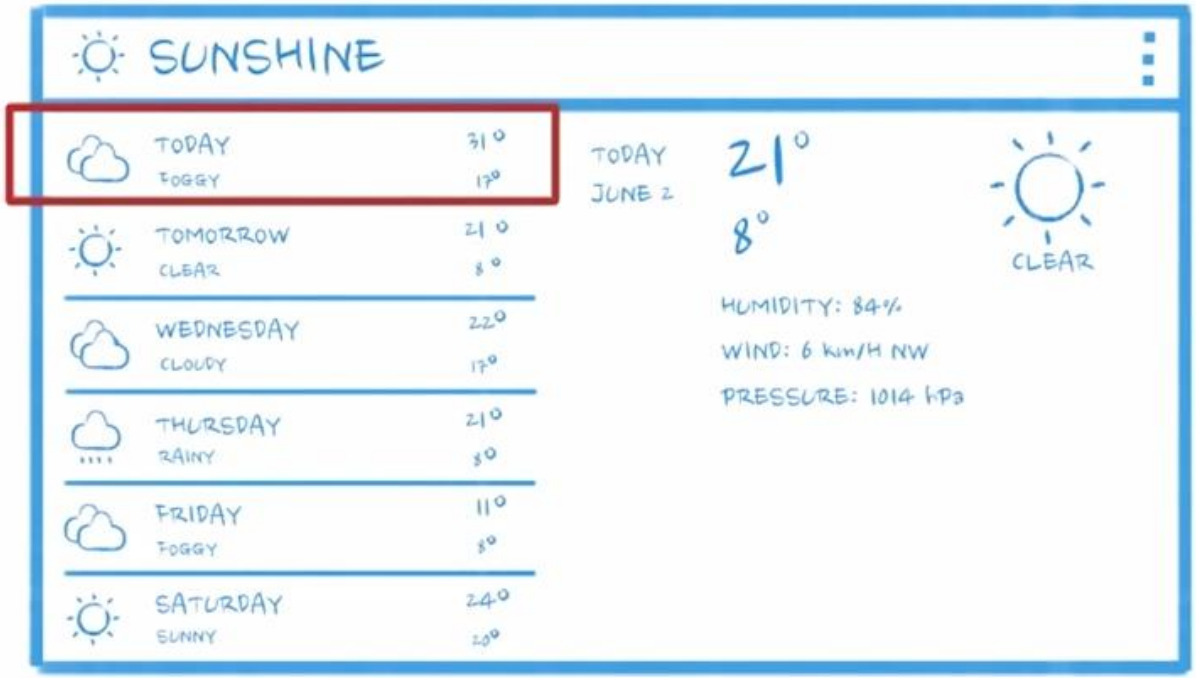
Use Layout Aliases

- http://developer.android.com/training/multiscreen/screensizes.html#TaskUseAliasFilters
- **We had to define the wide detail fragment layout in two places: `layout-land` and `layout-sw600dp`.**
- **To avoid having multiple copies of this file, we should use layout aliasing.**

Today Item on Tablet

- **In the phone layout, we wanted the today item to be a little bit bigger. But on the tablets, we just want it to look like all the other items.**





Modify the `getItemViewType` Method

ment.java × | MainActivity.java ×

```
    ForecastFragment forecastFragment =  ((ForecastFragment)getSupportFragmentManager()
            .findFragmentById(R.id.fragment_forecast));
    forecastFragment.setUseTodayLayout(!mTwoPane);
```

java × | ForecastFragment.java ×

```
public void setUseTodayLayout(boolean useTodayLayout) {
    mUseTodayLayout = useTodayLayout;
    if (mForecastAdapter != null) {
        mForecastAdapter.setUseTodayLayout(mUseTodayLayout);
    }
}
```

java × | ForecastFragment.java × | ForecastAdapter.java ×

```
public void setUseTodayLayout(boolean useTodayLayout) { mUseTodayLayout = useTodayLayout; }

@Override
public int getItemViewType(int position) {
    return (position == 0 && mUseTodayLayout) ? VIEW_TYPE_TODAY : VIEW_TYPE_FUTURE_DAY;
}
```