App Development
in Android Studio

HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

Lab 1: Views and events

November 28, 2017

Jón Steinn Elíasson

jonsteinn@gmail.com

Contents

1 Extensible Markup Language

Extensible Markup Language (XML) is a markup language, similar to HTML. The main difference being that HTML is designed to display predefined tags while XML is designed to carry arbitrary data. XML documents are structured as trees (as can be seen in the following example) and are required to have a single root element. A prolog prior to the root element defines the XML version and encoding.

```
<?xml version="1.0" encoding="UTF-8"?>
<root_element>
    <element>
        <leaf_element>
        </leaf_element>
    </element>
    <element>
        <element>
            <leaf_element>
            </leaf_element>
        </element>
    </element>
</root_element>
```

In Android we use XML to define structure of UI and attributes. Elements can have attributes, which are kept in quotation marks and can describe properties such as id, size and text for elements like buttons and input fields.

```
<element attribute1="value1" attribute2="value2"/>
```

We also use XML to store information in manifest, UI strings, drawables, styles and layouts. In the program we did in lab 0, we created an activity and with it came a layout XML document. We will take a better look at activities later but for now, we can think of them as a single screen and this layout holds info about its UI components.

2 Views

The `View` class is the base unit for user interface components. It is responsible for drawing on screen and handling events. It is the root class in the inheritance hierarchy of UI components (disregarding the `Object` class). Views can be visible component displayed on our screen or invisible containers called view groups to store other views.

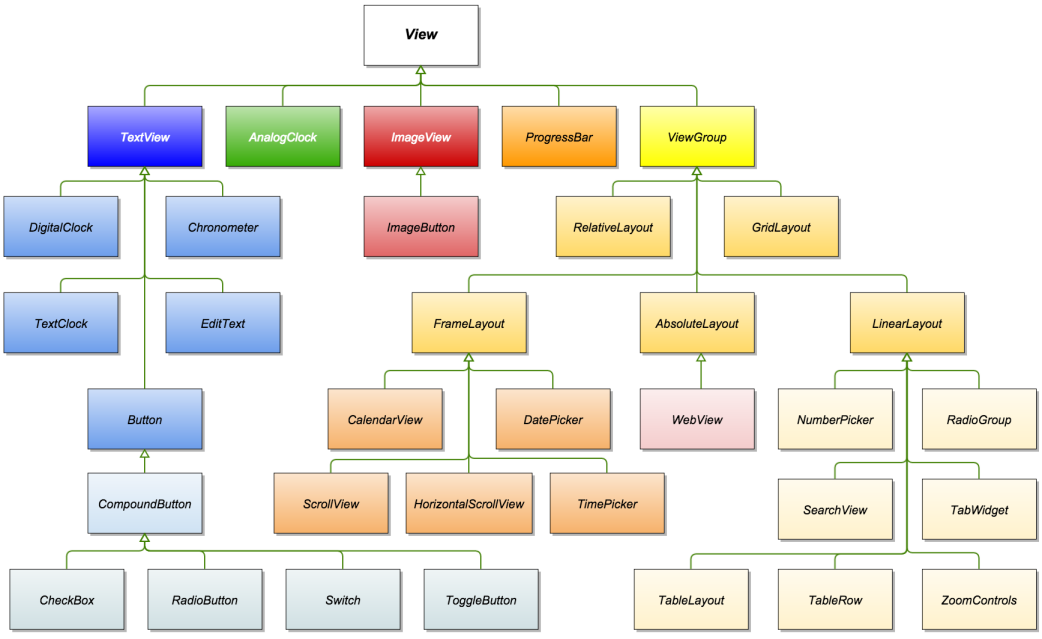

Figure 1: View inheritance hierarchy

Our layout XML document will have its own hierarchy of views, formed with an XML tree. The root is some container (view group) and it has some elements which can either be nested containers or drawable UI components.

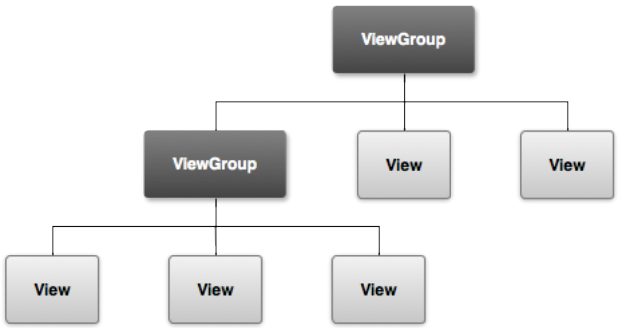

Figure 2: View hierarchy

3 Event driven programming

Unlike the programs we are more used to, there is no main function in Android. It will define what screen (activity) to render when we start the app and from there on, it will react to events, such as swipes and clicks. When such an event occurs, an appropriate event listener is set to handle it. By default, nothing is done but we can implement a method to run if an event occurs.

Some events can be implemented in XML. Below we define a button an `onClick` listener is set. For this to work, we need to define the method `clickMethod` in the corresponding activity.

```
<Button
    android:id="@+id/my_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="ClickMe"
    android:onClick="clickMethod"/>
```

We can also set listeners in Java by finding the view by its id. The following shows a lambda function being used to set the onClick listener.

```
findViewById(R.id.my_btn).setOnClickListener(v -> {
    // is called when clicked
});
```

4 Layouts

Layouts are invisible view groups that allows us to structure our UI. The root element in our layout XML has to be any of the available layouts. There are many layouts available in Android and we will only go over a few but you should nonetheless explore as many of them as you can. Before we look at the various layouts, we need to address certain XML attributes and units for their values. There are six different units for sizes in Android, seen in table 1, but luckily we will only ever have to use two, sp for fonts and dp for everything else.

Abbreviation	Name	Description
dp	Density Independent Pixel	1 pixel in 160 dpi screens (px = dp · (dpi /160))
in	Inches	Physical measurement
mm	Millimeters	Physical measurement
pt	Point	Physical measurement
px	Pixel	A single screen pixel
sp	Scale Independent Pixel	Scaled based on user's font size preference.

Table 1: Size units

The attributes to position elements are explained in figures 3, 4 and 5.

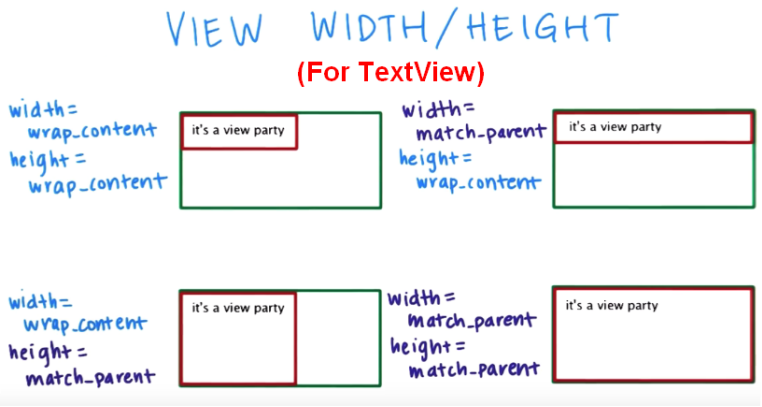

Figure 3: Width and height

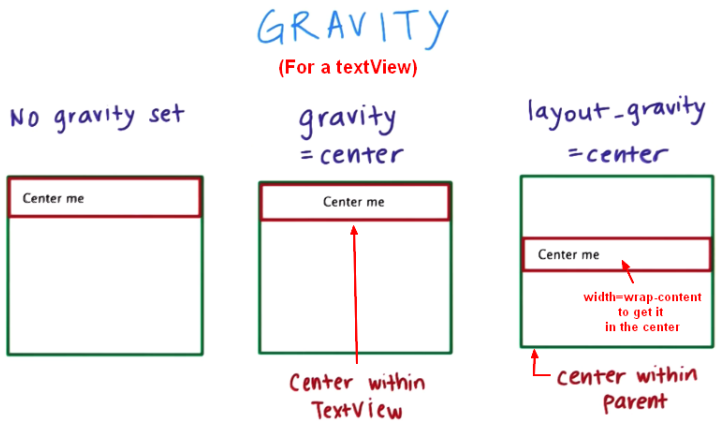

Figure 4: Gravity

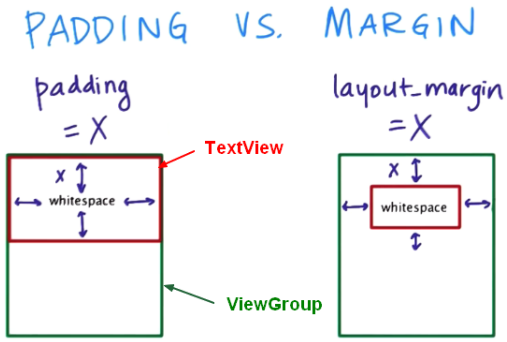

Figure 5: Padding and margin

4.1 Linear layout

Linear layout places other views either horizontally in a single row or vertically in a single column. The `orientation` attribute determines its direction. Below we have a linear layout in XML where the orientation is horizontal, so all children will be placed in a single row and they will not wrap if we reach the end of the screen. The gravity centers horizontally and vertically for both orientations. The views are placeholder for any view.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
```

```
    android:paddingRight="16dp"
    android:orientation="horizontal"
    android:gravity="center">
    <View> </View>
    <View> </View>
</LinearLayout>
```

4.2 Relative layout

Relative layouts specify the position of its views relative either it siblings or parent view. Attributes revolve around position relative to others given by id (or their unique parent view). Attributes like `below` and `toLeftOf` have view id's as value while alignments relative to parent have boolean values. Relative layout have largely been replaced by Constraint layout.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <SomeView
        android:id="@+id/id1"
        android:layout_alignParentTop="true"
        android:layout_marginStart="79dp"
        android:layout_marginTop="111dp" />
    <SomeView
        android:id="@+id/id2"
        android:layout_marginStart="44dp"
        android:layout_marginTop="45dp"
        android:layout_below="@+id/id1"
        android:layout_toEndOf="@+id/id1" />
</RelativeLayout>
```

4.3 Grid layout

The grid layout places its children with 2d positions in a grid. The column determines where the child lands horizontally and row vertically.

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <SomeView
        android:layout_column="0"
        android:layout_row="0"/>
    <SomeView
        android:layout_column="1"
        android:layout_row="1"/>
</GridLayout>
```

4.4 Lists view

List views offer scrollable views of list items. Static lists can be defined with string arrays in the string resource XML file and added with the entries tag as is shown below.

```
<!-- in values/strings -->
<string-array name="arr">
    <item>"item1"</item>
    <item>"item2"</item>
    <item>"item3"</item>
</string-array>

<!-- in Layout -->
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:entries="@array/arr"
    android:id="@+id/list_view_example">
</ListView>
```

Dynamic lists must be populated in Java with the `ArrayAdapter` class. Clicking an item in the list does not invoke a `OnClickEvent` on the item but `OnItemClickListener`. A source code with such an example can be found here or in this video.

5 Widgets

Widgets are the drawable UI components of the android views. We have already seen buttons when we looked at events but there are plenty others available. We will only look at a handful but you are encouraged to look at more. Note that in these examples, we are not storing strings in the resource string file for demonstration purposes but emphasize that they should always be placed there.

5.1 Text view

A text view is used to display text. The text can be changed in Java with the `setText` method and retrieved with the `getText` method. Note that `getText` returns a `CharSequence` rather than a string.

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="sans-serif-condensed"
    android:text="My Text View"
    android:textSize="30sp"/>
```

5.2 Edit text

Edit text are used for text input. They have various types that restrict their input, such as numeric, password, email and text.

```
<EditText
    android:id="@+id/my_input_form"
    android:layout_margin="15dp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter text"
    android:inputType="text" />
```

Listing 1: Edit text declaration

To access the input, we can use the following code. Watch out for empty inputs or inputs containing white spaces only.

```
EditText input = (EditText)findViewById(R.id.my_input_form);
String currentInput = input.getText().toString();
```

5.3 Image view

Image view can display image resources on screen. Suppose we have an image called image.png that we have placed in the drawable folder, then we can set an image view's src to its name and it will render it on screen.

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="250dp"
    android:background="@color/colorAccent"
    android:src="@drawable/image"/>
```

5.4 Radio buttons

Radio buttons have to be grouped in a container called `RadioGroup`. If not, choosing one will not cancel another. Toggling a radio button triggers an on click event and all buttons can be set to the same listener. An example can be found here (source code) or here (video).

5.5 Spinners

Spinners are Android's drop down lists. They work similarly to list views and can be populated with string array resources in XML. They can also be populated in Java with `ArrayAdapter` but use a different layout id than list views. The `onItemSelectedListener` is triggered as soon as an element is selected in the spinner. For spinners with a possible empty choice, the method `onNothingSelected` is invoked. An example can be found here (source code) or here (video).

6 Assignment - Simple calculator

In this assignment you should design a very simple calculator with two input fields and something to display the result (or error message). It should support addition, subtraction, multiplication and division.

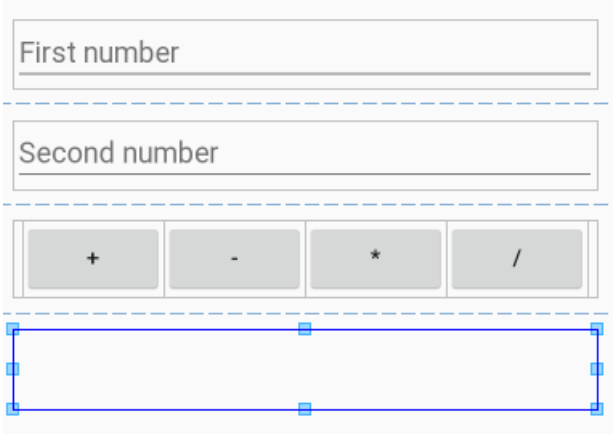

Figure 6: Simple calculator