LE/EECS 4443: Mobile User Interfaces (LAB)

Week 4: Advanced Layouts & UI Building Blocks

Shogo Toyonaga[1]

[1]Lassonde School of Engineering
York University

January 17, 2025

Table of Contents

Introduction

By the end of this tutorial, you will be able to...

1. Select the most appropriate layouts for your activity
2. Use and adapt the supported Android Widgets
3. Design your applications for performance
4. Design your applications for UI/UX

Introduction

- **Layouts** define a set of widgets and their positions on the screen.
- Located under app/res/layout/*.xml
- **Note:** You can technically instantiate a layout through Java code at run-time, however, we heavily discourage this because:
  1. XML Layouts offer a better separation of concern between the, "look" of an application and its functionality
  2. Easier to debug and troubleshoot
  3. Offers reusability across multiple applications with little to no modifications

Layout Example

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:orientation="vertical"
9      android:gravity="center"
10     tools:context=".MainActivity">
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="Hello World!"
16         />
17 </LinearLayout>
```

Layout Explanation

- As you can see, the XML consists of **elements** and **attributes**.
- **Elements** are represented by tags that are opened and closed.

  e.g., $< TextView .../ >$
- **Attributes** are represented by the internal properties of elements.

  e.g., android:id, android:layout_width, android:layout_height

Common Attributes & Explanations

- android:id $\rightarrow$ Used to generate a resource ID in the R class. We use an ID to connect a widget in the Controller with findViewById($\cdots$)
- android:layout_width $\vee$ android:layout_height $\rightarrow$ Used to specify the dimensions of a View.
  1. match_parent makes the View as big as its parent.
  2. wrap_content makes the View as big as its inner contents require.
- android:orientation $\rightarrow$ Determines whether the layouts children will appear vertically or horizontally. The order in which children are defined determines the order in which they appear on the screen.

Common Attributes & Explanations

android:gravity → Specifies how to place children inside of a layout.

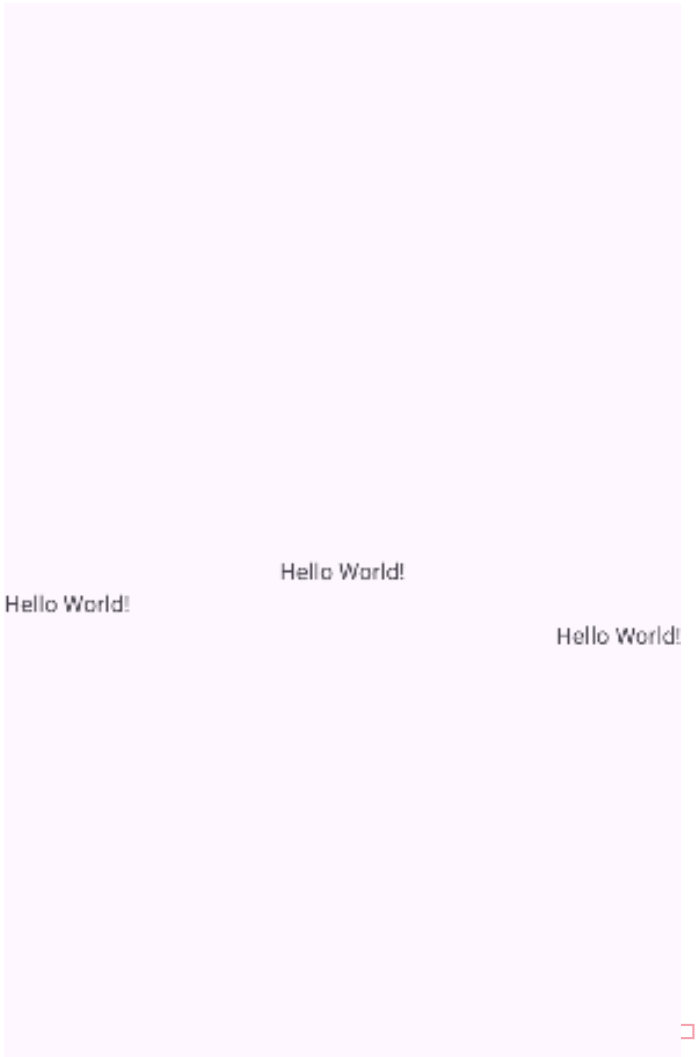
e.g., left, right, top, bottom, center

android:layout_gravity → Specifies how to place the View itself in the context of the parent.

Putting It All Together

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:id="@+id/main"
6      android:layout_width="match_parent"
7      android:layout_height="match_parent"
8      android:orientation="vertical"
9      android:gravity="center"
10     tools:context=".MainActivity">
11     <TextView
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:text="Hello World!"
15         />
16     <TextView
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"
19         android:text="Hello World!"
20         android:layout_gravity="start"
21         />
22     <TextView
23         android:layout_width="wrap_content"
24         android:layout_height="wrap_content"
25         android:text="Hello World!"
26         android:layout_gravity="end"
27         />
```

Putting It All Together



Types of Layouts

1. **Static**
   - LinearLayout
   - RelativeLayout
2. **Dynamic**
   - ListView
   - GridView
   - RecyclerView

https://developer.android.com/develop/ui/views/layout/declaring-layout

LinearLayout

- Used when you want widgets arranged in a single column or row.
- Supports android:layout_weight which assigns "importance" to individual children. Higher weights take up more available space.
- The default weight for children is 0; it fills the provided space as necessary.

https://developer.android.com/develop/ui/views/layout/linearWeight

Try It!

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:gravity="center"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Apples"
        android:padding="5dp"
        android:background="@color/material_dynamic_neutral40"
        android:textColor="@color/white"
        android:layout_weight="1"
        />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Oranges"
        android:padding="5dp"
        android:background="@color/material_dynamic_neutral40"
        android:textColor="@color/white"
        android:layout_weight="3"
        />
```

RelativeLayout

- Used when you want to arrange widgets in relative positions with relation to parents and other siblings.
- Keeping the user interface, "flat" rather than relying on nested ViewGroups improves performance.
- "By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties."

https://developer.android.com/develop/ui/views/layout/relative

Introduction

- Dynamic layouts are useful for handling large amounts of data via an Adapter
- The Adapter "binds" data to the layout and facilitates communication between the database and View. Following this, it converts each entry into a View that is inserted into the AdapterView.
- We like to use dynamic views (e.g., RecyclerView) as they improve responsiveness and power consumption compared to static views.

Case Study: Demo_ListView_1

1. Demo_ListView_1 Documentation
   - **Objective:** Populate a ListView from a String Array source.
   - Controller extends ListActivity to support the required functions.
   - The Array of words is referenced from app/res/words.xml which hosts a string-array.
   - A custom Adapter is instantiated and attached to the ListView to populate it at run-time.

Case Study: Demo_ListView_2

1 Demo_ListView_2 Documentation

- **Objective:** Populates a ListView with images stored on a device's SD card.
- Displaying images brings special challenges, since the image files must be accessed and converted to bit maps for display in the view objects that appear in the ListView.
- In the main activity we create a String array of image filenames (rather than a String array of words, as in Demo_ListView_1).
- Since this program is accessing the device's internal memory card, the manifest must include the following permission (placed just before the application element): "android.permission.READ_EXTERNAL_STORAGE"

GridView

- **Definition:** A view that shows items in a two-dimensional scrolling grid.
- Lets implement it using a RecyclerView!

RecyclerView: Components

1. **RecyclerView**: The ViewGroup in your main layout that contains the corresponding views & data sources.
2. **RecyclerView.ViewHolder**: Each "child" view in the list. It doesn't have any data associated with it until it is binded to the data source.
3. **RecyclerView.Adapter**: Used to facilitate the communication between the RecyclerView and data source to generate the children seamlessly.
4. **LayoutManager**: Arranges the individual elements in your RecyclerView.
   - LinearLayoutManager $\longrightarrow$ Arranges Viewholders in a one-dimensional list
   - GridLayoutManager $\longrightarrow$ Arranges ViewHolders in a two-dimensional grid

Implement the Adapter & ViewHolder

1. Implement ViewHolder class which contains the layout for an individual item in the list.
2. Implement the Adapter which creates ViewHolder objects as needed and sets the data for those views. You will need to override:
   - **onCreateViewHolder():** Used to create a new ViewHolder at run-time. Do note that the data is not set (or, "binded") in this method.
   - **onBindViewHolder():** Sets the data in the ViewHolder(s).
   - **getItemCount():** Gets the size of the data source. RecyclerView uses this to determine when there are no more items to be displayed!

Dynamic Layouts - Case Studies

1. **DemoReminders**: A simple TODO application that explores new event listeners. It uses a ListView.
2. **DemoGridView**: A friends list application that uses a GridView to display profile avatars, names, and their current status (like Discord).Use this for learning how to use RecyclerViews.
3. **DemoPokedex**: Uses a SearchView, ListView, and ArrayAdapter to simulate a simplified query-able database.
4. **DemoRecyclerViewLoL:** Uses a RecyclerView to implement a GridView which pulls information from drawable and the Champions database.

Introduction

- You will have to pick a widget for a task where there there can be many competing alternatives.
  1. Spinners, Checkboxes, or Radio Buttons?
  2. Progress or Activity Indicators?
- You should be able to justify your choices above all else.
- Pro Tip: Pick the best widgets that minimize user error and footprint.

Introduction (Case Study)

1. **Spinners** are optimal if every applicable option does not need to be shown at once. They are also good when the UI space is scarce.
2. **Checkboxes** are optimal if a user is expected to select multiple options among a set of items. You should have a decent amount of UI space if you choose this option.
3. **Radiobuttons** are optimal if a user is expected to select a single option among multiple items. There should be a decent amount of UI space if you choose this option.

So many choices!

1. **Fixed Tabs** vs **Scrollable Tabs**?
2. **ListView** vs **GridView**?
3. **Indicator Scrolling** vs **Index Scrolling**?
4. **Text-Only Buttons** vs **Image Buttons**?
5. **Alerts** vs **Popups**?

UI Components & Attributes

- Just like layouts, each widget has its own set of **attributes** which allow you to change its appearance and behaviour(s).
- Choose the correct and most intuitive attributes for a use-case. It is paramount that the purpose of the widget is easily understandable to a novice or expert user alike.

UI Components & Attributes

Remark

"It's time to calculate your risks, make amends to reduce errors, and increase your successes."
- Martin Powell

Conclusion

Remark

Thank you for your attention!
Questions?