LE/EECS 4443: Mobile User Interfaces (LAB)

Week 4: Layouts & User Interface (Building Blocks)

S.Toyonaga[1]

[1]Lassonde School of Engineering
York University

February 2, 2024

Table of Contents

Introduction

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

1. Understand Layouts
2. Determine how to select the best layout for your use-case
3. Determine how to select the best UI components for your use-case

Layouts: Definition

Remark

A layout defines the **structure** for a user interface in your app, such as in an activity. All elements in the layout are built using a **hierarchy** of View and ViewGroup objects.

- A View usually draws something the user can see and interact with.
- A ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.

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

Layouts: View & ViewGroup

- **Views** (Buttons, TextViews, Sliders, ...)
- **ViewGroups** (LinearLayout, RelativeLayout, ConstraintLayout,...)

Layouts: Instantiation

1. **XML (Layout)**
   - **Location:** *app* $\rightarrow$ *res* $\rightarrow$ *layout* $\rightarrow \cdots$
   - Good design, supporting **separation of concerns** and MVC paradigm
2. **Activity (Java)**
   - **Location:** *app* $\rightarrow$ *java* $\rightarrow$ *com*$\cdots \rightarrow \alpha$*Activity.java*
   - Instantiates Views at runtime
   - Not recommended as presentation and control become mixed/intertwined

Types of Layouts

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

Static Layouts

1. LinearLayout
   - (+) Supports **layout weights** which dynamically assign available space to views based on priority
   - (+) Very simple design, supporting **vertical** and **horizontal** orientation modes.
   - (-) Nesting layouts leads to very poor performance
2. RelativeLayout
   - (+) Very simple and intuitive syntax
   - (-) Re-Organizing the Layout after an initial concept is very time-consuming
   - (-) Verbose Code in XML File

LinearLayout: Weights

Remark

LinearLayout also supports assigning a weight to individual children with the *android:layout_weight* attribute. This attribute assigns an **"importance"** value to a view in terms of how much space it occupies on the screen. A larger weight value lets it expand to fill the **remaining space** in the parent view. Child views can specify a weight value, and any remaining space in the view group is assigned to children proportionately, based on their declared weight. **The default weight is zero.**

Source: `https://developer.android.com/develop/ui/views/layout/linear`

LinearLayout: Weights (Example)

Example

- If there are three text fields and two of them declare a weight of 1, while the other is given no weight, then:
  - The third text field without weight doesn't grow. Instead, this third text field occupies only the area required by its content.
  - The other two text fields expand **equally** to fill the space **remaining** after all three fields are measured.

Source: https://stackoverflow.com/questions/39508909/understanding-androidlayout-weight

Static vs Dynamic Layouts

- Dynamic views are great for handling large amounts of data
- Improves **responsiveness** and **power consumption** compared to static views in many cases
- As you scroll, views are **recycled** and new information is loaded into the layout.
- **Note:** The data is loaded into the Dynamic View by leveraging an **Adapter**.

Adapters

- The Adapter acts as the **mediator** between the dynamic view and data source
- You may either use pre-existing adapters (ArrayAdapter, CursorAdapter, SimpleCursorAdapter) or create your own to interface with the Dynamic Layout.
- **Demo:** `https://github.com/stoyonaga/EECS4443_W24_Assets/tree/main/TA%20Demos/ArrayAdapter`
- **Demo:** `https://github.com/stoyonaga/EECS4443_W24_Assets/tree/main/TA%20Demos/GridView`

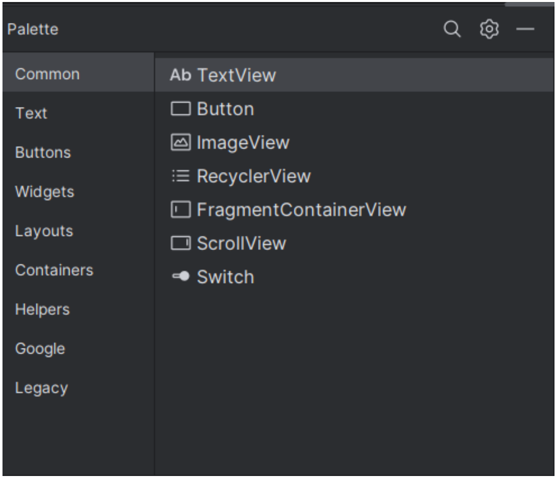
Introduction: UI Components

- There are a variety of components which can be used to achieve similar purposes / use-cases
  1. Spinners vs Checkboxes vs Radio Buttons
  2. Progress vs Activity Indicators
- **Gamer Move:** Pros will choose the best components which **minimize** user error $\wedge$ user footprint for a particular **task**.

UI Components: Optimal Selection

- Spinners are **optimal** if showing every applicable option at once is not necessary and space is very scarce.
- Checkboxes are **optimal** if a user is expected to select multiple options and there is a sufficient amount of space.
- Radiobuttons are **optimal** if a user is expected to select a singular option among a list of values and there is a sufficient amount of space.

Android Studio: Component Picker



Attributes: UI Components

- Each component (View, ViewGroup) have a set of attributes which allow you to finetune their **appearance** and **behaviour(s)**
- Within the layout (XML), they often have an appearance similar to *android:attribute* = *value*
- **Being able to choose correct and intuitive attributes to match a particular use-case is paramount!**

Conclusion

Remark

Thank you for your time!!
Do you have any questions? :)