CMPS 312

Views & Layout

Dr. Abdelkarim Erradi

CSE@QU

Outline

1. Activity

2. Views

3. Constraint Layout
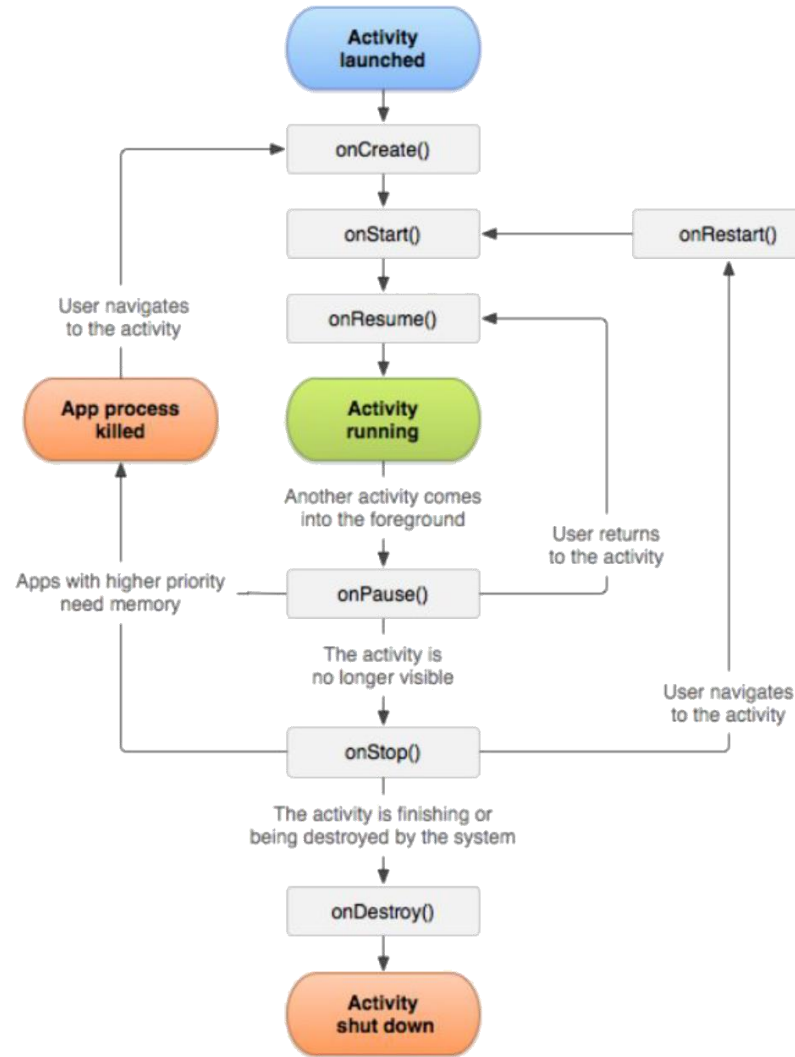
Activity
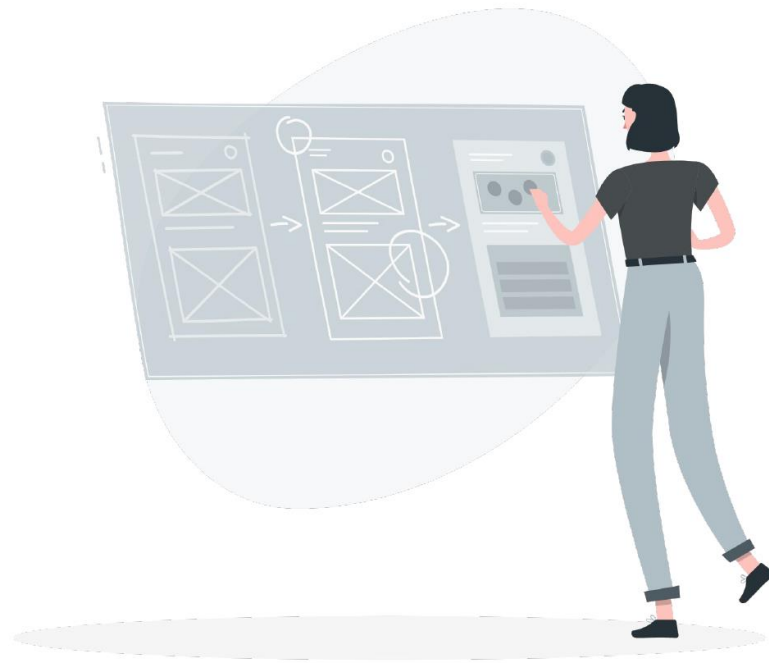
Activity

- **Activity** provides the UI that the user interacts with
    - Allow the user to do something such as order groceries, send email
    - Has layout (.xml) file & Activity class
    - This allows a **clear separation** between the UI and the app logic
- Connecting activity with the layout is done in the **onCreate** method
- Activity class define listeners to handle events:
    - User interaction events such press a button or enters text in a text view
    - External events such as receiving a notification or screen rotation
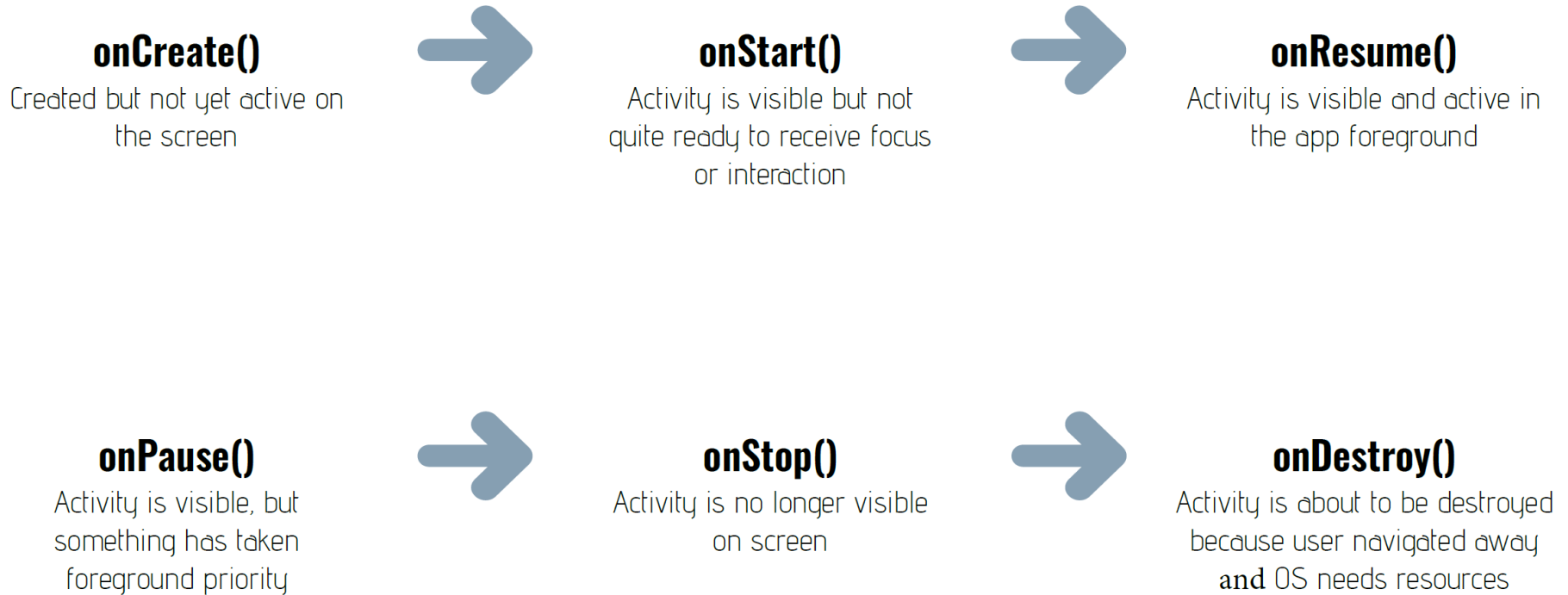- Can start other activities in the same or other apps

Activity Lifecycle

An activity has essentially **four states**:

- *Active* if the activity in the foreground of the screen

- *Paused* if the activity has lost focus but is still visible (e.g., beneath a dialog box). A paused activity is alive but can be killed by the system in case of low memory.

  - When the user returns to the activity, it is **resumed**

- **Stopped** if the activity is completely obscured by another activity. It still retains its state but can be killed by the system when memory is needed.

  - When the user navigates to the activity, it must be **restarted** and restored to its previous state.

- **Destroyed** if an activity is paused or stopped, it maybe killed.

 - When the user navigates to the activity, it must be recreated.

Activity Lifecycle

onCreate()
Created but not yet active on the screen

onStart()
Activity is visible but not quite ready to receive focus or interaction

onResume()
Activity is visible and active in the app foreground

onPause()
Activity is visible, but something has taken foreground priority

onStop()
Activity is no longer visible on screen

onDestroy()
Activity is about to be destroyed because user navigated away and OS needs resources

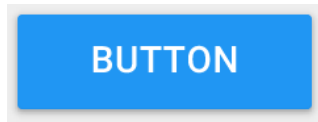- Can run events handlers to runs in response to these events
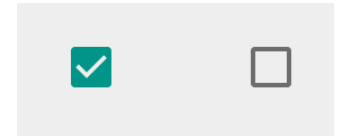
What Makes up an Activity UI?
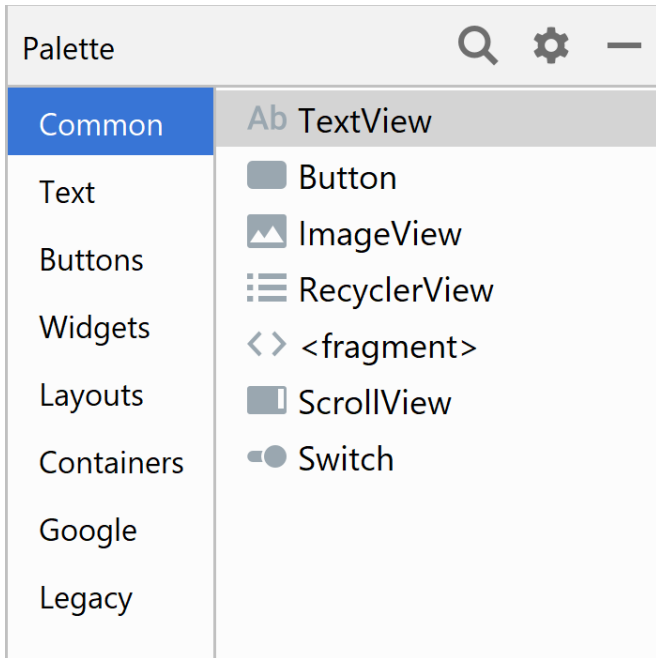
- **Views**
  - Set of pre-built UI components that can be composed to create a GUI
  - e.g. Button, TextView, Menu, List, etc.

- **Layout**
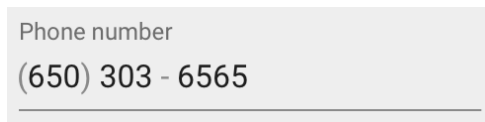 - Container that controls the size and positioning of views in the Activity
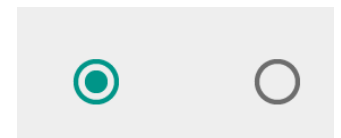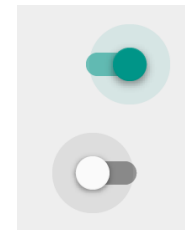
Views

Button

EditText

SeekBar

CheckBox

RadioButton

Switch

Views

- **View = Widget = Control**

  o Examples: Button, Switch, Spinner, TextView, EditText, ImageView

  o Advanced Views (covered later): **RecyclerView** & MapView

- **Common Attributes**

  o id (i.e. **android:id="@+id/myViewId"**)

  o `layout_width`, `layout_height`

    - Values: `match_constraint` (or 0dp), `wrap_content`, fixed size (e.g., 50dp)

Views (Attributes and Listeners)

- TextView - Displays text on the screen
  - `text`

- EditText - Allows entering user input
  - `inputType` : such as email, phone number, etc.
  - `text`
  - `.addTextChangedListener { … }`

- Button - Clickable view responding to user clicks
  - `text`
  - `.setOnClickListener { … }`

- ImageView - Displays image from a URL or from a resource file
  - `.setImageDrawable(drawable)` `// set image to display`
  - `.setOnClickListener { … }`

Customize TextEdit with inputType

- **textPersonName**: Single line of text

- **textCapCharacters**: Set to all capital letters

- **textPassword**: Conceal an entered password

- **number**: Restrict text entry to numbers

- **textEmailAddress**: Show keyboard with @ conveniently located

- **phone**: Show a numeric phone keypad
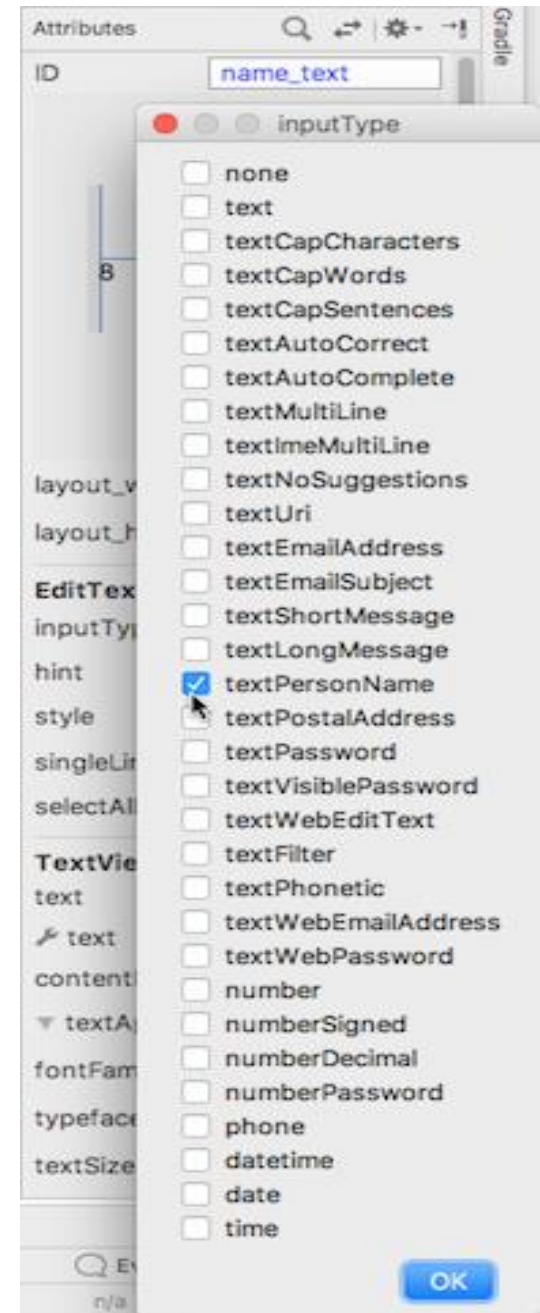
Views (Attributes and Listeners)

- **Switch (on/off)**

  - `.checked = booleanVal` – set check state

  - `.setOnCheckedChangeListener { … }`

- **Spinner (dropdown list)**

  - `.setAdapter(ArrayAdapter)` – specify list values

  - `.setSelection(int)` – specify selected item

  - `onItemSelectedListener { … }`

- **SearchView**

  - `queryHint` –text to display when the field is empty

 - `.setOnQueryTextListener { … }`

Setting Entries of a Spinner in the XML Layout File
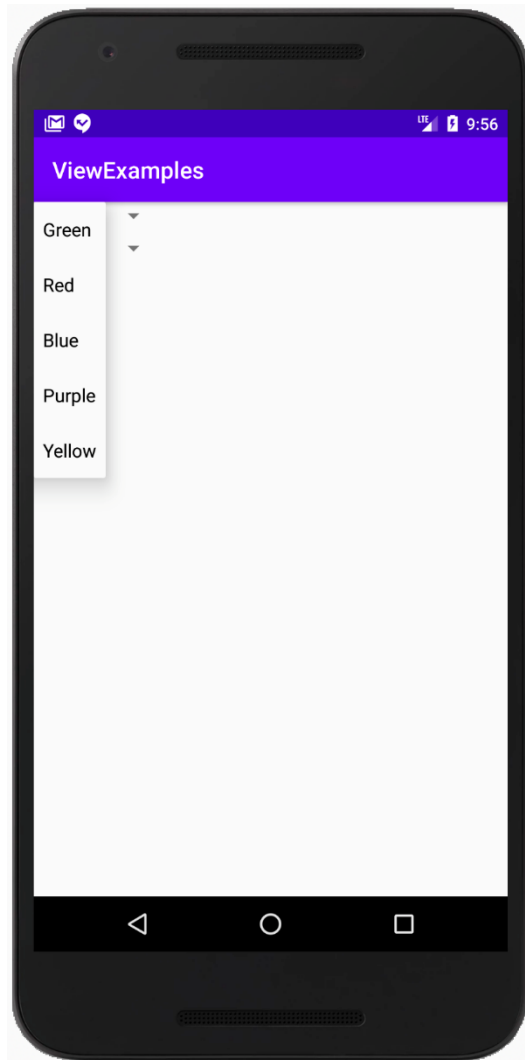
```xml
<Spinner
    android:id="@+id/colorSelector1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="32dp"
    android:layout_marginBottom="4dp"
    android:entries="@array/colorChoices"/>
```

strings.xml ×

```xml
1  <resources>
2      <string name="app_name">ViewExamples</string>
3
4      <string-array name="colorChoices">
5          <item>Green</item>
6          <item>Red</item>
7          <item>Blue</item>
8          <item>Purple</item>
9          <item>Yellow</item>
10     </string-array>
11
12  </resources>
```

Setting Entries of a Spinner in Code
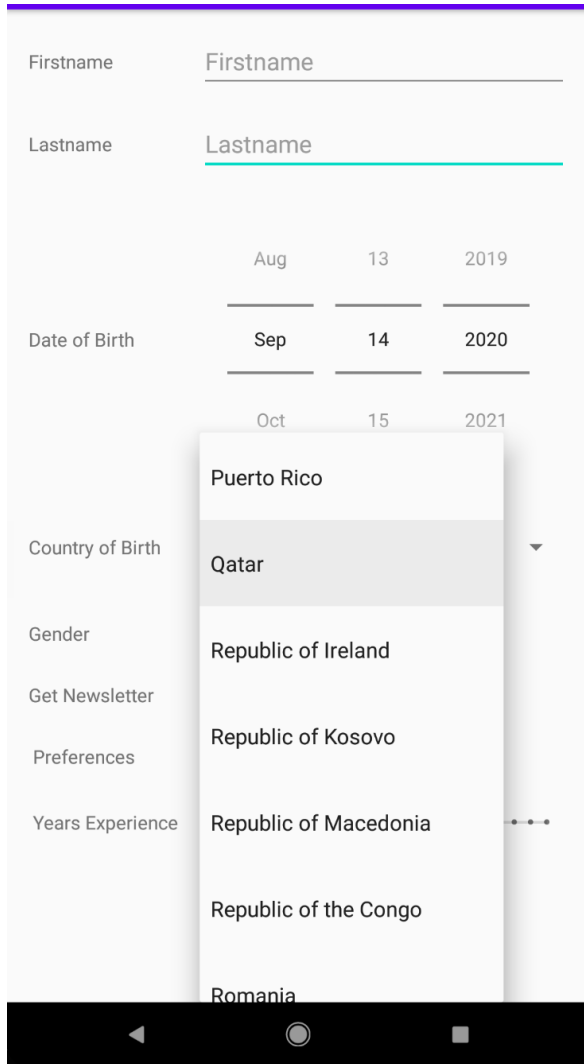

```xml
<Spinner
    android:id="@+id/countriesSp"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
/>
```

```kotlin
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_register)

    CountryRepository.loadCountries(this)

    val adapter = ArrayAdapter<String>(
        this,
        android.R.layout.simple_dropdown_item_1line,
        CountryRepository.countryNames
    )
    countriesSp.adapter = adapter
}
```

Which View gets focus next?

- Topmost view on the activity layout
- After user submits input, focus moves to nearest neighbor—priority is left to right, top to bottom
- Arrange input controls in a layout from left to right and top to bottom in the order you want focus assigned
- Specify ordering in XML

```
android:id="@+id/top"
android:focusable="true"
android:nextFocusDown="@+id/bottom"
```
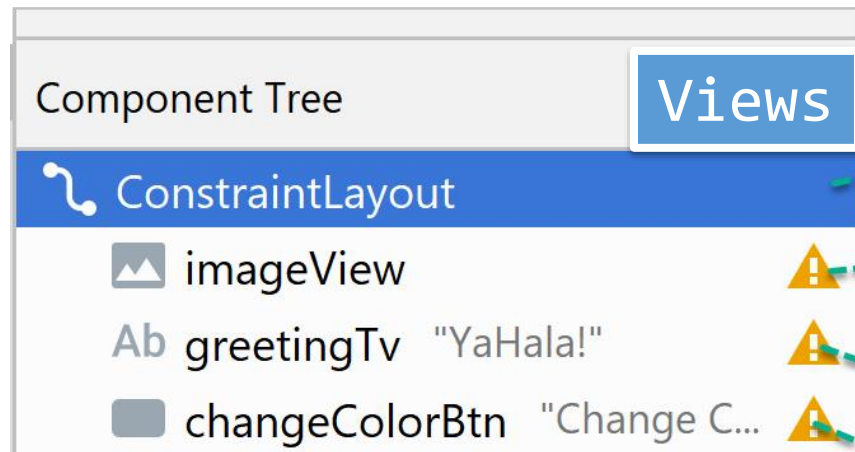
Set focus explicitly

Use methods of the View class to set focus

- setFocusable() sets whether a view can have focus

- requestFocus() gives focus to a specific view

- setOnFocusChangeListener() sets listener for when view gains or loses focus

- Find the view with focus

    o Activity.getCurrentFocus()

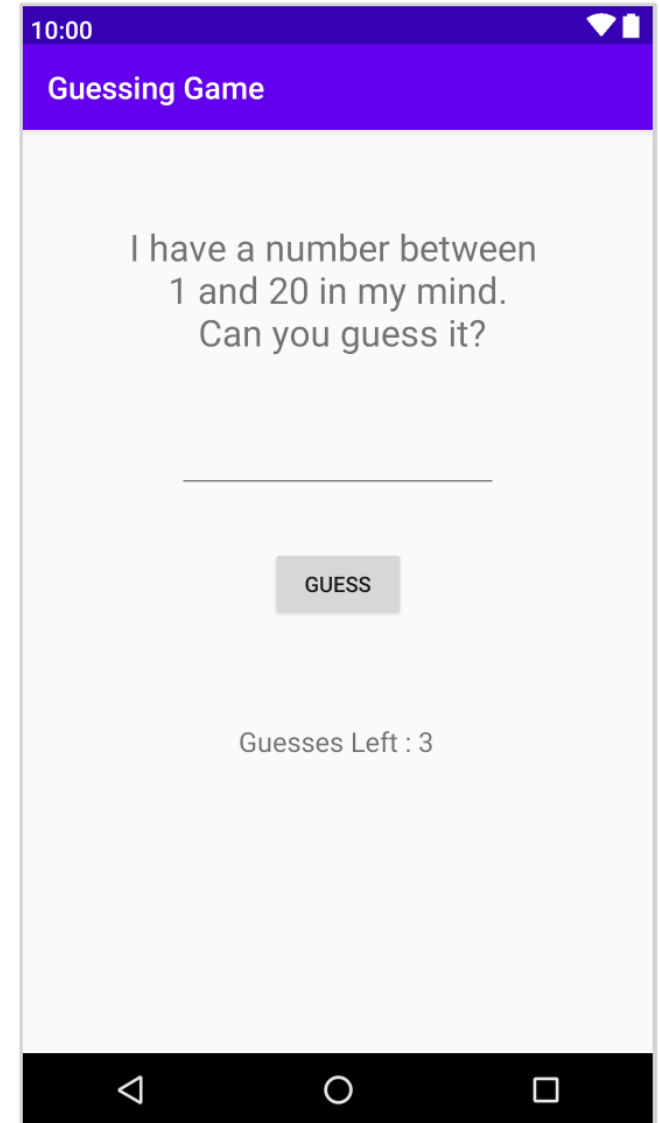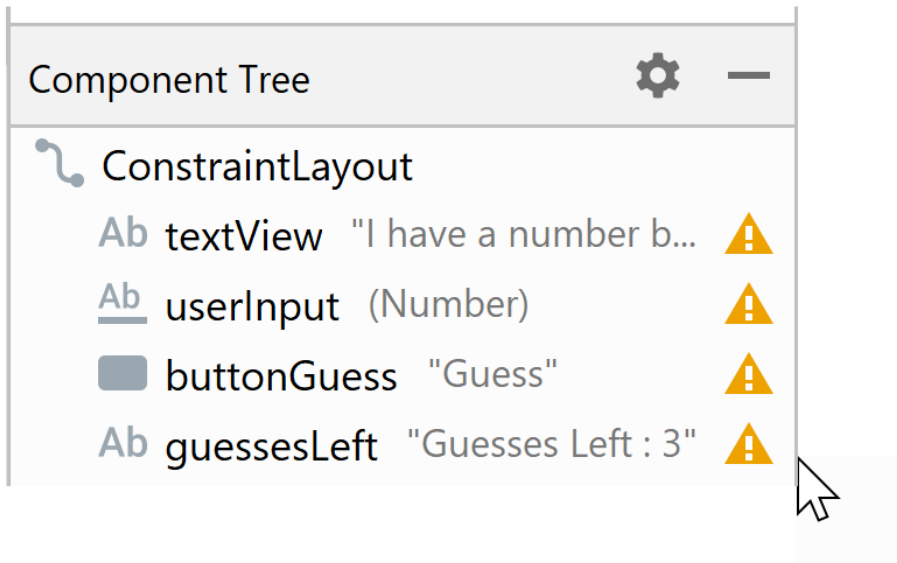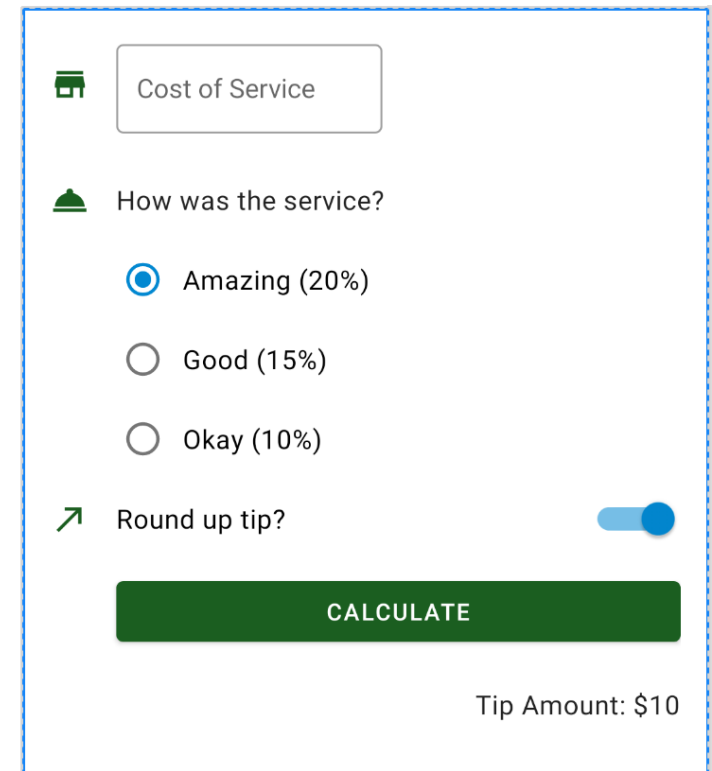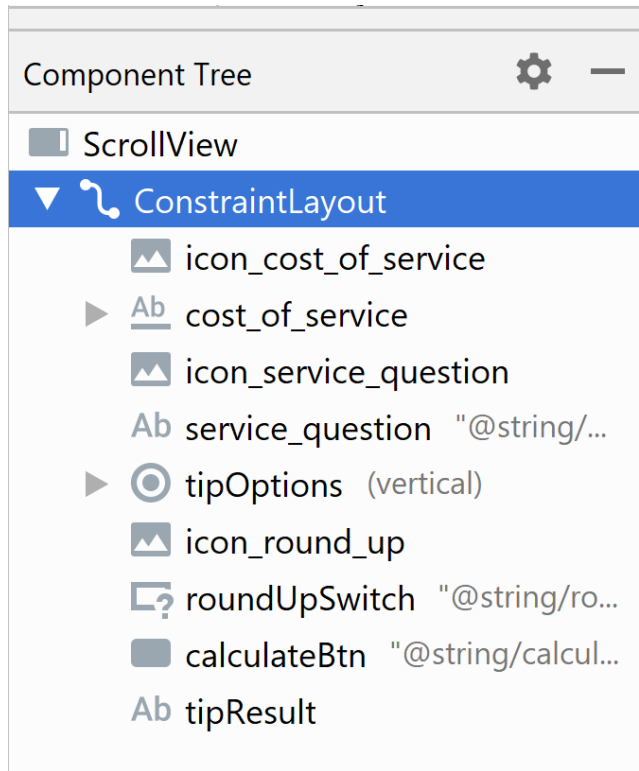   o ViewGroup.getFocusedChild()

App 1 - Color Changer

App that contains Text reading "YaHala!", an Image and a Button that randomly changes text's color with every click

Component Tree | Views

- ConstraintLayout
  - imageView ⚠
  - Ab greetingTv "YaHala!" ⚠
  - changeColorBtn "Change C..." ⚠

App 2 – Guessing Game

Component Tree ⚙ —

- ⌐ ConstraintLayout
  - **Ab** textView "I have a number b... ⚠
  - **Ab** userInput (Number) ⚠
  - ▭ buttonGuess "Guess" ⚠
  - **Ab** guessesLeft "Guesses Left : 3" ⚠

10:00 ▼▮

Guessing Game

I have a number between
1 and 20 in my mind.
Can you guess it?

_____

GUESS

Guesses Left : 3

App 3 – Tips Calculator

Component Tree

- ScrollView
  - ▼ ConstraintLayout
    - icon_cost_of_service
    - ▶ Ab cost_of_service
    - icon_service_question
    - Ab service_question "@string/...
    - ▶ tipOptions (vertical)
    - icon_round_up
    - roundUpSwitch "@string/ro...
    - calculateBtn "@string/calcul...
    - Ab tipResult

Cost of Service

How was the service?

- Amazing (20%)
- Good (15%)
- Okay (10%)

Round up tip?

CALCULATE

Tip Amount: $10

Registration Form

Material Design Components

- Using MDC to make your app look great easily

https://material.io/components

- o Float labels – TextInputLayout

- o FloatingActionButton

- o NavigationDrawer

- o Toolbar

- o CardView

- o TabLayout

- o BottomNavigationView

- o BottomSheet

- o Snackbar

Will be used Later

Constraint Layout

Back
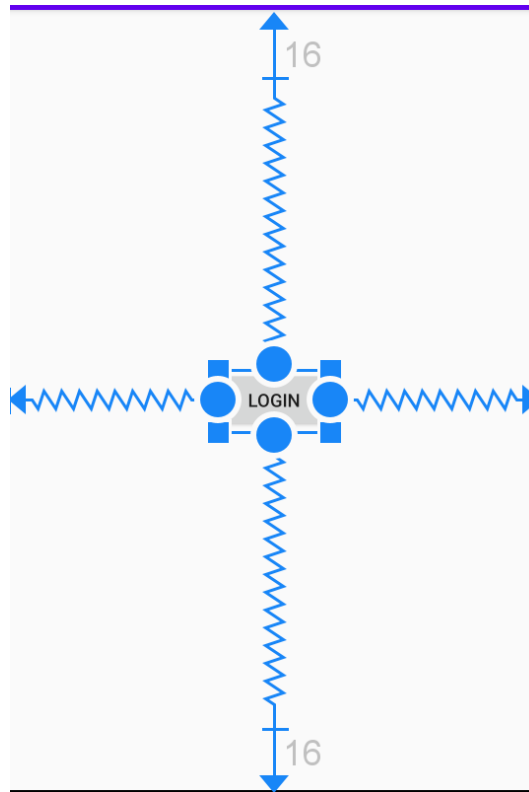
Layouts

- Layout automatically **controls** the **size** and **placement** of views to create a **Responsive UI**

  - Frees programmer from handling/hardcoding the sizing and positioning of UI elements

  - **Responsive UI =** When the screen is resized, the views reorganize themselves based on the rules of the layout

Constraint Layout

- ConstraintLayout: Allows buliding a Responsive UI by defining constraints for views

  o A constraint is a **connection** to another view, parent layout, or invisible Guideline / Barrier

  o Constraints control the **position** and **alignment** of UI elements

    - Position a view relative others including the parent

    - Need to add at least one horizontal and one vertical constraint
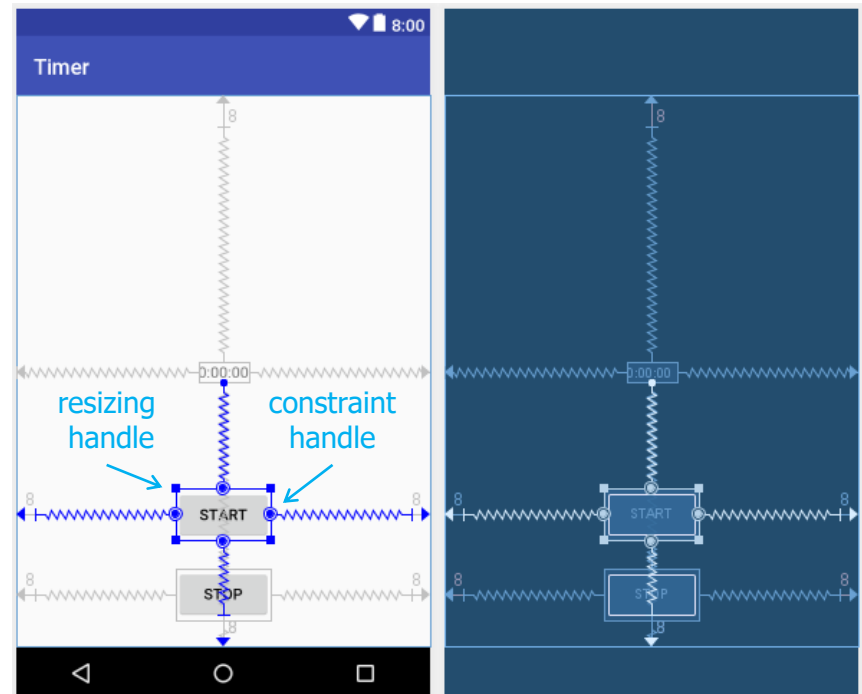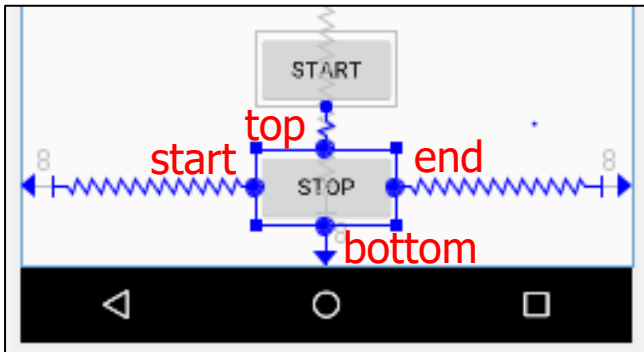
    - Center views by adding constraints to opposite sides of the view

Defining Constraints

Steps
1. Drop a view to the editor
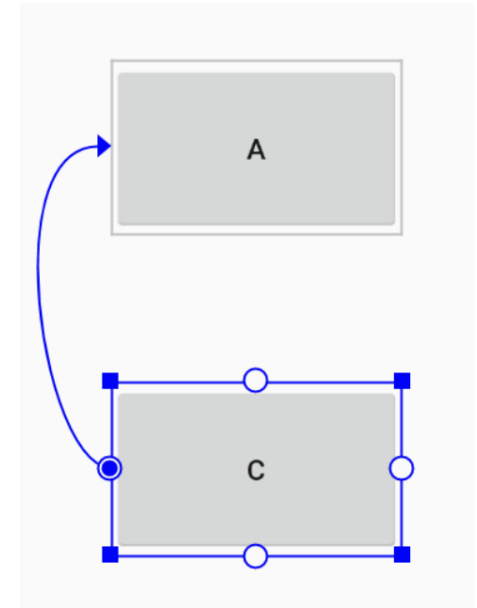2. Connect constraint handles
   (e.g., top/bottom/left/right)
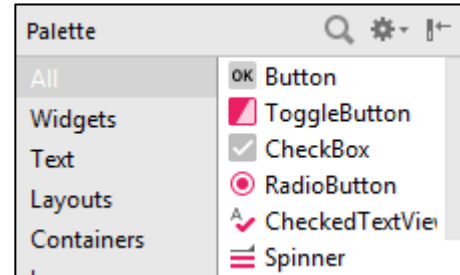
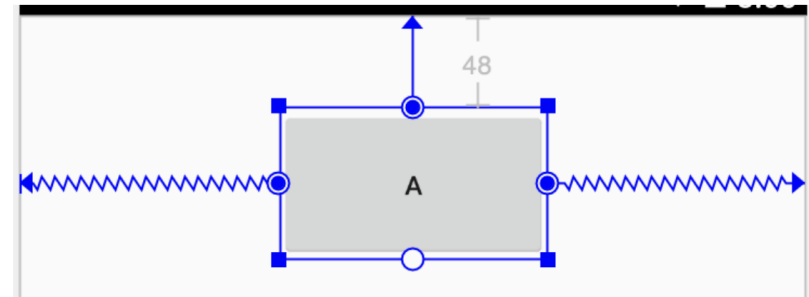Add at least one horizontal
and one vertical constraint

Alignment

- Align the edge of a view to the same edge of another view.

- The left side of C is aligned to the left side of A.

  o If you want to **center** view C, create a constraint on both sides

Bias

- If you add opposing constraints on a view, the constraint lines become like a **spring** to indicate the opposing forces

- The view becomes centered between the two constraints with a bias of 50% by default
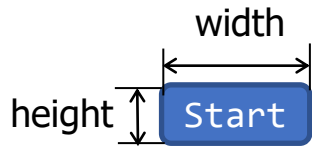
- You can adjust the bias by dragging the view

View Constraints Editor

View Size

layout_width="0dp"

layout_width="wrap_content"

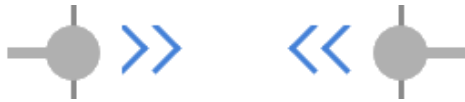**layout_width="200dp"**

- The view expands to **match constraints** on each side (after accounting for the view's margins)
  - View will grow/shrink on resizing

- The view expands as needed to **fit** its contents

- **Fixed** size (e.g., 200dp density-independent pixels)

Guideline

- A guideline is a visual guide used to divide the layout

- Add a vertical or horizontal **guideline** to which you can constrain views, and the guideline will be <u>invisible</u> to app users

- Position the guideline within the layout based on either **dp** (Density-independent pixels) units or percent, relative to the layout's edge
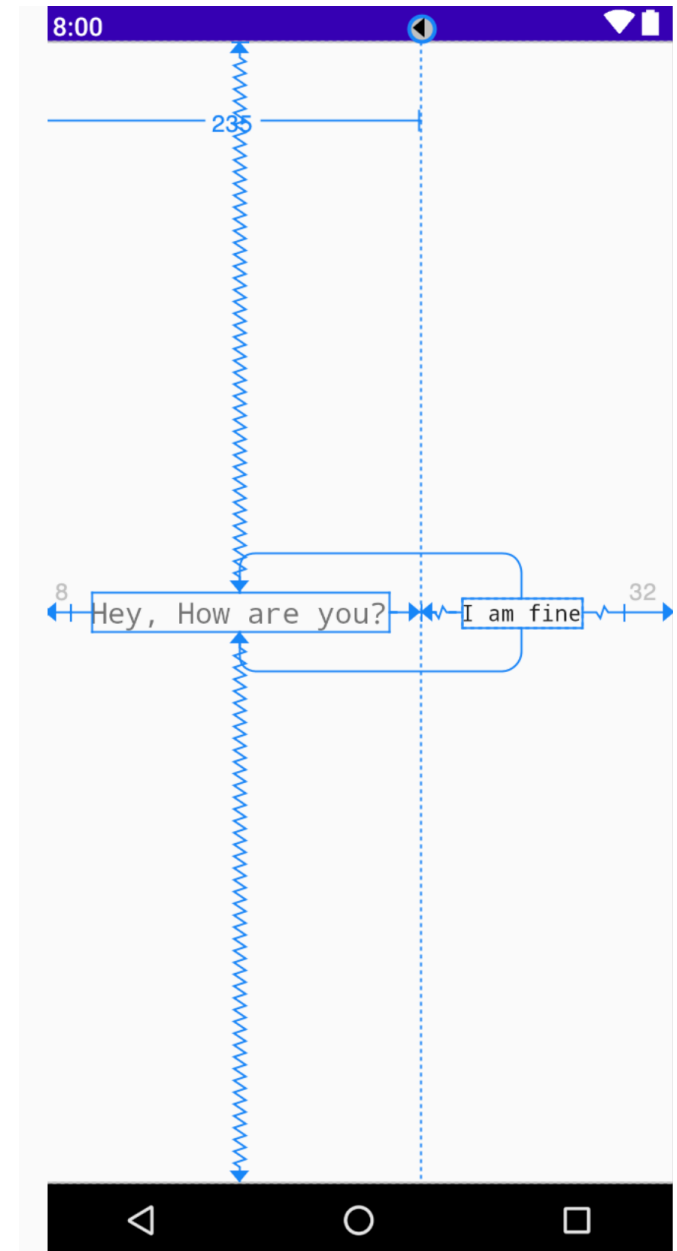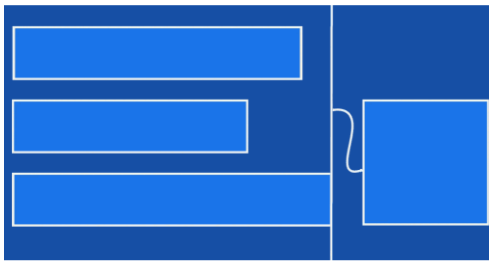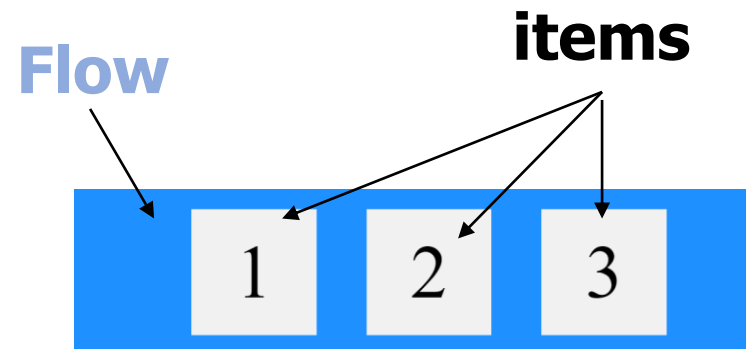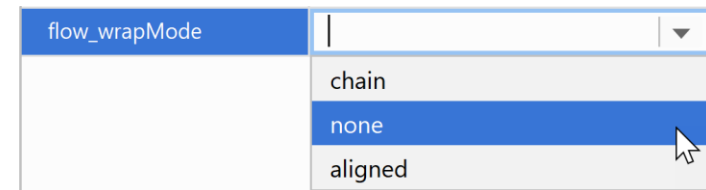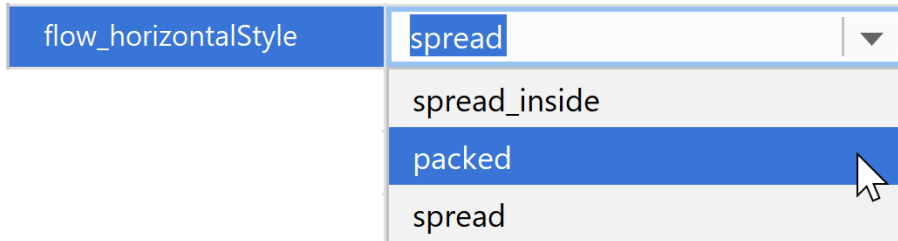
Barrier

- A Barrier is a virtual view, similar to a Guideline, to which we can constrain objects.

- The Barrier width/height are determined by the views placed in it

- You'll want to use a barrier any time the views placed in it **could dynamically vary in size** based on user input or language setting

ABC

ABDCDEFGHIJKLMNOPQRSTUVWXYZ

```
<android.support.constraint.Barrier
    android:id="@+id/barrier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="start"
    app:constraint_referenced_ids="button1,button2" />
```

Flow

- Flow provides an efficient way to **distribute space** among items in the flow while accommodating different screen sizes

orientation	
	horizontal
	vertical

flow_horizontalStyle	spread
	spread_inside
	packed
	spread

flow_wrapMode	
	chain
	none
	aligned

items

Flow

1	2	3
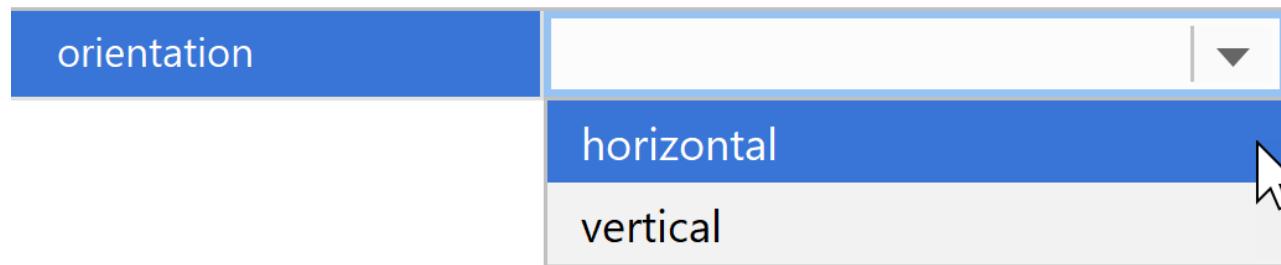
Flow - wrapMode

app:**flow_wrapMode** = "none | chain | aligned"
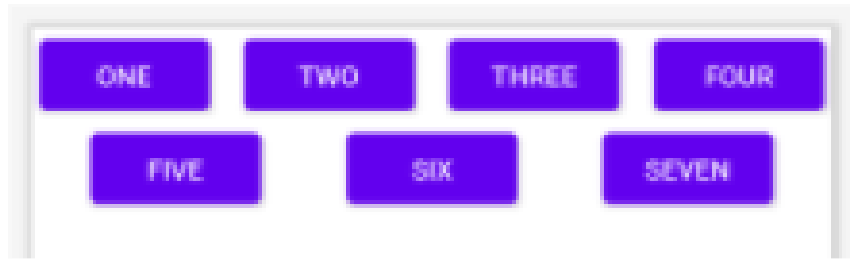
app:flow_wrapMode="chain"

app:flow_wrapMode="aligned"

app:flow_wrapMode="none"

Reusing Layouts

- Extract commonly used elements into common layout and then use `<include>` tag to include a layout

```
<include
    android:id="@+id/toolbar"
    layout="@layout/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Summary

- **Activity** provides the UI that the user interacts with

    - It has layout (.xml) file & Activity class => This allows a **clear separation** between the UI and the app logic

    - Activity class define listeners to handle events

- ConstraintLayout enables responsive design

.. mastering it will take some time and effort 🧗‍♀️ 🏋️ …

Resources

- Build a Responsive UI with ConstraintLayout
  - https://developer.android.com/training/constraint-layout

- ConstraintLayout codelab
  - https://codelabs.developers.google.com/codelabs/constraint-layout/

  - https://developer.android.com/codelabs/kotlin-android-training-constraint-layout

- Android Dev Guide
  - https://developer.android.com/guide/