

CMPS 312

Read Chapter 1 & 6



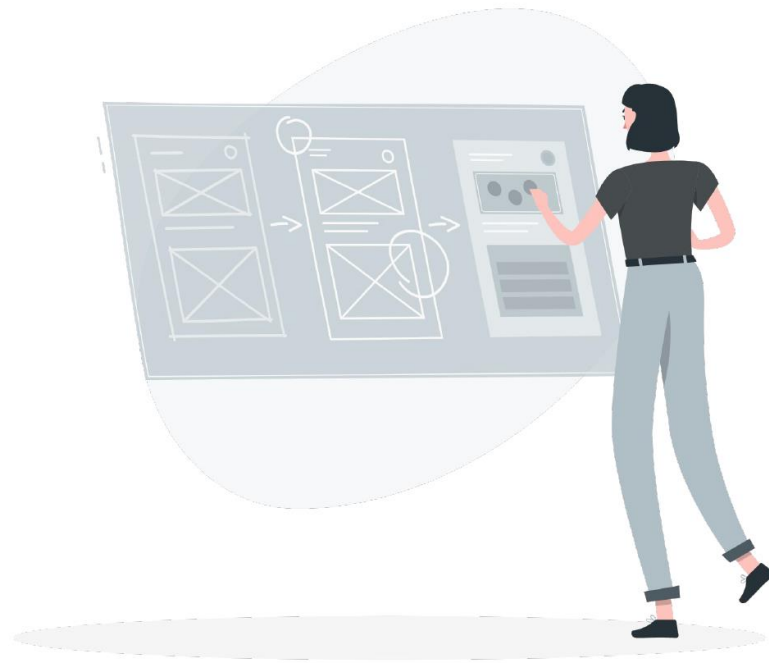
Views & Layout

Dr. Abdelkarim Erradi
CSE@QU

Outline

1. Activity
2. Views
3. Constraint Layout

Activity



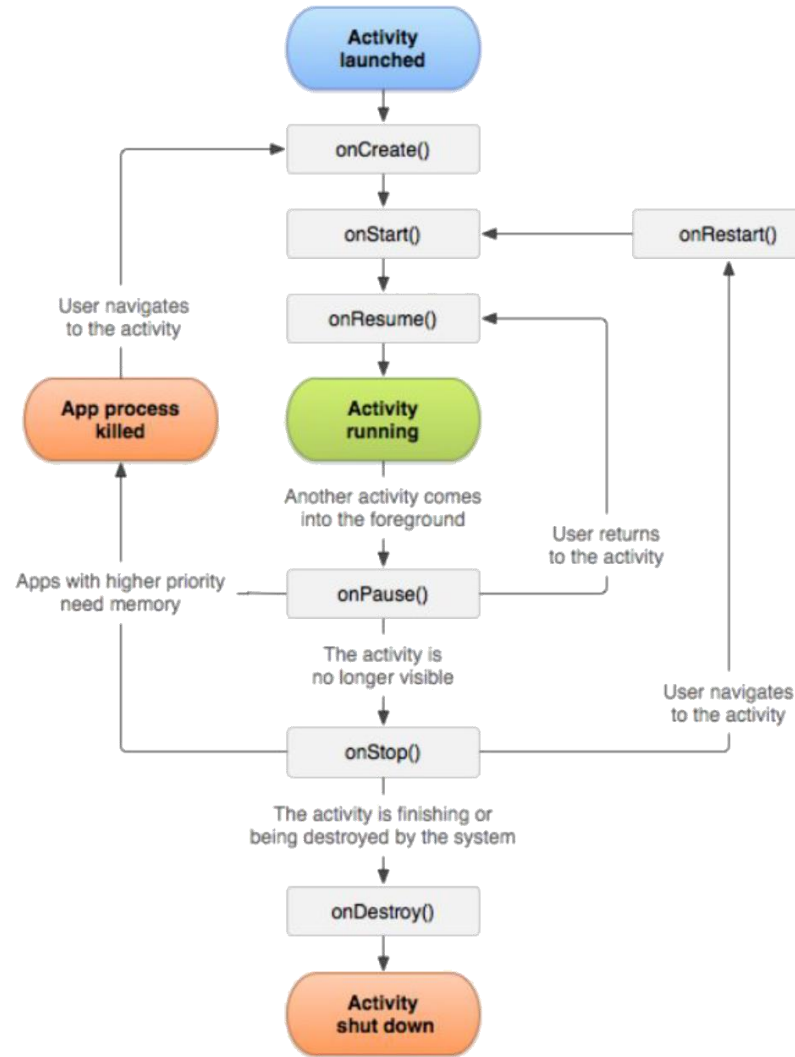
Activity

- **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

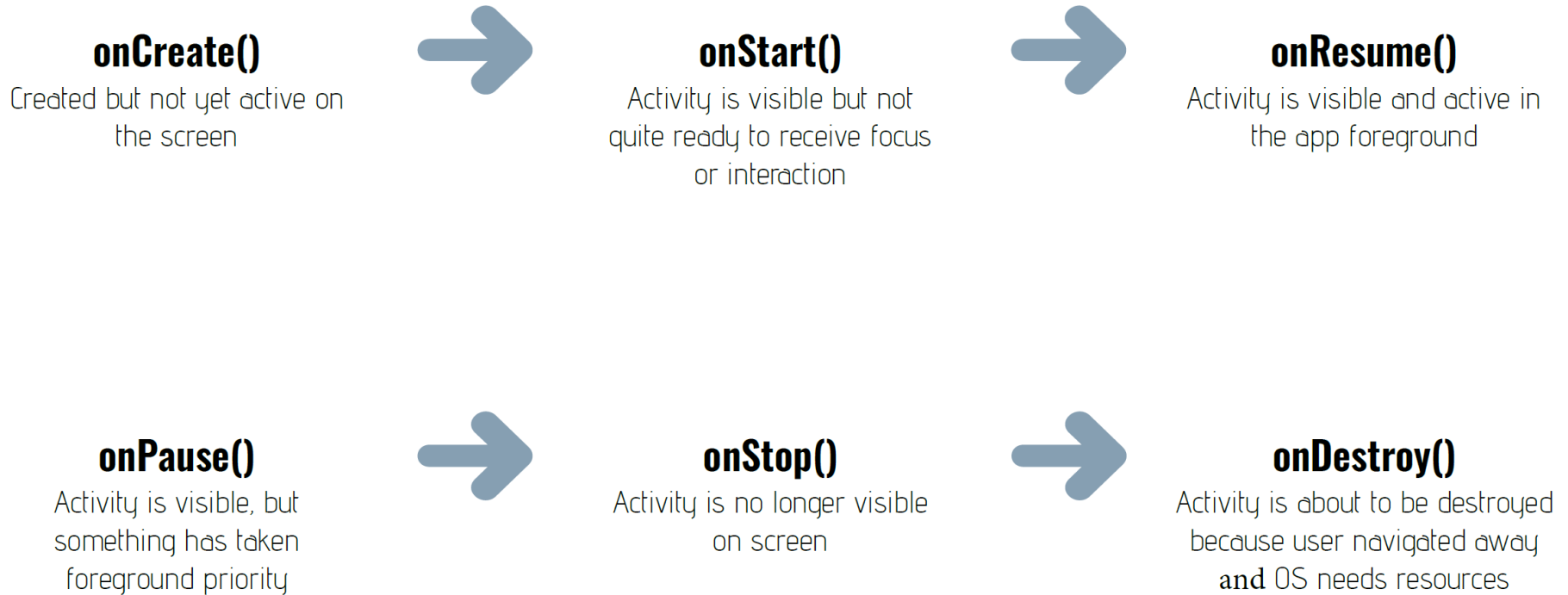
Activity Lifecycle

An activity has essentially **four states**:

- **Active** if the activity is 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 may be killed.
 - When the user navigates to the activity, it must be recreated.

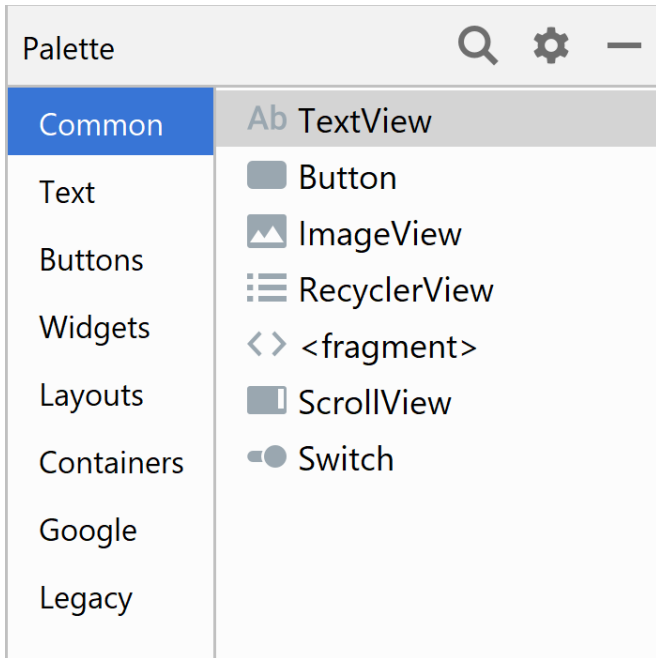


Activity Lifecycle



- Can run events handlers to runs in response to these events

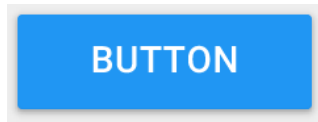
What Makes up an Activity UI?



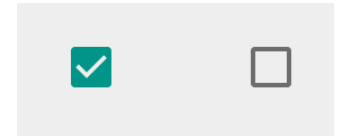
- **Views**
 - Set of pre-built UI components that can be composed to create a GUI
 - e.g. Button, TextView, Menu, List, etc.
- **Layout**
 - Container that controls the size and positioning of views in the Activity

Views

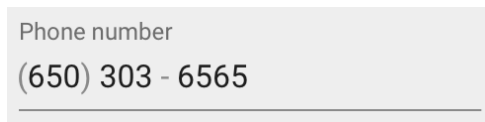
Button



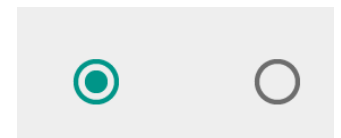
CheckBox



EditText



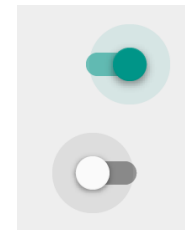
RadioButton



SeekBar



Switch



Views

- **View = Widget = Control**
 - Examples: Button, Switch, Spinner, TextView, EditText, ImageView
 - Advanced Views (covered later): **RecyclerView** & **MapView**
- **Common Attributes**
 - id (i.e. `android:id="@+id/myViewId"`)
 - `layout_width`, `layout_height`
 - Values: `match_constraint` (or `0dp`), `wrap_content`, fixed size (e.g., `50dp`)

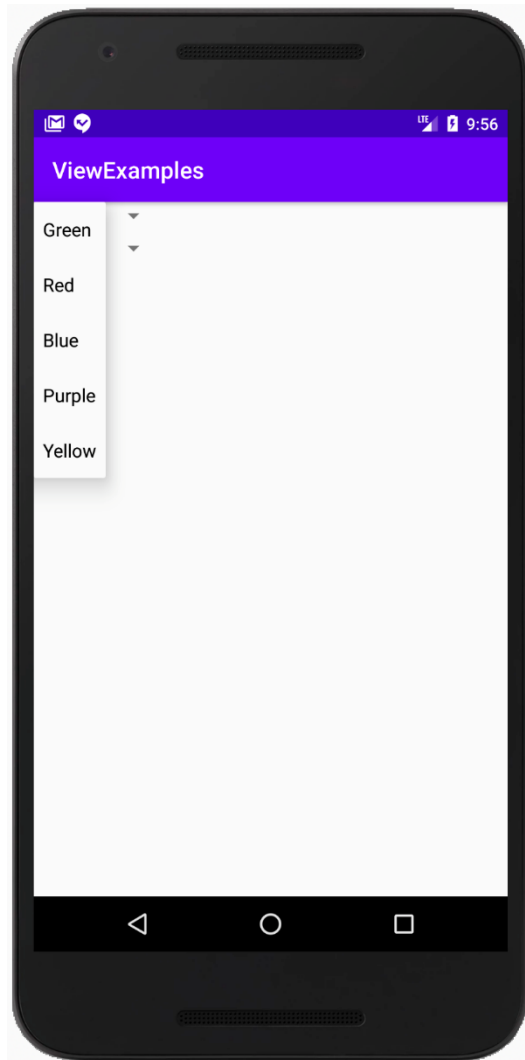
Views (Attributes and Listeners)

- 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 { ... }

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
 - `iconifiedByDefault` – Display the field or just an icon until clicked
 - `.setIconified(boolean)` – make always visible
 - `.setOnQueryTextListener { ... }`

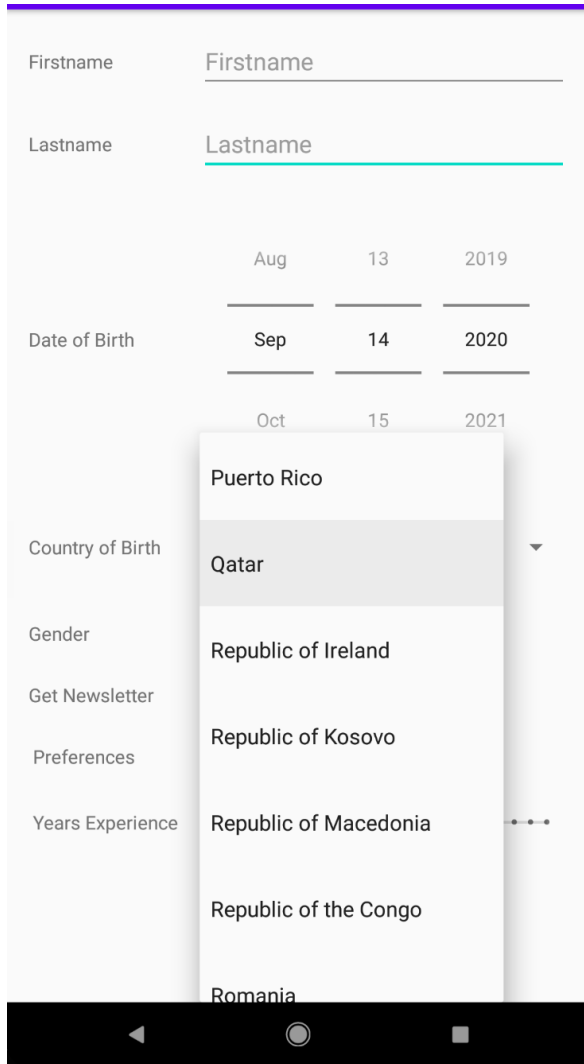
Setting Entries of a Spinner in the XML Layout File



```
<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
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



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

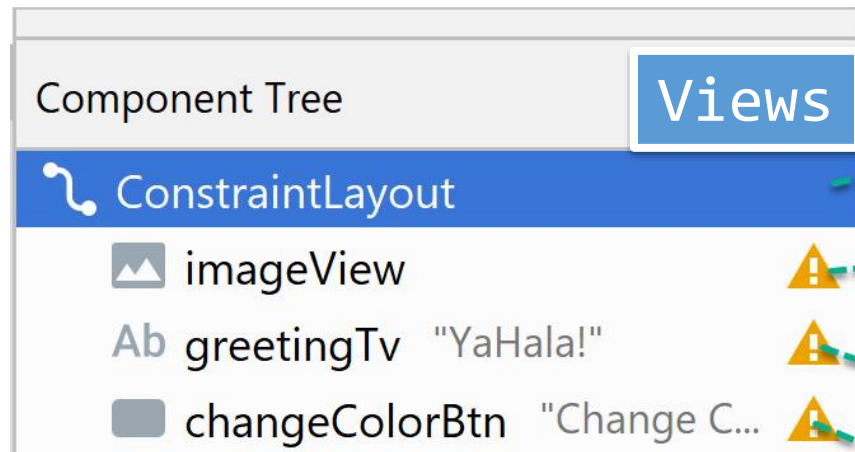
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_register)

    CountryRepository.loadCountries(this)

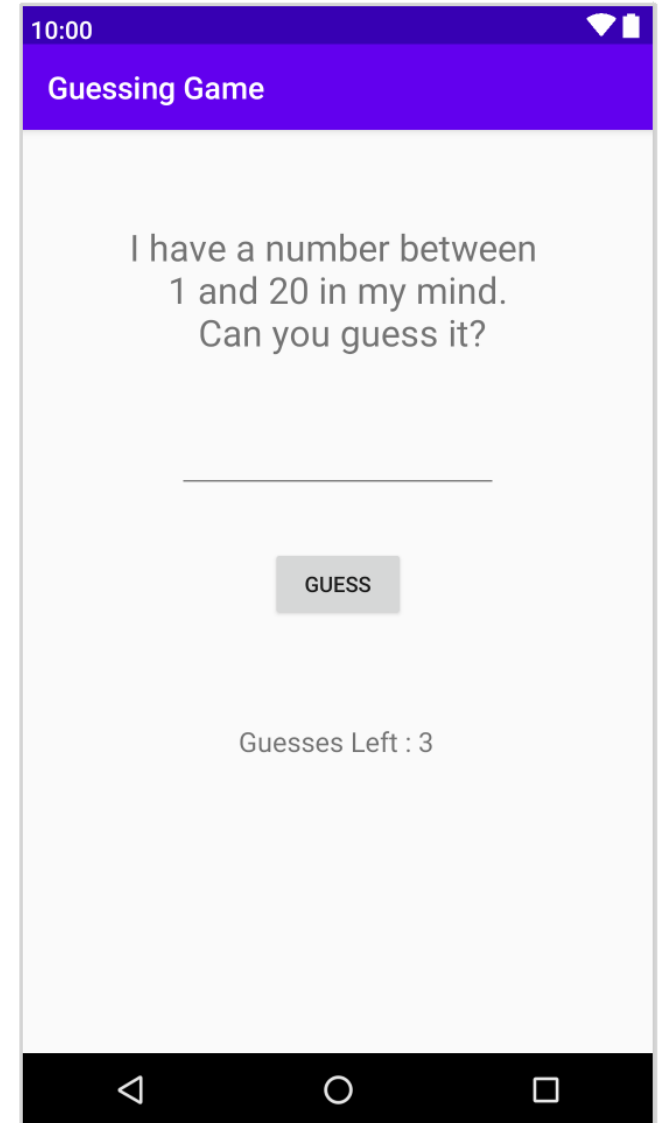
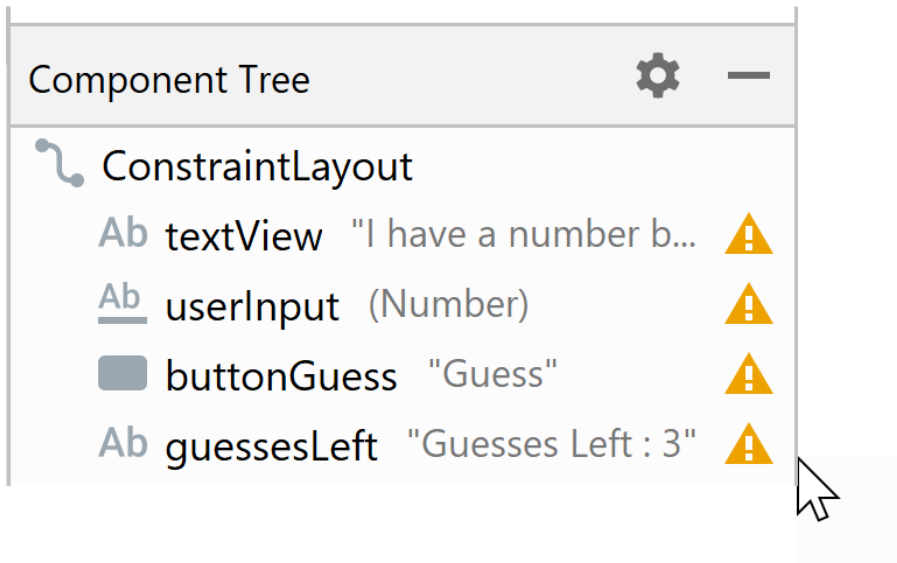
    val adapter = ArrayAdapter<String>(
        this,
        android.R.layout.simple_dropdown_item_1line,
        CountryRepository.countryNames
    )
    countriesSp.adapter = adapter
}
```

App 1 - Color Changer

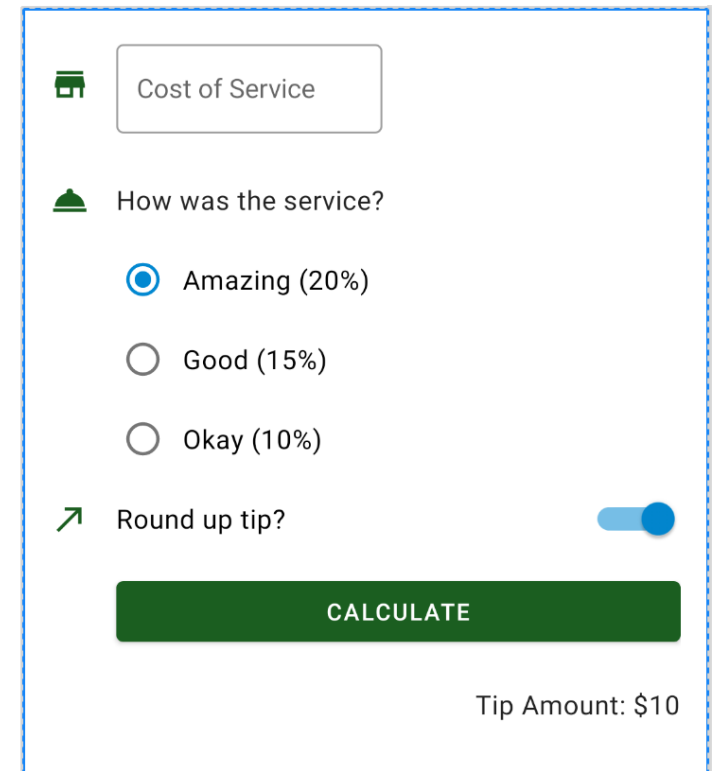
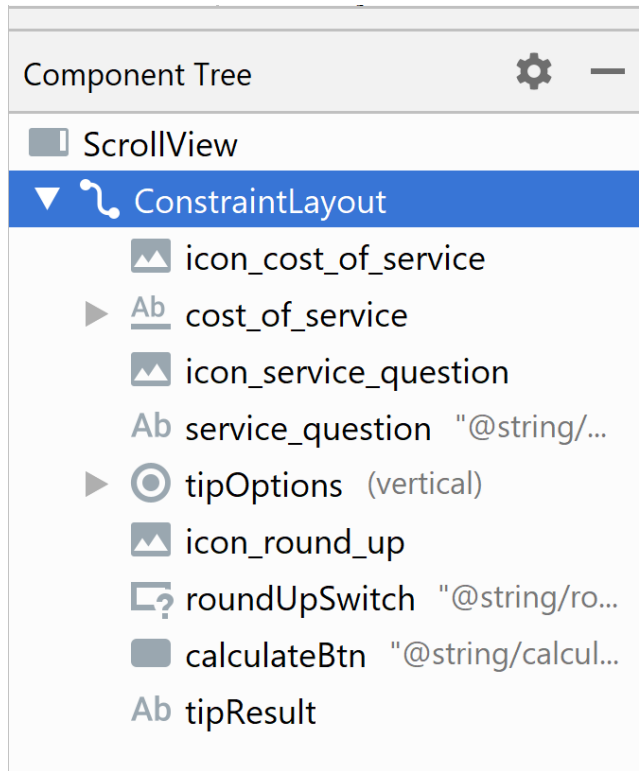
App that contains Text reading “YaHala!”, an Image and a **Button** that randomly changes text’s color with every click



App 2 – Guessing Game



App 3 – Tips Calculator



Registration Form

Firstname	<input type="text" value="Firstname"/>		
Lastname	<input type="text" value="Lastname"/>		
Date of Birth	Aug	13	2019
	Sep	14	2020
	Oct	15	2021
Graduated From	Qatar University ▼		
Gender	<input type="radio"/> Male <input type="radio"/> Female		
Get Newsletter	<input type="checkbox"/>		
Preferences	<input type="checkbox"/> Email <input type="checkbox"/> SMS		
Years Experience	<input type="range" value="2"/>		

Material Design Components

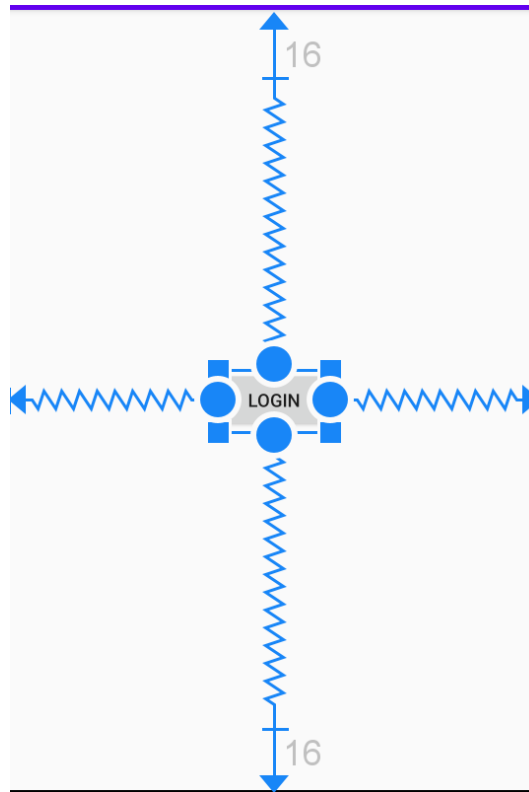
- Using MDC to make your app look great easily

<https://material.io/components>

- FloatingActionButton
- NavigationDrawer
- TextInputLayout
- Toolbar
- CardView
- TabLayout
- BottomNavigationView
- BottomSheet
- Snackbar



Constraint Layout



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

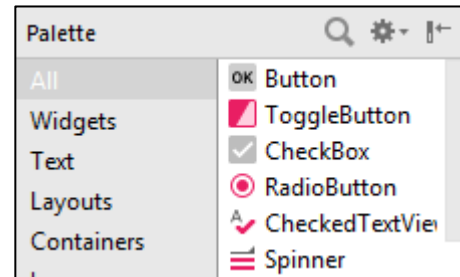
Constraint Layout

- ConstraintLayout: Allows building a Responsive UI by defining constraints for views
 - A constraint is a **connection** to another view, parent layout, or invisible Guideline / Barrier
 - 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

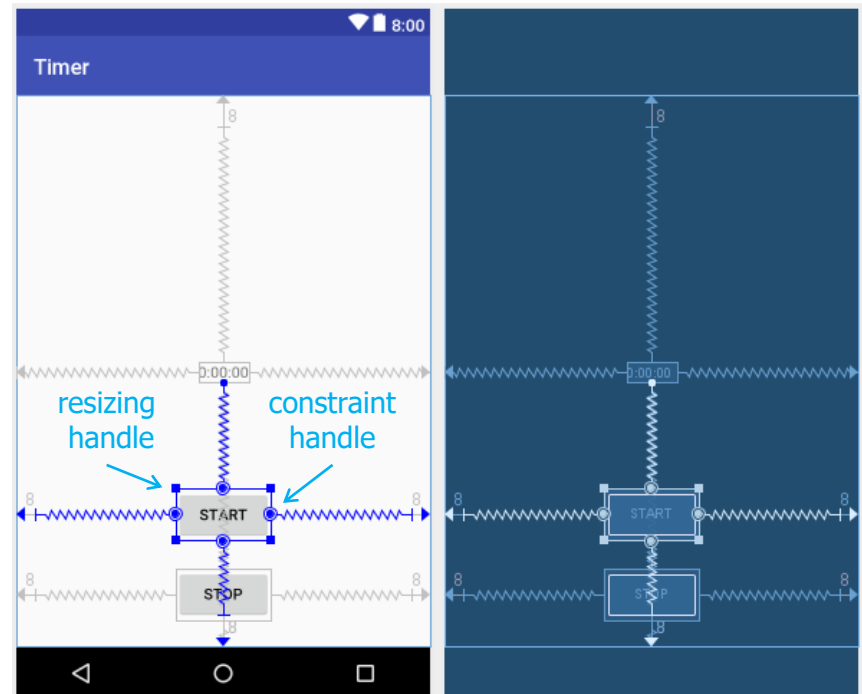
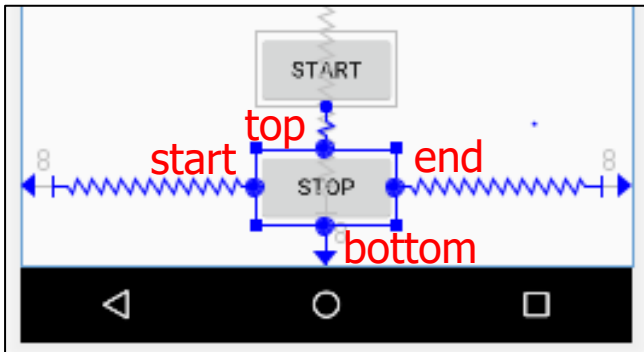
Defining Constraints

Steps

1. Drop a view to the editor
2. Connect constraint handles
(e.g., top/bottom/left/right)

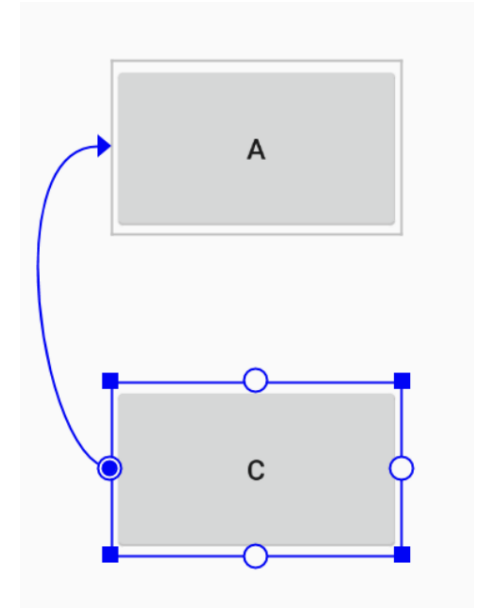


Add at least **one horizontal** and **one vertical** constraint



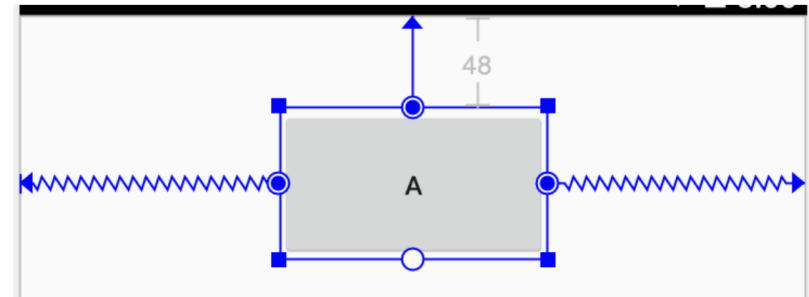
Alignment

- 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.
 - 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



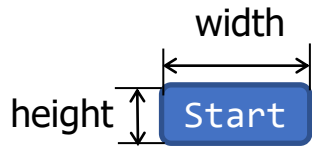
View Constraints Editor

Constraint Widget

Diagram illustrating the View Constraints Editor interface. The central widget is connected to four parent widgets (top, bottom, left, right) via lines with arrowheads. Each connection has a numeric value in a box: top (16), bottom (16), left (0), and right (0). A vertical slider on the left is labeled '50' and 'Constraint Bias'. A horizontal slider at the bottom is labeled '50'. A green callout 'Delete Constraint' points to the left connection. A green callout 'Margins' points to the right connection. A green callout 'Height / Width Mode' points to the bottom connection. A green callout 'Constraint Bias' points to the bottom slider.

▼ Constraints

- Start → StartOf **parent** (0dp)
- End → EndOf **parent** (0dp)
- Top → TopOf **parent** (16dp)
- Bottom → BottomOf **parent** (16dp)
- Horizontal Bias (0.5)



View Size



`layout_width="0dp"`

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



`layout_width="wrap_content"`

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

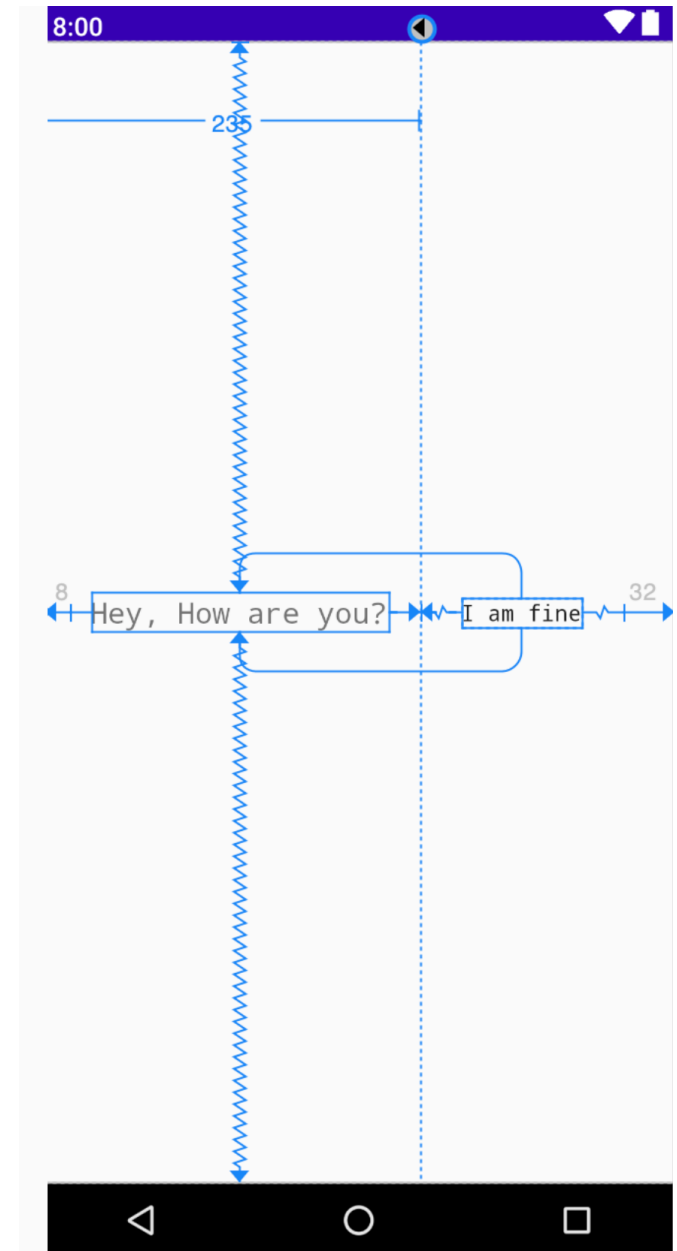


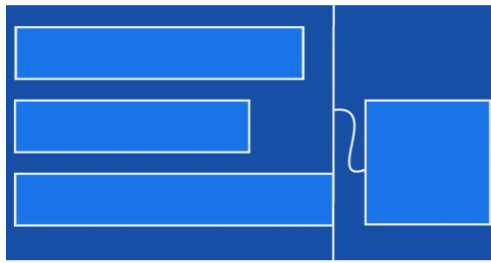
`layout_width="200dp"`

- **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 invisible 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

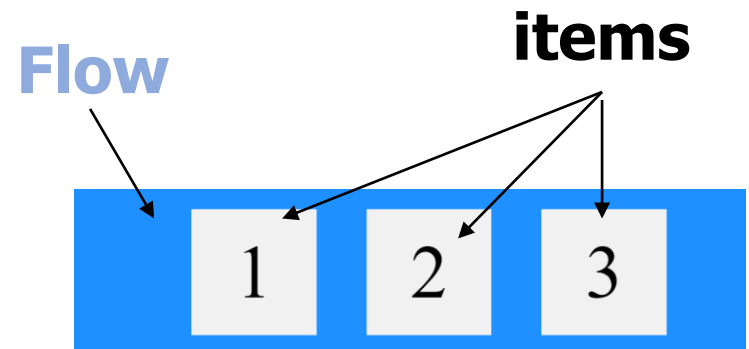
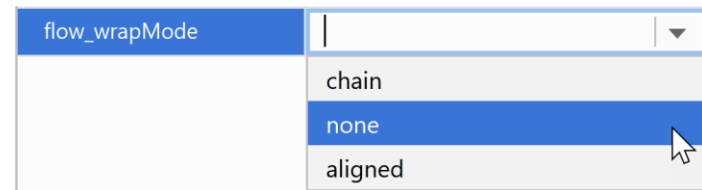
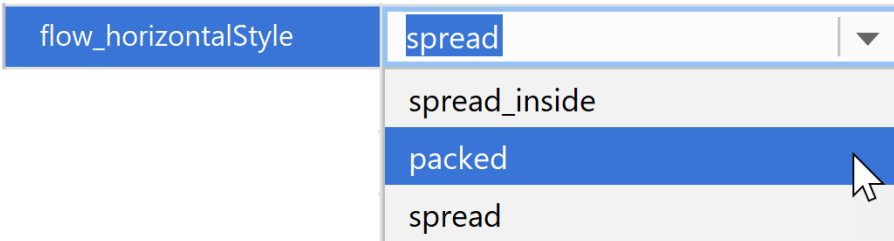
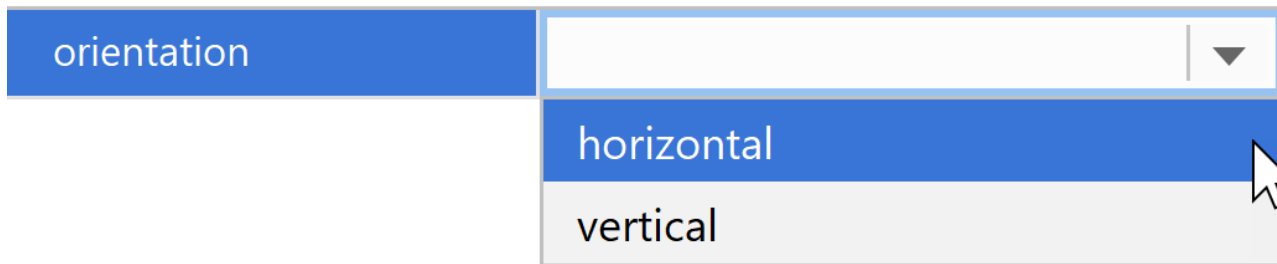
- A Barrier is a virtual view, similar to a Guideline, to which we can constrain objects.
- The Barrier width/height are determined by the views placed in it
- You'll want to use a barrier any time the views placed in it **could dynamically vary in size** based on user input or language setting



```
<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" />
```

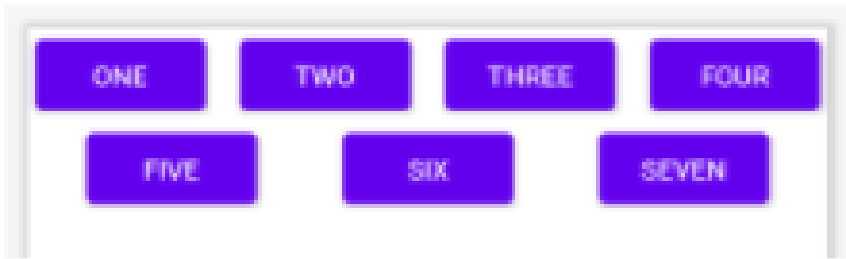
Flow

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



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/>