

# CMPS 312

Read Chapter 1 & 6



## Views & Layout

**Dr. Abdelkarim Erradi**  
**CSE@QU**

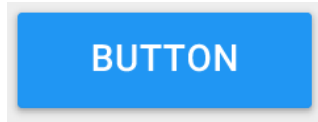
# Outline

1. Views
2. Constraint Layout

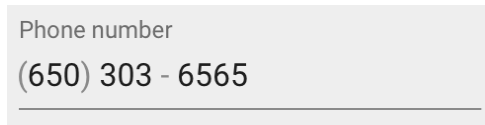
# Views

# View Examples

Button



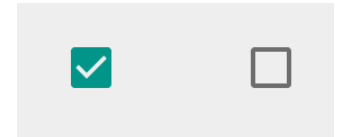
EditText



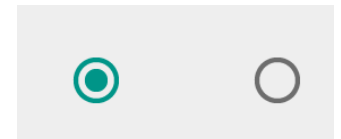
Slider



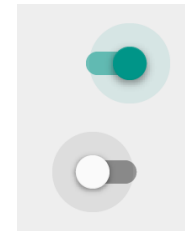
CheckBox



RadioButton



Switch



# Views and ViewGroups

- **View = Widget = Control**
  - Examples: Button, Switch, Spinner, TextView, EditText
  - Advanced Views (covered later): RecyclerView & MapView
- **ViewGroup = Container**
  - Views that contain other views
  - Examples: LinearLayout, TableLayout, ScrollView
- **Common Attributes**
  - id (i.e. **android:id="@+id/myViewId"**)
  - Layout\_width, layout\_height
    - Values: match\_parent (or 0dp), wrap\_content, 16dp

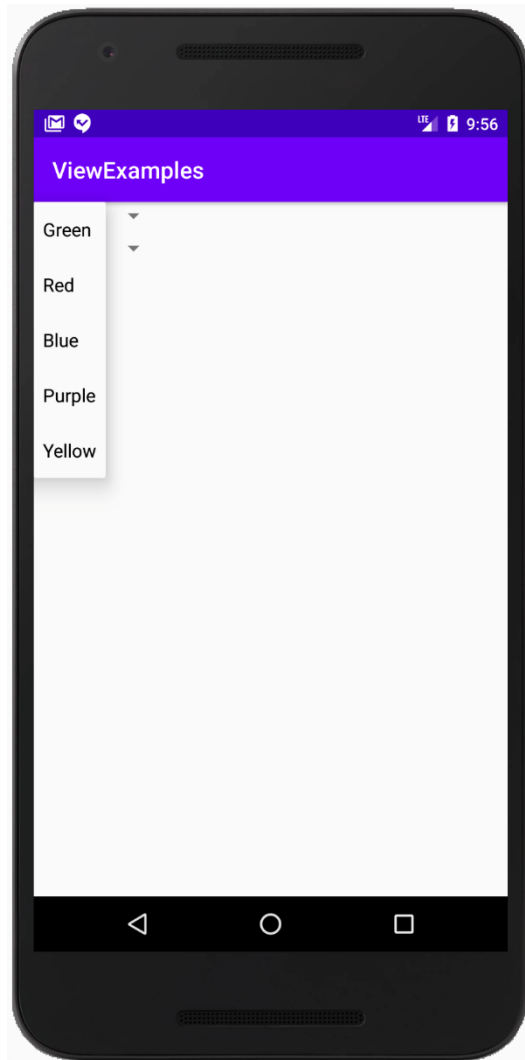
# Views (Attributes and Listeners)

- TextView (text labels)
  - text, textAppearance
  - .setClickable(boolean) – make clickable
  - .setOnClickListener(View.OnClickListener)
- EditText (text fields)
  - inputType
  - .addTextChangedListener(TextWatcher)
- Button
  - text
  - .setOnClickListener(View.OnClickListener)
- ImageView (display image)
  - .setClickable(boolean) – make clickable
  - .setImageDrawable(Drawable) – set icon to display
  - .setOnClickListener(View.OnClickListener)

# Views (Attributes and Listeners)

- **Switch (on/off)**
  - `.setChecked(boolean)` – set check state
  - `.setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener)`
- **Spinner (dropdown list)**
  - `entries` or `.setAdapter(ArrayAdapter)` – specify list values
  - `.setSelection(int)` – specify selected item
  - `onItemSelectedListener(`  
`AdapterView.OnItemSelectedListener)`
- **SearchView**
  - `queryHint` – Background text displayed when the field is empty
  - `iconifiedByDefault` – Display the field or just an icon until clicked
  - `.setIconified(boolean)` – make always visible
  - `.setOnQueryTextListener(SearchView.OnQueryTextListener)`

# Setting Entries of a Spinner in the XML Layout File

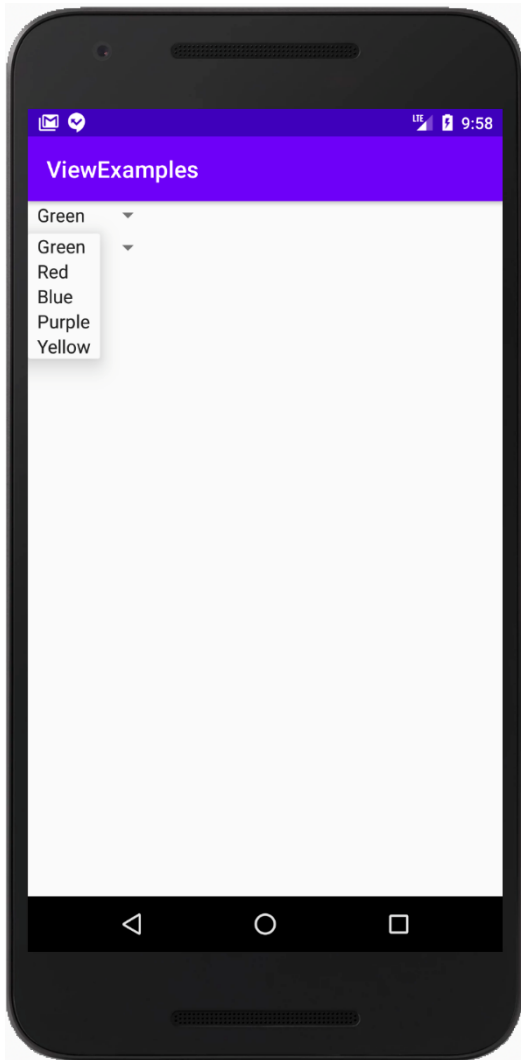


```
<Spinner
    android:id="@+id/colorSelector1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="32dp"
    android:layout_marginBottom="4dp"
    android:entries="@array/colorChoices"/>
```

```
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/colorSelector2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="32dp"
    android:layout_marginBottom="4dp"/>
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Spinner colorSelector2 = findViewById(R.id.colorSelector2);
    ArrayAdapter<CharSequence> adapter =
        ArrayAdapter.createFromResource(context: this,
            R.array.colorChoices, android.R.layout.simple_spinner_item);
    colorSelector2.setAdapter(adapter);
}
```

# Family Map Client: Login Fragment

- How would you create this view?
- Does it need to scroll?

Family Map

Server Host: 10.0.2.2

Server Port: 8080

User Name: jane

Password: ....

First Name: Jane

Last Name: Smith

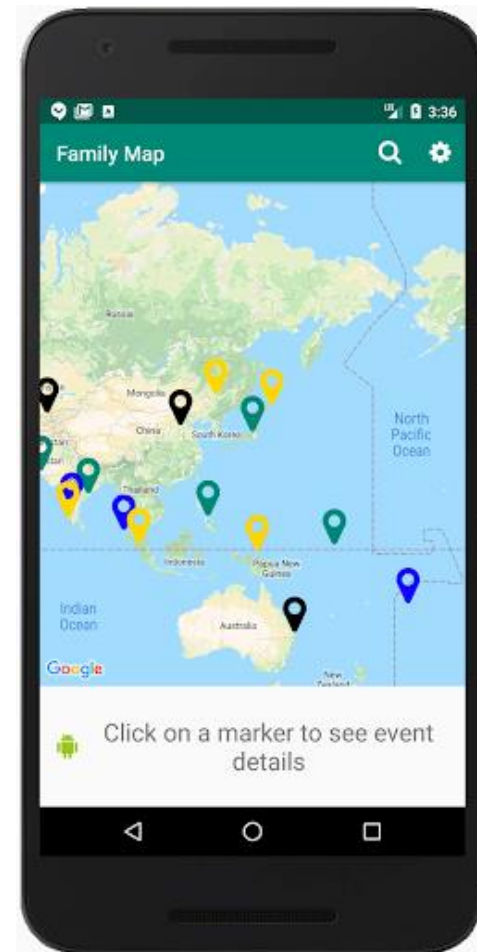
Email: jane@gmail.com

Gender: ☐ Male ☒ Female

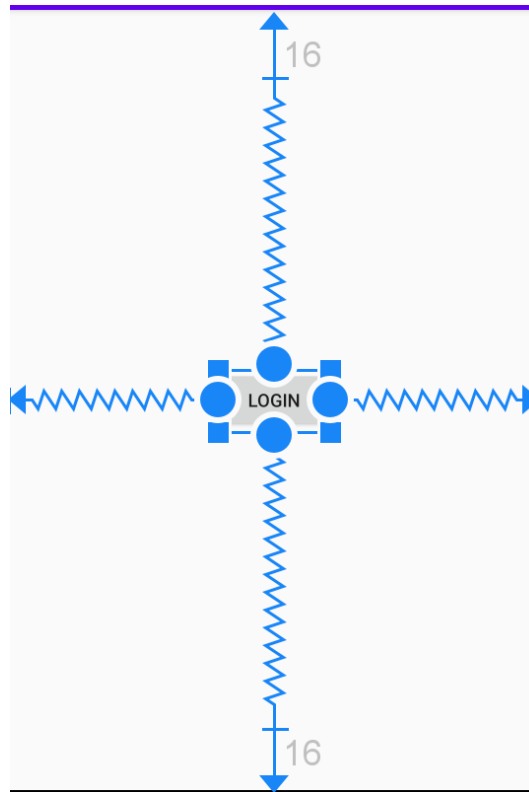
SIGN IN REGISTER

# Family Map Client: Map Fragment

- How would you create this view?
- Does it need to scroll?



# 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 UI components reorganize themselves based on the rules of the layout

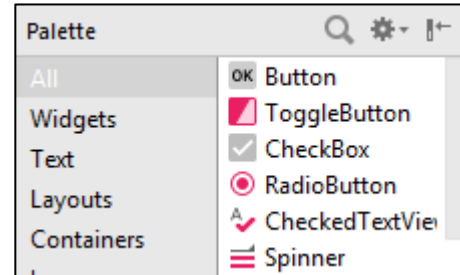
# Constraint Layout

- ConstraintLayout: Allows building a Responsive UI by connecting views with constraints
  - Position a view relative others including the parent
  - Need to add at least one horizontal and one vertical constraint
  - Constraint is a connection to another view, parent layout, or invisible Guideline / Barrier
  - Uses constraints to determine the position and alignment of UI elements
  - Allows positioning UI elements in various ways: relative, centered, using **flow**

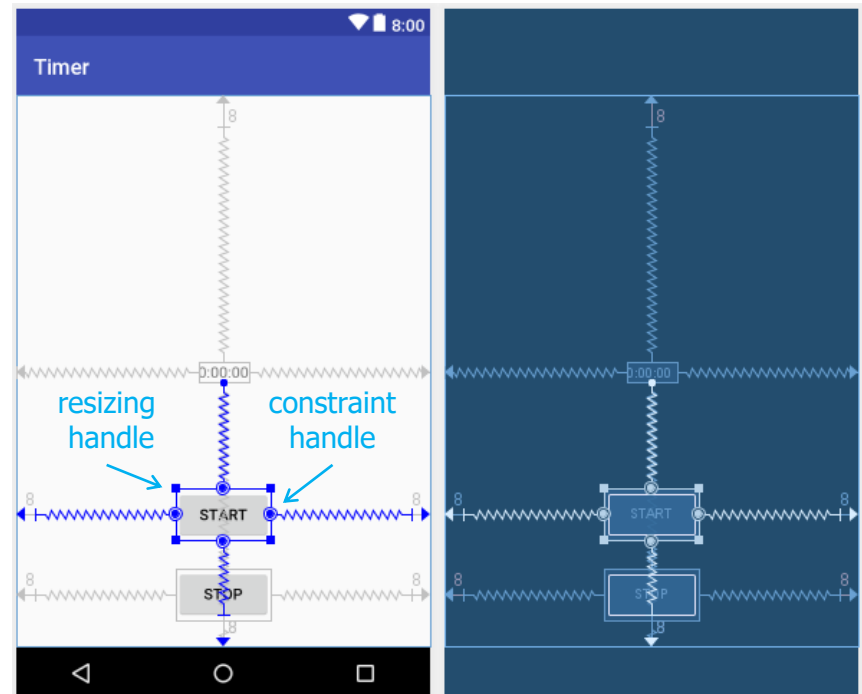
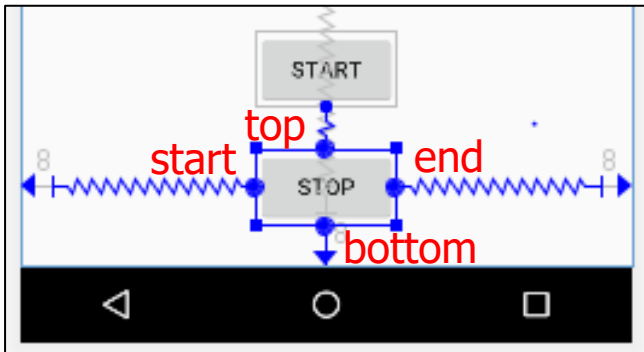
# Defining Constraints

## Steps

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

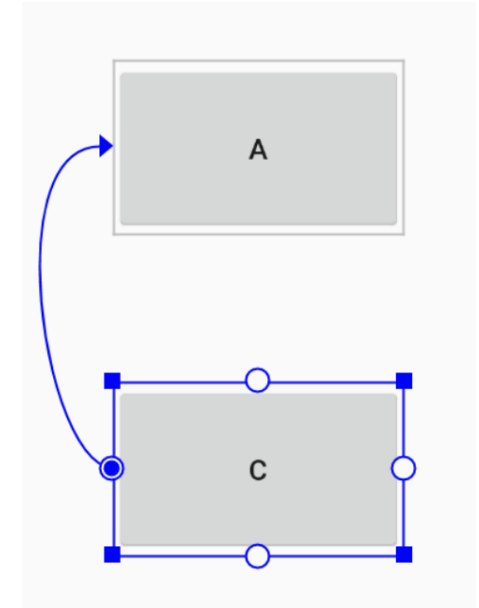


At least **one horizontal** and **one vertical** constraint



# Alignment

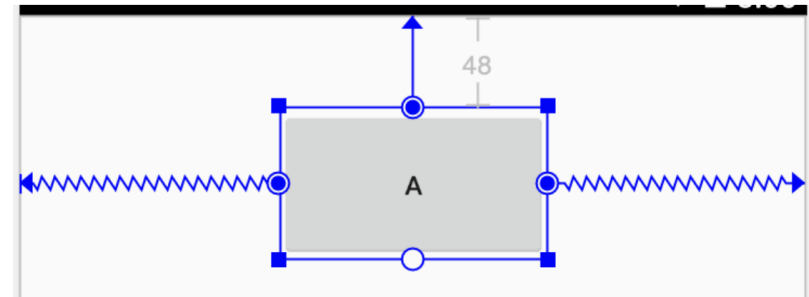
- Align the edge of a view to the same edge of another view.
- The left side of C is aligned to the left side of A. If you want to align the view centers, create a constraint on both sides





# 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



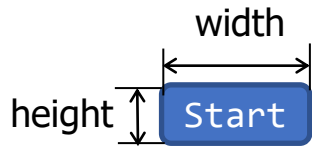
# View Constraints Editor

Constraint Widget

Diagram illustrating the View Constraints Editor interface. The central widget is connected to four parent widgets (represented by boxes with '16' and a dropdown) via lines with arrowheads. The connections are labeled with '>>' (right) and '<<' (left) for horizontal constraints, and '>>' (down) and '<<' (up) for vertical constraints. A green callout 'Delete Constraint' points to the left horizontal connection. A green callout 'Margins' points to the right horizontal connection. A green callout 'Height / Width Mode' points to the bottom vertical connection. A green callout 'Constraint Bias' points to a slider below the widget, which is set to 50. A green callout '50' points to a circular indicator on the left vertical connection.

▼ Constraints

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

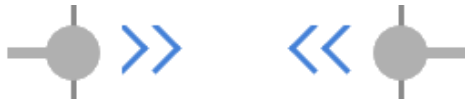


# View Size



`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

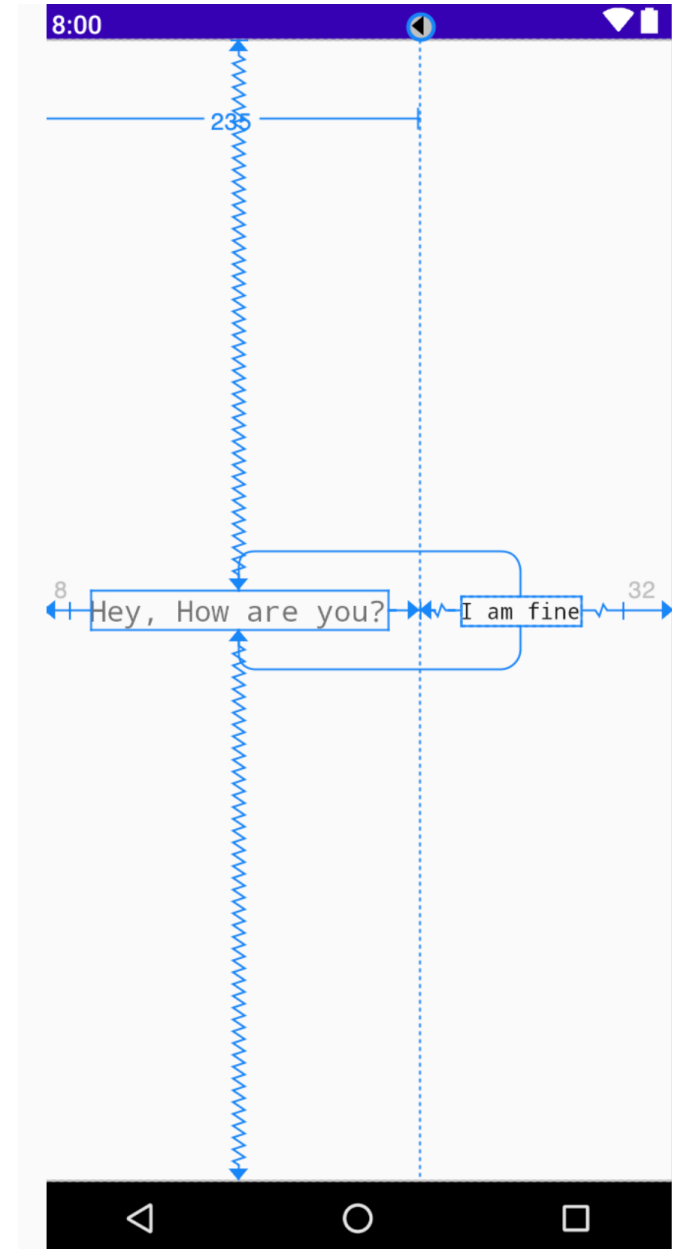


`layout_width="200dp"`

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

# Guideline

- 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** units or percent, relative to the layout's edge



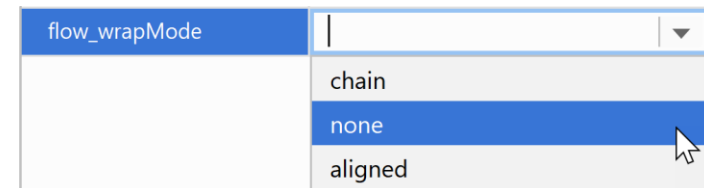
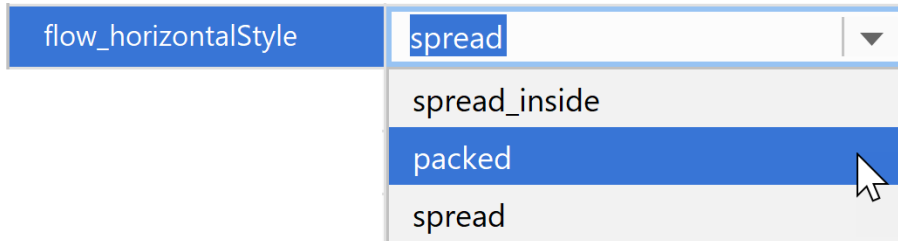
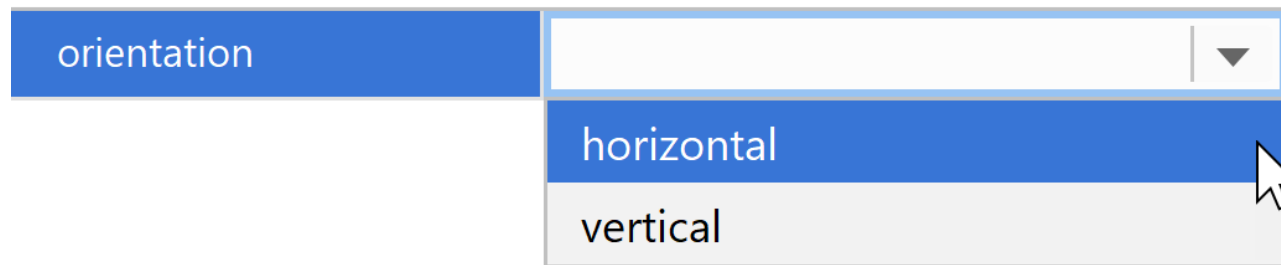
# Barrier





```
<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



# Summary

- ConstraintLayout enables responsive design  
.. mastering it will take some time and effort   ...