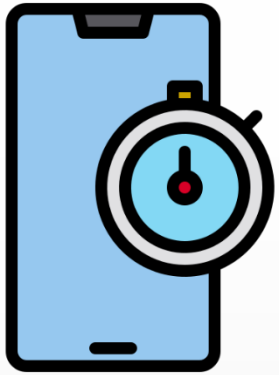
Build a Simple User Interface

Dmytro Zubov, PhD
dmytro.zubov@ucentralasia.org

Naryn, 5:51 pm, Aug 3, 2022

Lessons learnt last time

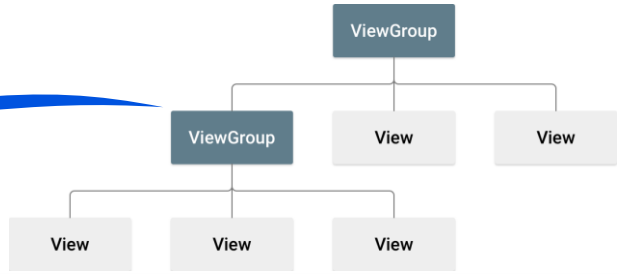
- Intro to Java programming language
- History of Android OS briefly
- Get the free software we need
- Analyzing an Android App structure
- Creating an Android app



What we gonna discuss today?

- Build a Simple User Interface
- Comments in Java
- Identifiers in Java
- Variables in Java
- Java types
- Compound expressions in Java
- Bitwise operators in Java

Build a Simple User Interface

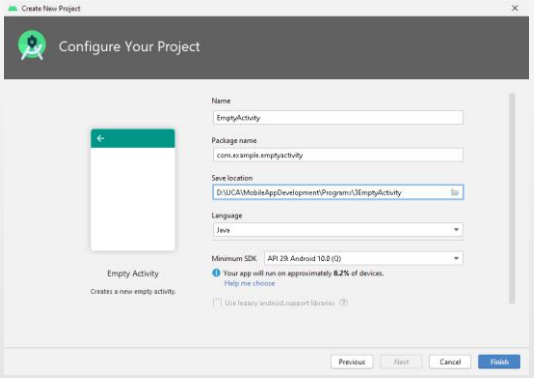


- The UI for an Android app is built as a hierarchy of *layouts* and *widgets*
- The layouts are `ViewGroup` objects, containers that control how their child views are positioned on the screen
- Widgets are `View` objects, UI components such as buttons and text boxes
- Android provides an XML vocabulary for `ViewGroup` and `View` classes, so UI can be defined in XML files. We will create a layout using Android Studio's Layout Editor. The Layout Editor writes the XML for us as we drag and drop views to build our layout.

https://developer.android.com/training/basics/firstapp/building-ui

Build a Simple User Interface (cont.)

1. Create an Android project based on the **Empty activity** template
2. In the Project window, open
   `app > res > layout > activity_main.xml`
3. If editor shows the XML source, click the `Design` tab
4. Click `Select Design Surface` and select `Blueprint`
5. Click `Show` in the Layout Editor toolbar and make sure that `Show All Constraints` is checked

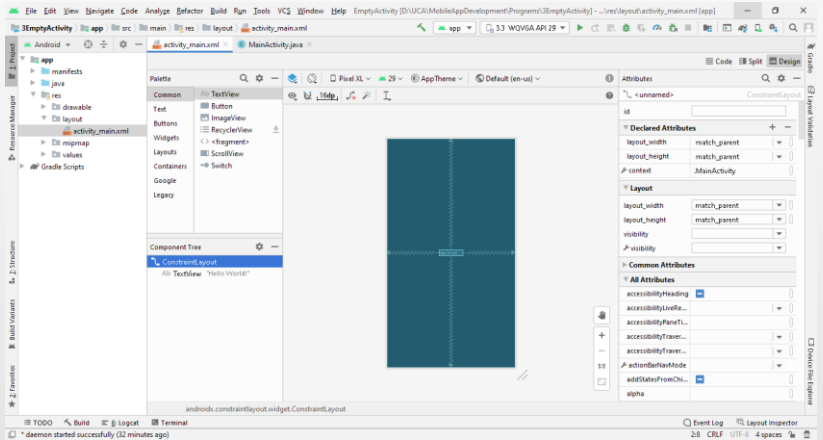


Build a Simple User Interface (cont.)

6. Make sure Autoconnect is off. A tooltip in the toolbar displays `Enable Autoconnection to Parent` when Autoconnect is off.

7. Click `Default Margins` in the toolbar and select **16**. If needed, we can adjust the margins for each view later.

8. Click `Device for Preview` in the toolbar and select **5.5, 1440 × 2560, 560 dpi (Pixel XL)**



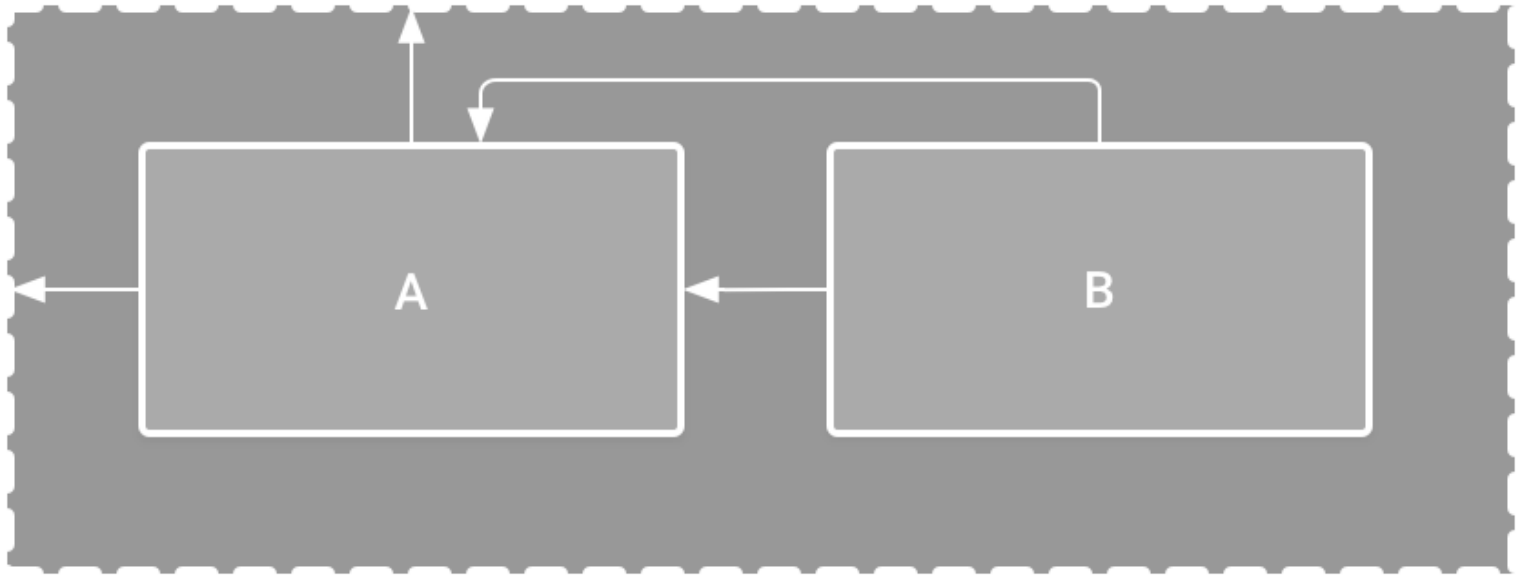
Pls try other devices :)

ConstraintLayout

- **ConstraintLayout** is a layout that defines the position for each view based on constraints to sibling views and the parent layout. So, we can create both simple and complex layouts with a flat view hierarchy. This type of layout avoids the need for nested layouts.
- A **nested layout**, which is a layout inside a layout can increase the time required to draw the UI

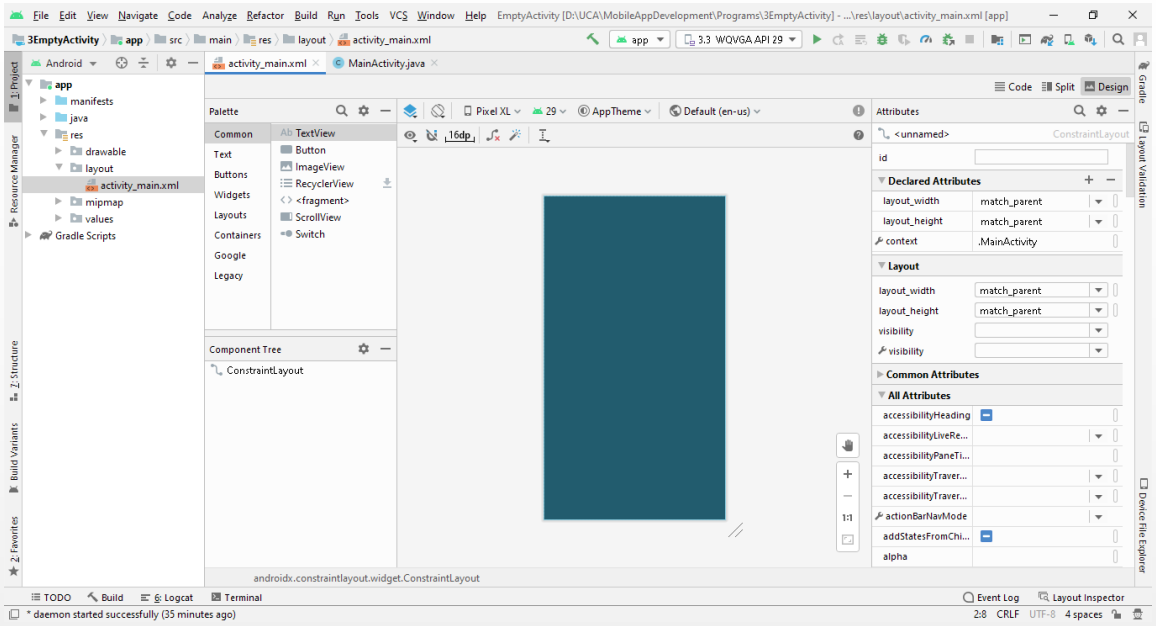
Illustration of two views positioned inside `ConstraintLayout`

- View A appears 16 dp from the top of the parent layout
- View A appears 16 dp from the left of the parent layout
- View B appears 16 dp to the right of view A
- View B is aligned to the top of view A



Build a Simple User Interface: Add a text box

1. Click `TextView` in the `Component Tree` panel and then press the **Delete** key

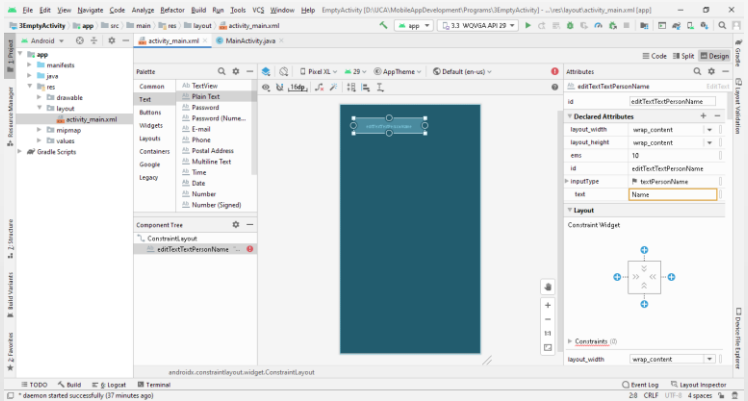


2. In the `Palette` panel, click `Text` to show the available text controls

Build a Simple User Interface: Add a text box (cont.)

3. Drag the `Plain Text` into the design editor and drop it near the top of the layout. This is an `EditText` widget that accepts plain text input

4. Click the view, i.e., `EditText`, in the design editor. We can now see the square handles to resize the view on each corner, and the circular constraint anchors on each side. For better control, we might want to zoom in on the editor. To do so, use the `Zoom` buttons in the Layout Editor toolbar.

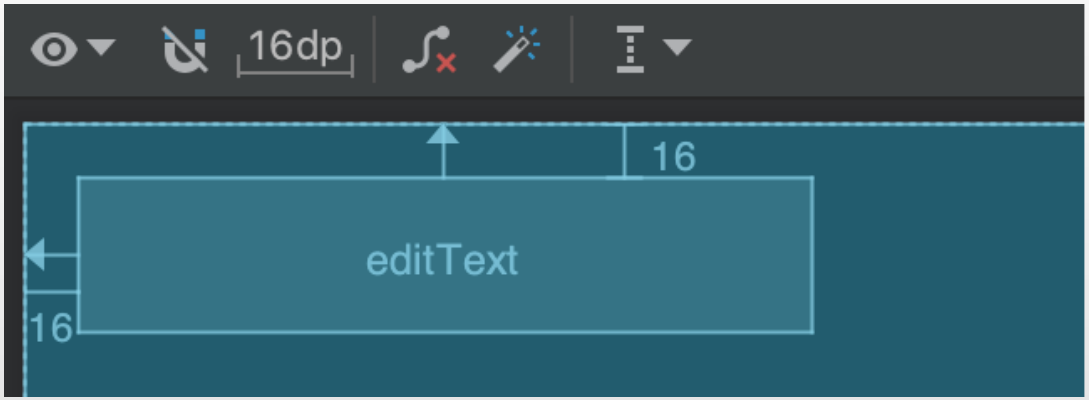


Build a Simple User Interface: Add a text box (cont.)

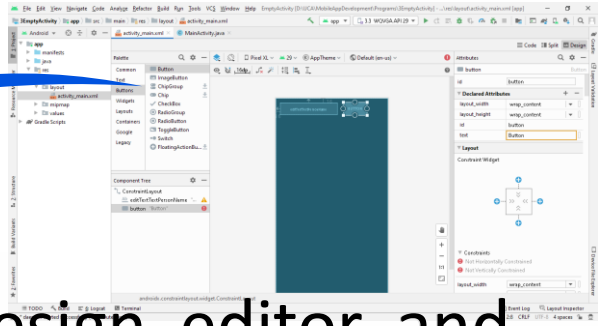
5. Click and hold the anchor on the top side, drag it up until it snaps to the top of the layout, and then release it. **That is the constraint**: it constrains the view within the default margin that was set. In this case, we set it to 16dp from the top of the layout.

6. Use the same process to create a constraint from the left side of the view to the left side of the layout

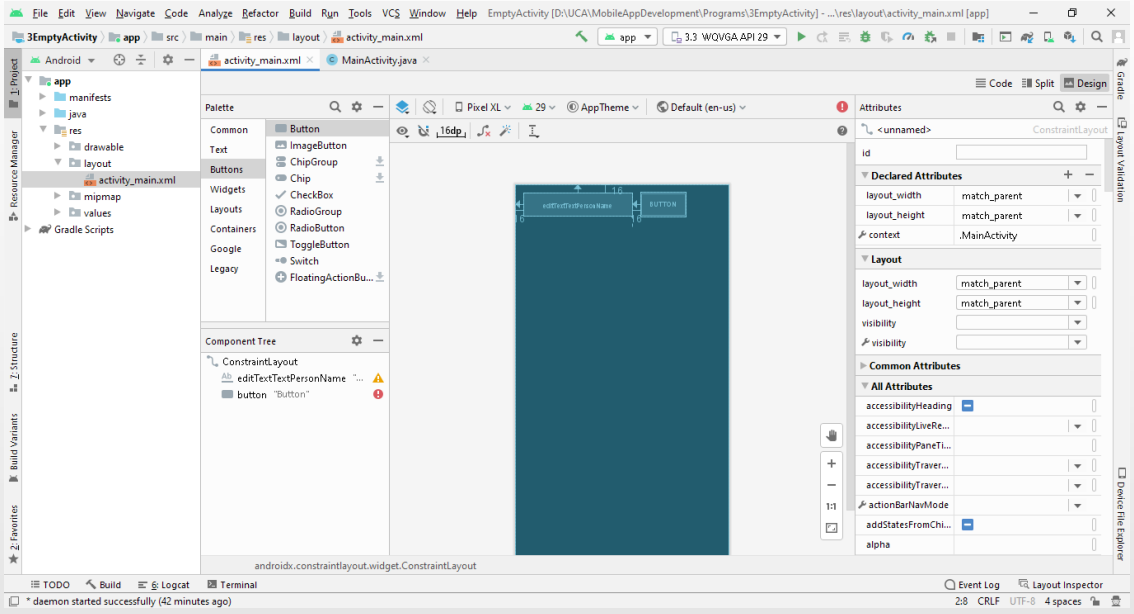
The text box is constrained to the top and left of the parent layout



Build a Simple User Interface: Add a button

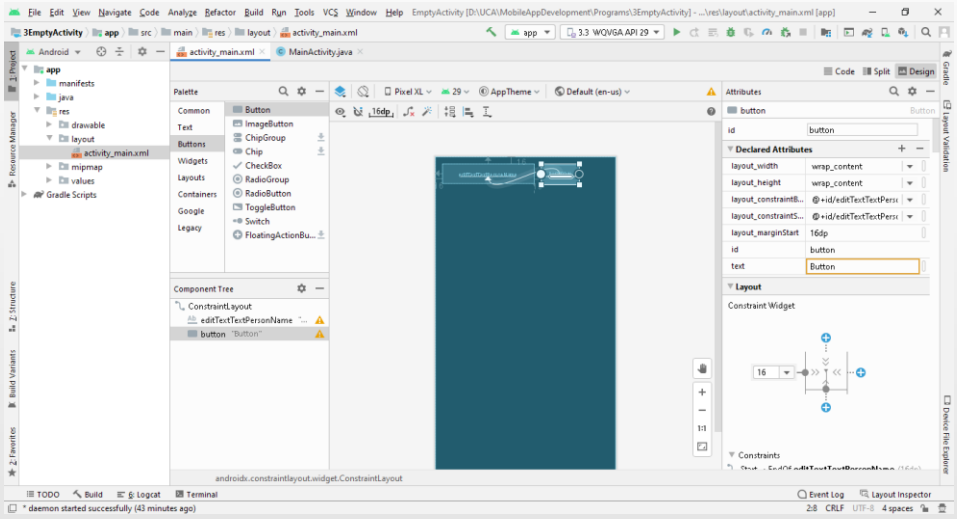


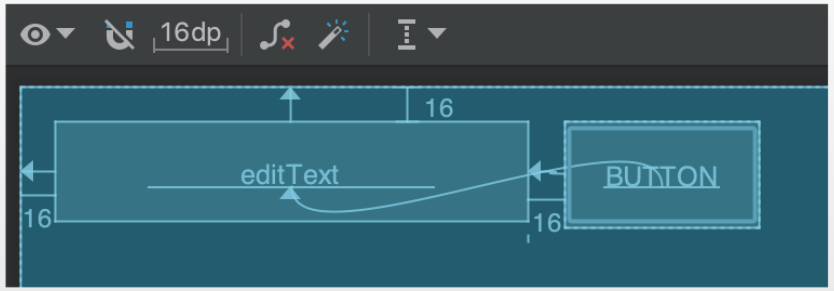
1. In the `Palette` panel, click `Buttons`
2. Drag the `Button` widget into the design editor and drop it near the right side
3. Create a constraint from the left side of the button to the right side of the text box



Build a Simple User Interface: Add a button (cont.)

4. To **constrain the views in a horizontal alignment**, create a **constraint between the text baselines**. To do so, right-click the button and then select Show Baseline . The baseline anchor appears inside the button. Click and hold this anchor, and then drag it to the baseline anchor that appears in the adjacent text box.

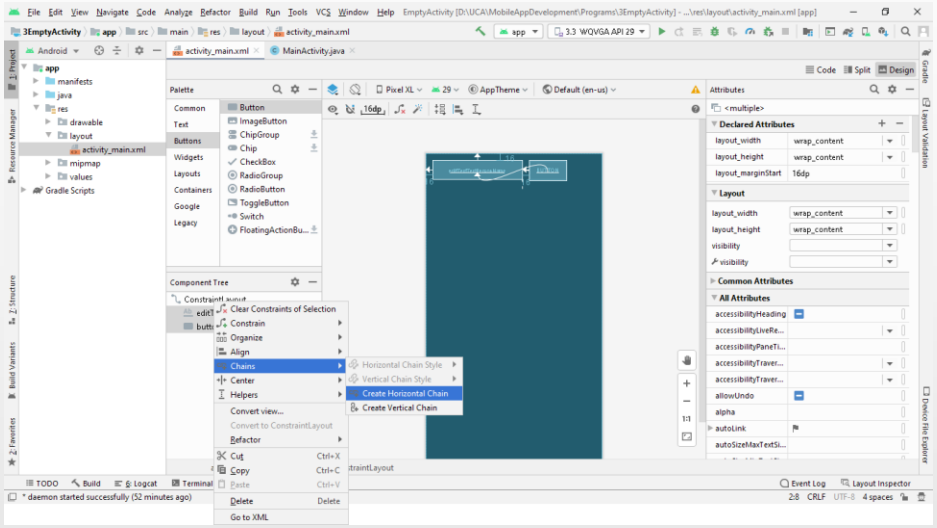


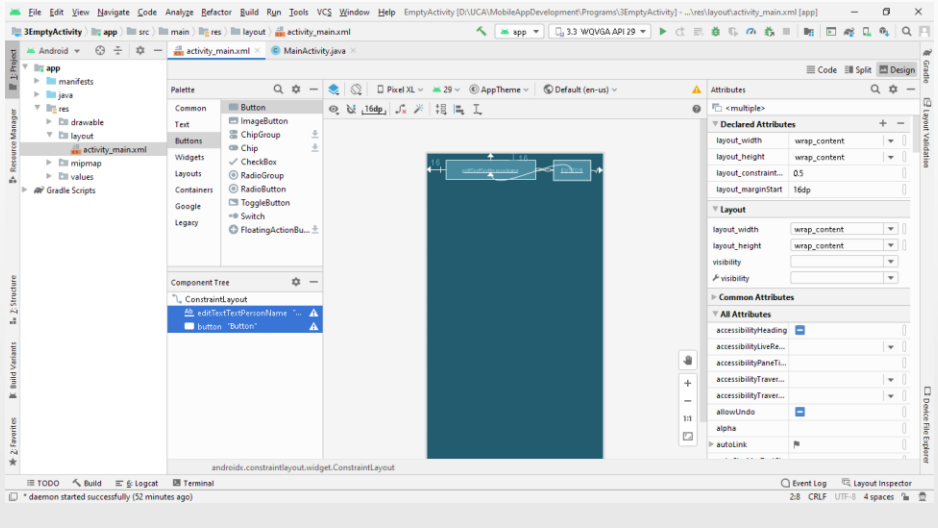


Build a Simple User Interface: Make the text box size flexible

1. Select both views. To do so, click one, hold `Shift`, then click the other, and then right-click either one and select `Chains > Create Horizontal Chain`.

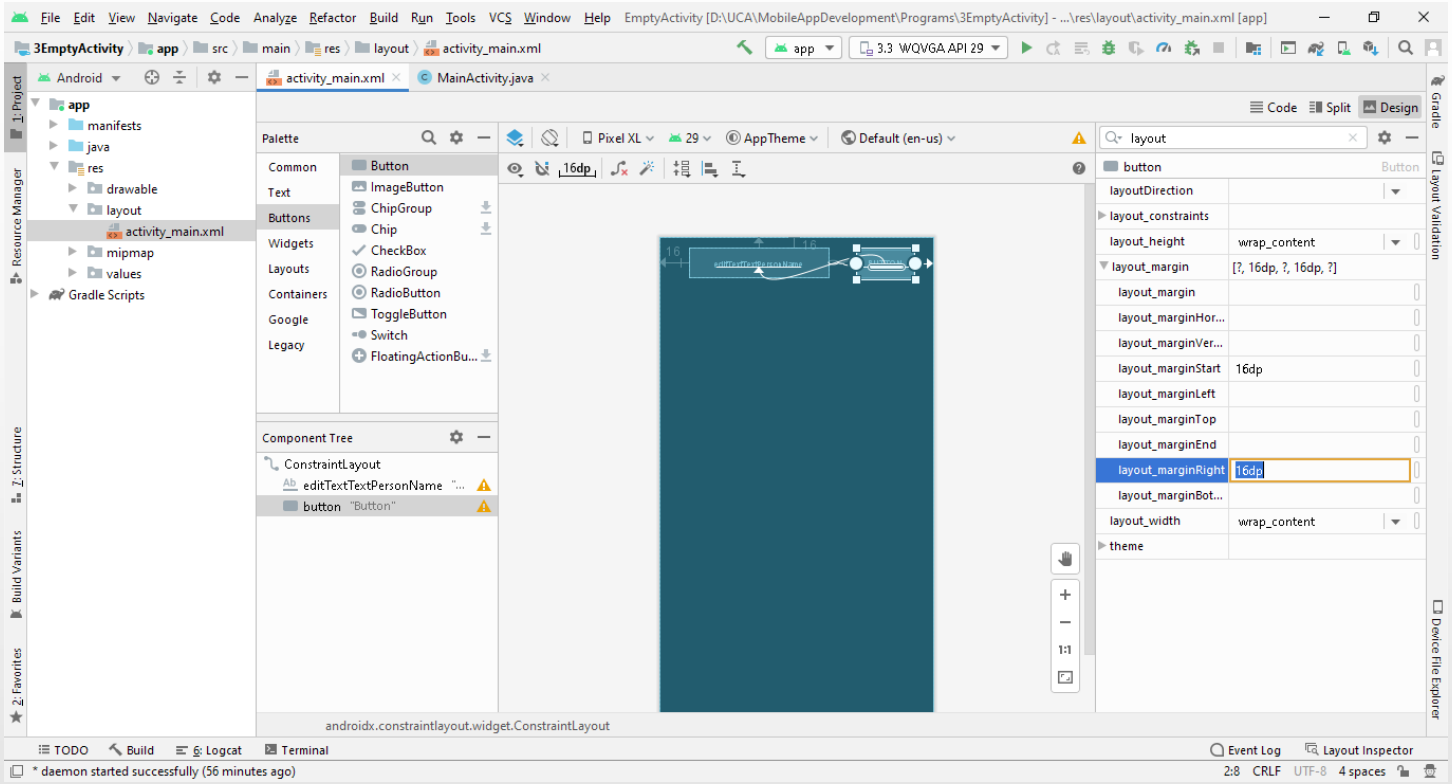
- A chain is a bidirectional constraint between two or more views that allows you to lay out the chained views in unison





Build a Simple User Interface: Make the text box size flexible (cont.)

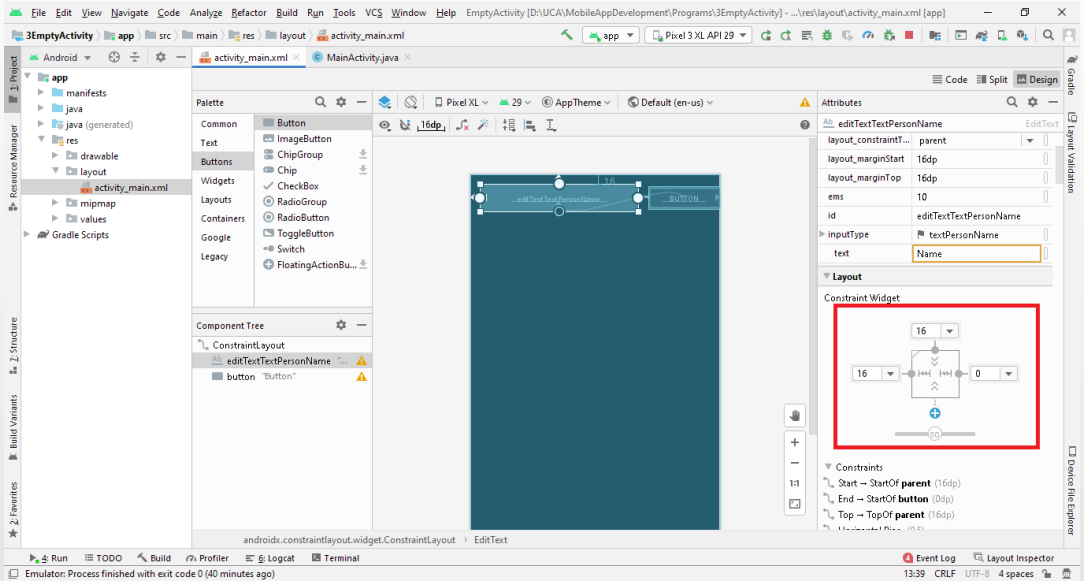
2. Select the button and open the `Attributes` window. Then, use the view inspector at the top of the `Attributes` window to set the right margin to 16dp



Build a Simple User Interface: Make the text box size flexible (cont.)

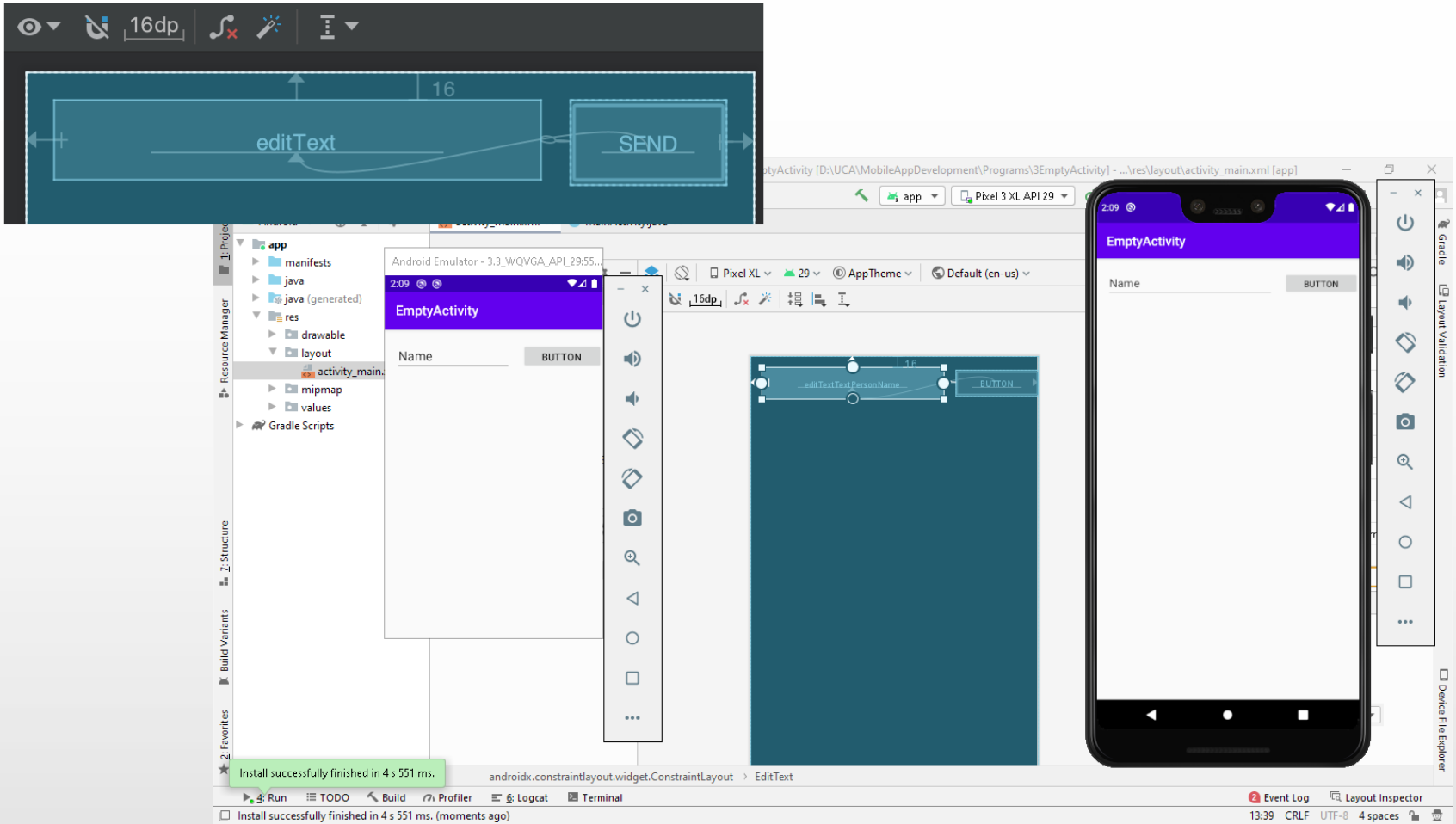
3. Click the text box to view its attributes. Then, click the width indicator twice so it's set to Match Constraints.

- Match constraints means that the width expands to meet the definition of the horizontal constraints and margins. Therefore, the text box stretches to fill the horizontal space that remains after the button and all the margins are accounted for.



Build a Simple User Interface: Make the text box size flexible (cont.)

- The text box now stretches to fill the remaining space



Comments

- **Single-line comment** occupies all or part of a single line of source code. This comment begins with the // character sequence and continues with explanatory text.

```
//x=Double.parseDouble(editText.getText().toString());
```

- **Multiline comment** occupies one or more lines of source code. This comment begins with the /* character sequence, continues with explanatory text, and ends with the */ character sequence.

```
/*x=Double.parseDouble(editText.getText().toString());
  y=Double.parseDouble(editText2.getText().toString());*/
```

Identifiers

- An ***identifier*** consists of letters (A-Z, a-z), digits (0-9), connecting punctuation characters (e.g., the underscore), and currency symbols (e.g., the dollar sign $). This name must begin with a letter, a currency symbol, or a connecting punctuation character; and its length cannot exceed the line in which it appears.

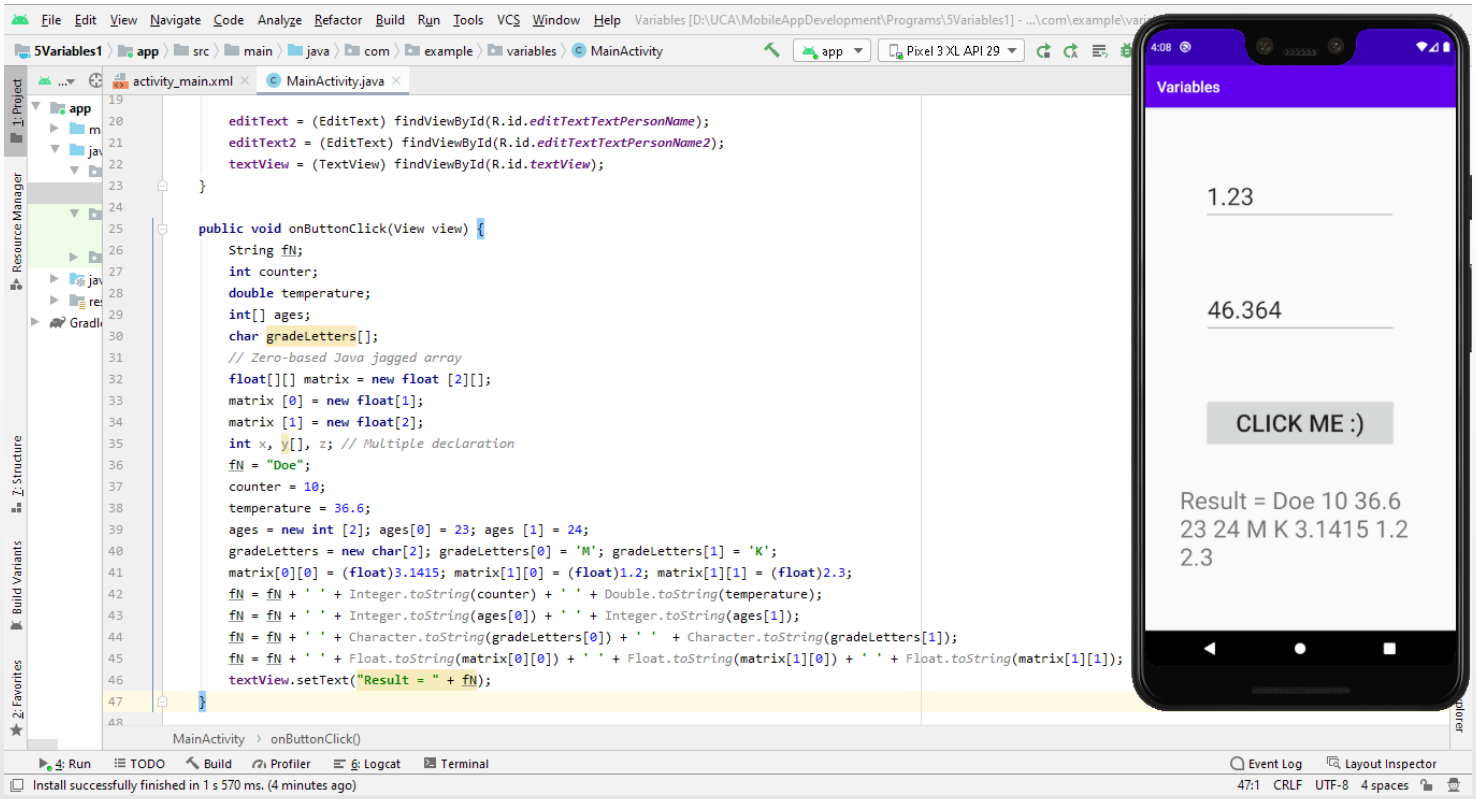
Java is a case-sensitive language

Identifiers (cont.)

- Almost any valid identifier can be chosen to name a class, method, or other source code entity. However, some identifiers are reserved for special purposes; they are known as ***reserved words***: abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, enum, else, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, void, volatile, and while.

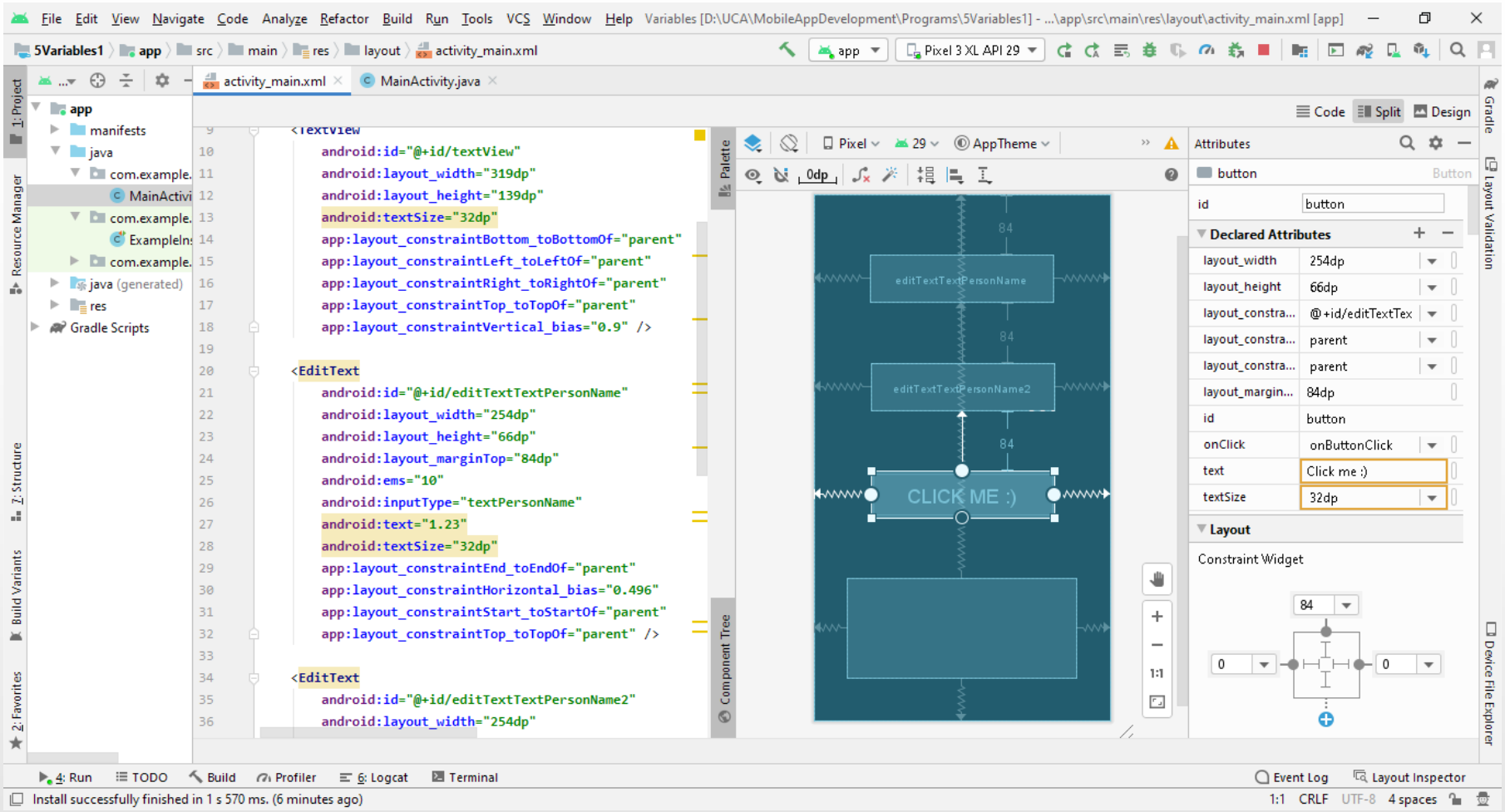
Variables

- A **variable** is a named memory location that stores some type of value. A variable that stores a reference is often referred to as reference variable (reference types and value types)



Variables (cont.)

- Layout:



- Java types
- Compound expressions in Java
- Bitwise operators in Java

Do you have any questions or comments?

Thank you
for your attention !

In this presentation:

- Some icons were downloaded from flaticon.com and iconscout.com