Tutorial 02

Introduction to Android – 02

SEG3125 – Analysis and Design of User Interfaces – Summer 2018

Presented by: Amir Eaman– TA

Class Plan

- Layouts
- Widgets
- Live Demo (Calculator)

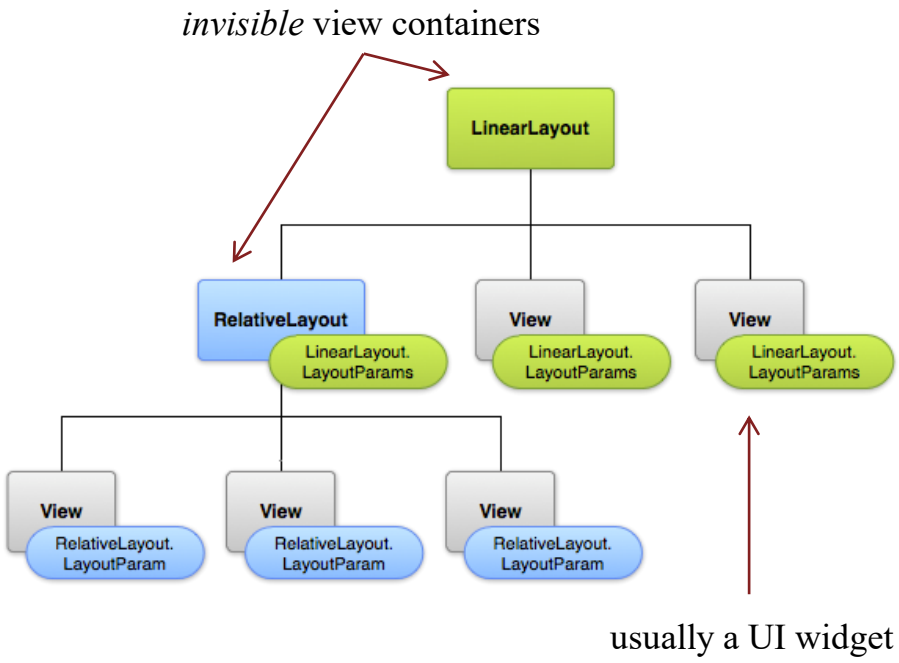
USER INTERFACE: LAYOUTS

What does android provide?

- Pre-build UI components such as structured layout objects and UI controls
- Other UI modules such as dialogs, notifications, and menus (action bars)
- A mechanism to declare elements (e.g. layouts) through either source code or an XML file
  - Easy and quick to create and manage UIs
  - Separation of logic from presentation
  - Reusable
  - can easily specify different layouts for different screen sizes (and densities, languages,...)
  - Easy to understand

User Interface Elements' Hierarchy

- View
  - basic building block for user interface components displayed on the screen.
    - E.g. Button,TextView
  - Occupies a rectangular area on the screen and is responsible for drawing and event handling.
- ViewGroup
  - Functional elements used to organize the UI.
  - The view group is the base class for layouts and views containers
  - Generally contain other views (called children.)
    - E.g.: Relative Layout



Width and Height

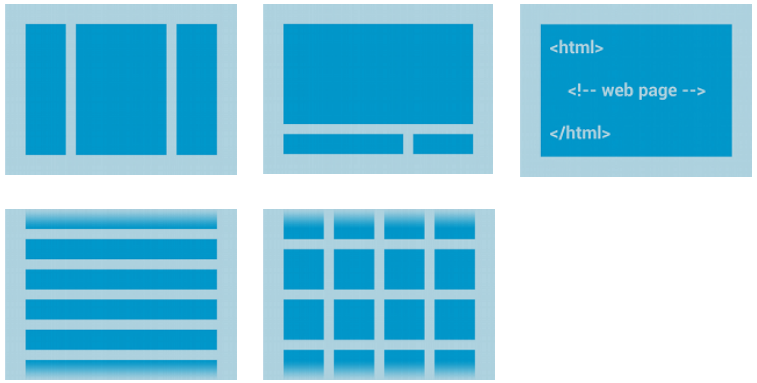
- The Width and Height of views and layouts should be set with fixed dimension values (e.g.: pixels) or with relative constants:

- **MATCH_PARENT** means that the view wants to be as big as its parent, minus the parent's padding.

- **WRAP_CONTENT** means that the view wants to be just large enough to fit its own internal content, taking its own padding into account.

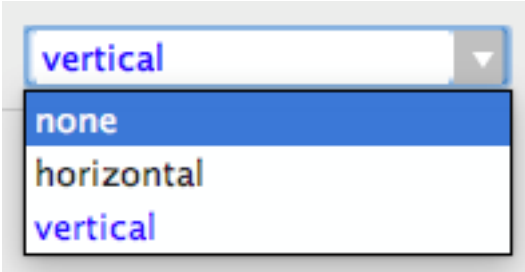
Layouts

- Define the visual structure for a user interface
- Can be declared in either an XML file or source code (runtime)
- Common Layouts
  - Linear Layout
  - Relative Layout
  - ListView
  - GridView
  - WebView



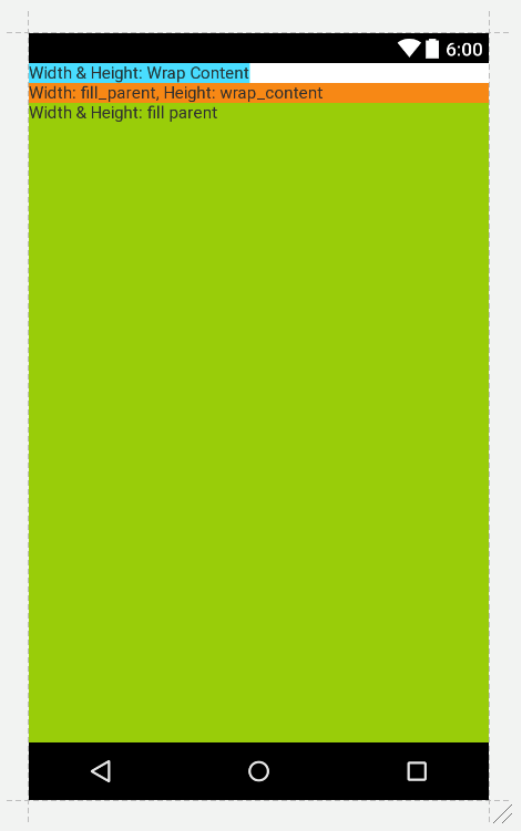
Linear Layout

- **LinearLayout** is a view group implementation that aligns all children in a single direction, either **vertically** or **horizontally**.
- We need to specify the layout direction with the ***android:orientation*** attribute.
- Content is placed sequentially and items take as much space as requested.
  - ***Note****: Items listed in linear lists can be "smooshed" if no space is available on screen.*

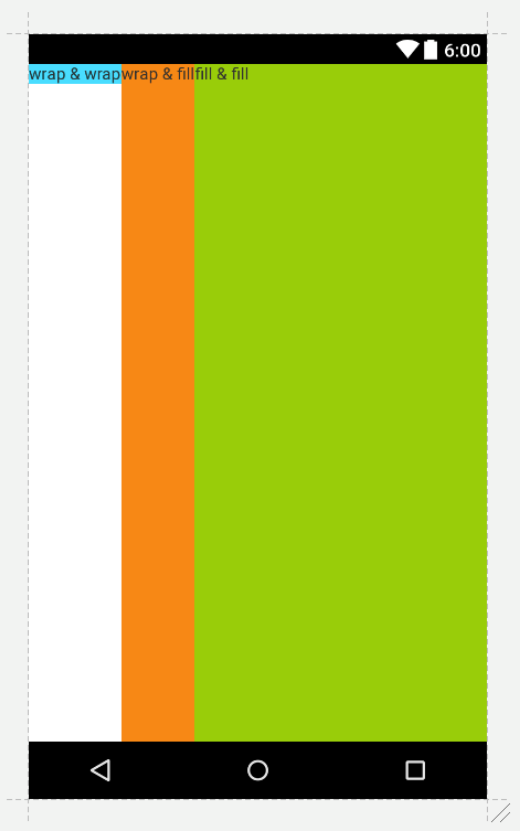


Linear Layouts

Vertical Layout

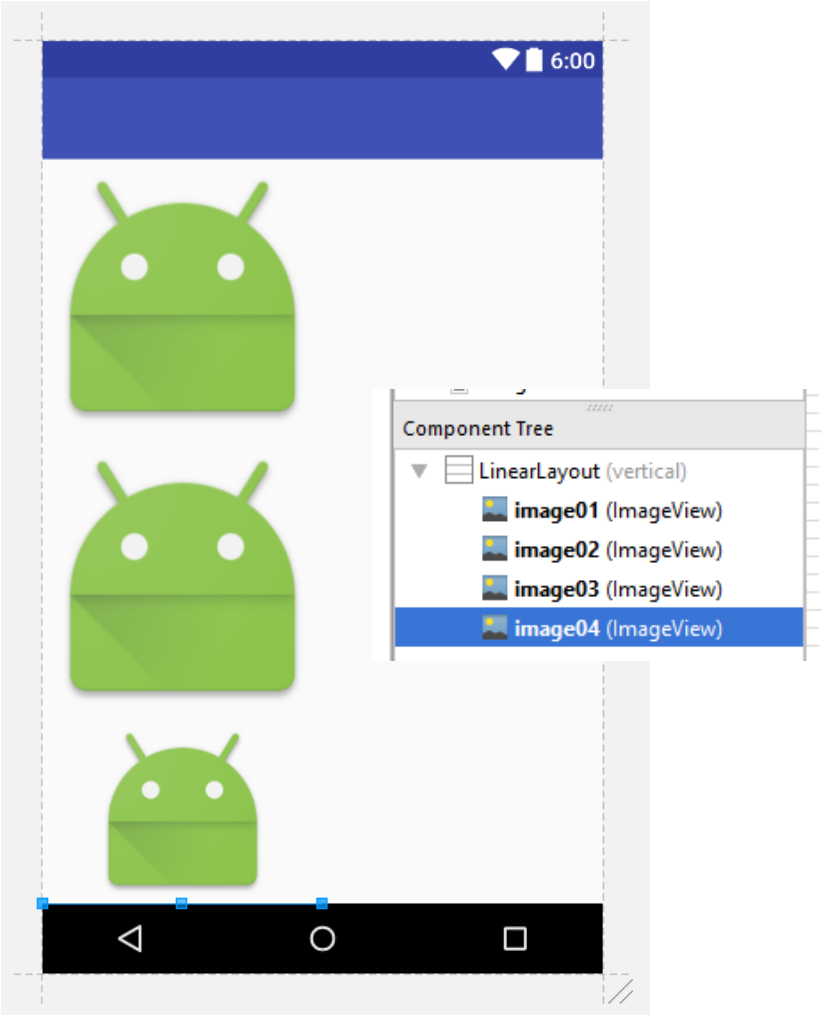


Horizontal Layout



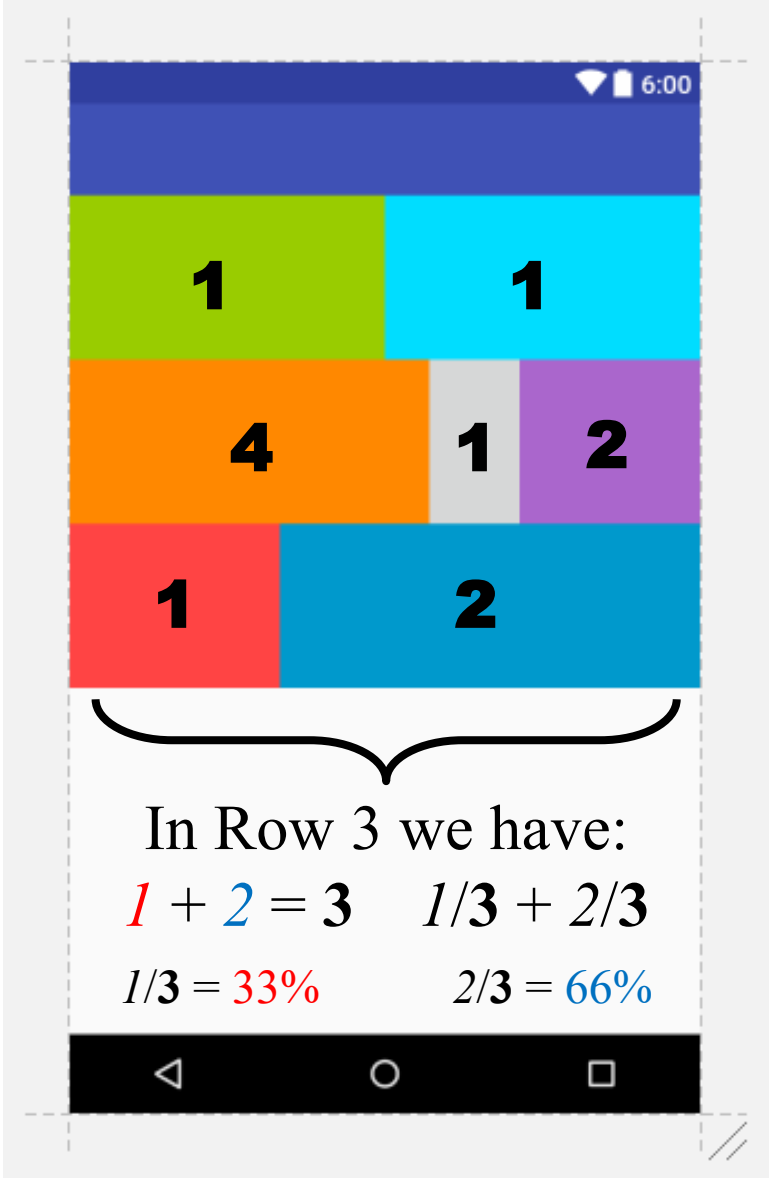
Linear Layout

- The dimensions of an object are affected by the available space on screen.
- In the example displayed, all ***ImageView*** elements have the same height and width property value: ***wrap_content***.
- Limited space affect the dimensions of images 3 & 4.
  - ***Note****: Image 4 is not visible at all. By selecting it in the component tree, a blue line is displayed in the design view.*



Layout Weight

- The ***layout_weight*** attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen.
- A larger weight value allows a view element to expand and fill any remaining space in the parent view.
- When dividing the available space using weights, the design guidelines suggest setting the corresponding dimension value to **0dp**.
  - E.g.: to distribute width using weights, set ***layout_width*** to *0dp*. (see image to the side)



Layout Gravity

- **Gravity** positions the view relative to its parent in both X and Y Axes.
- The **parent** view must be bigger than the **child** for Gravity have visible effect on the view's position.
- Layout Gravity attributes affect **horizontal** and **vertical** orientations differently

- **Horizontal** LinearLayout:
  - **Top**, **Center**, **Bottom**
- **Vertical** LinearLayout:
  - **Left**, **Center**, **Right**, **Start**, **End**
- **Clip**:
  - Defines whether the edge is clipped to its container
- **Fill**:
  - Defines Whether the size of the object should grow to completely fill its container

Relative Layout

- **RelativeLayout** is a view group that displays child views in relative positions (specified by ID).

- The position of each view can be specified as:
  - Relative to sibling elements, such as to the left-of or below another view.
  - Relative to the parent **RelativeLayout** area, such as aligned to the bottom, left or center.

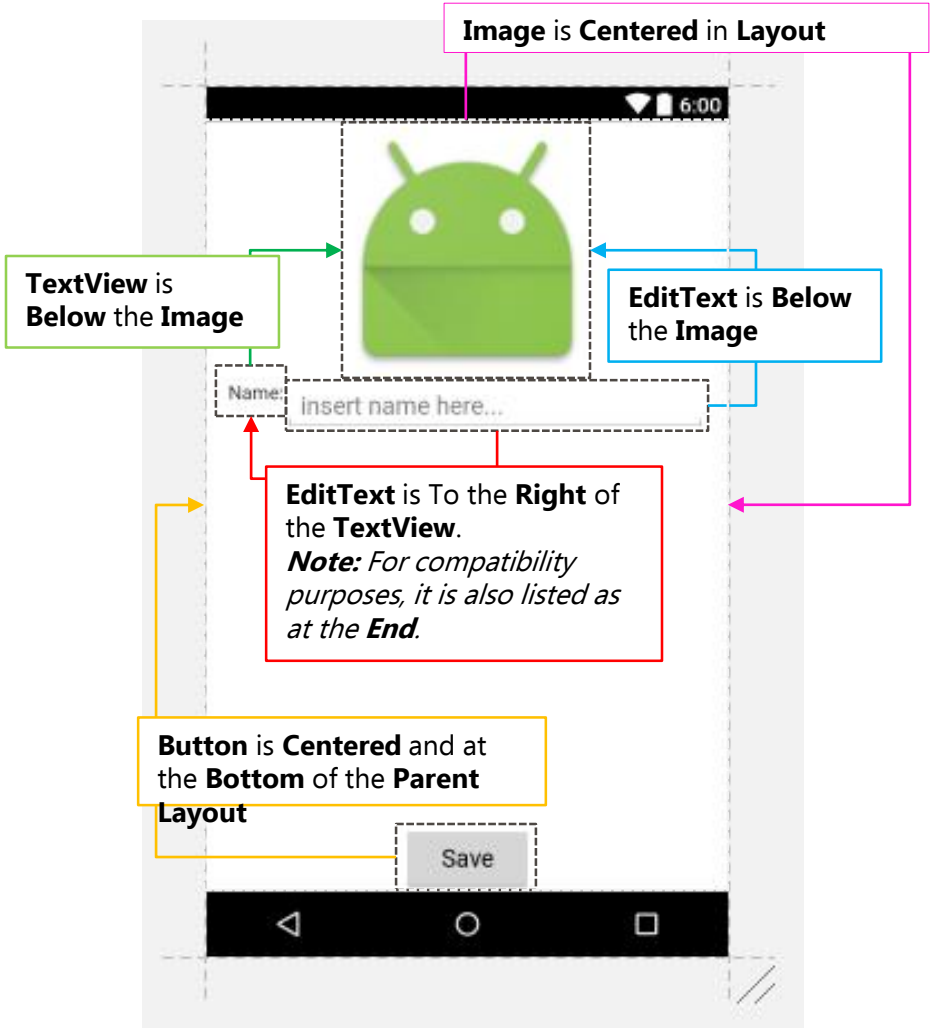
Relative layout

- Parent Relative Alignment (Boolean):
  - Layout_alignParentLeft
  - Layout_alignParentRight
  - Layout_alignParentBottom
  - Layout_alignParentStart
  - Layout_alignParentEnd

- Sibling Relative Alignment (Sibling ID):
  - Layout_above
  - Layout_below
  - Layout_toLeftOf
  - Layout_toRightOf
  - Layout_toStartOf
  - Layout_toEndOf

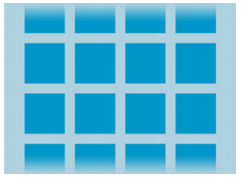
Additional Layout Properties

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/ic_android_large"
    android:id="@+id/avatarImage"
    android:layout_centerHorizontal="true"
    android:contentDescription="User Avatar" />

<TextView
    android:text="Name:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/nameTextView"
    android:layout_below="@+id/avatarImage"/>

<EditText
    android:id="@+id/nameField"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="insert name here..."
    android:layout_below="@+id/avatarImage"
    android:layout_toRightOf="@+id/nameTextView"
    android:layout_toEndOf="@+id/nameTextView"/>

<Button
    android:layout_width="96dp"
    android:layout_height="wrap_content"
    android:text="Save"
    android:layout_centerInParent="true"
    android:layout_alignParentBottom="true"/>
```



Space between children may be specified either by using instances of the dedicated Space
Space between children may be specified either by using instances of the dedicated Space
Space between children may be specified either by using instances of the dedicated Space

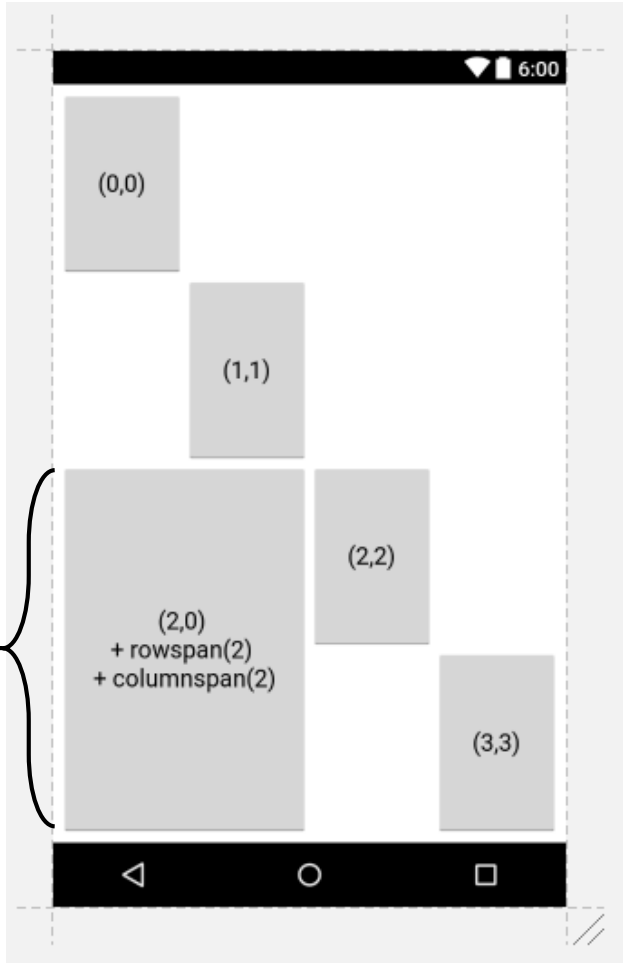
Grid layout



- A Grid layout places its children in a rectangular *grid* composed of a set of infinitely thin lines that separate the viewing area into *cells*.
- The **size** of the grid is specified using the **rowCount** and **columnCount** properties to specify number of rows and columns.
- Views are placed by specifying the desired row and column values, counting from 0.
- A view can takes more than 1 row and column,
- ***Note:*** *Space between children can be specified using either Space views or the margin layout parameters.*

- ***Note:*** *As of API-LVL-21, Weight properties can be used in* ***GridLayouts*** *with* ***layout_columnWeight*** *and* ***layout_rowWeight*** *being available, following similar logic to standard weight property.*
- *Prior to this version, it was recommended that nested Horizontal and Vertical Layouts be used.*

```
<Button
android:text="(2,0) \n+ rowspan(2) \n+ columnspan(2)"
android:layout_width="0dp"
android:layout_height="0dp"
android:layout_columnWeight="1"
android:layout_rowWeight="1"
android:id="@+id/button5"
android:layout_row="2"
android:layout_column="0"
android:layout_rowSpan="2"
android:layout_columnSpan="2"
/>
```



Other layouts

Frame Layout

- **FrameLayout** is designed to block out an area on the screen to display a single item.
- Generally used to display only one view, or views which overlap.
- Multiple children can be added to a single **FrameLayout**. Their position is controlled by assigning gravity to each child, using the **layout_gravity** attribute.

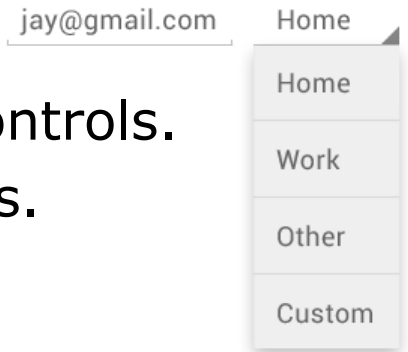
Table Layout

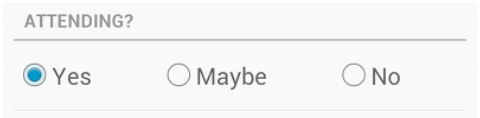
- **TableLayout** let you arranges components in rows and columns, just like the standard table layout in HTML, <tr> and <td>.
- A TableLayout consists of a number of TableRow objects, each defining a row.
- As content is added following the HTML style, rows have to be added in sequence.

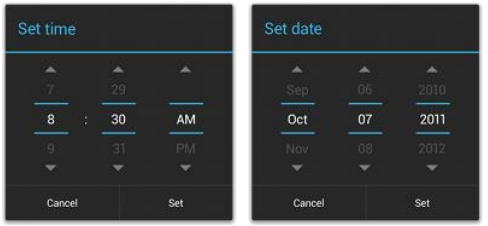
USER INTERFACE: WIDGETS

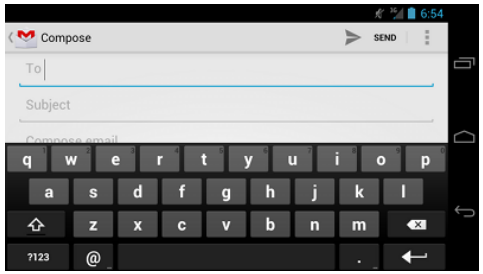
Input Controls

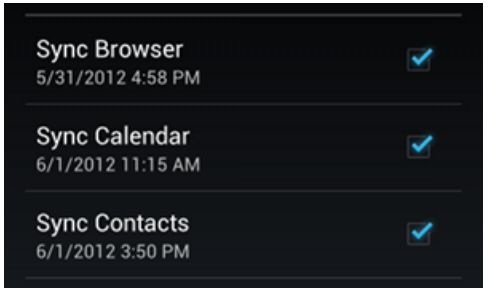
- Android provides varied pre-built inputs controls.
- You can build you own custom components.
- Common input controls:
  - **Button**
  - **Text field**
  - Checkbox
  - Radio button
  - Toggle button
  - Spinner
  - Pickers











Responding to button clicks

Method 1: (reference in xml file)

- In the layout *XML* file, set the **onClick** property:
  - *android:onClick="<name of method in the java code>"*
- Then, implement the method in the corresponding activity *.java* file.

Method 2: (implementing onClickListener)

- In the *.java* file, implement the ***onClickListener*** method for the desired button:

```
final Button button = (Button) findViewById(R.id.button);
button.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
      //<HERE GOES YOUR BUTTON FUNCTIONALITY>
    }
});
```

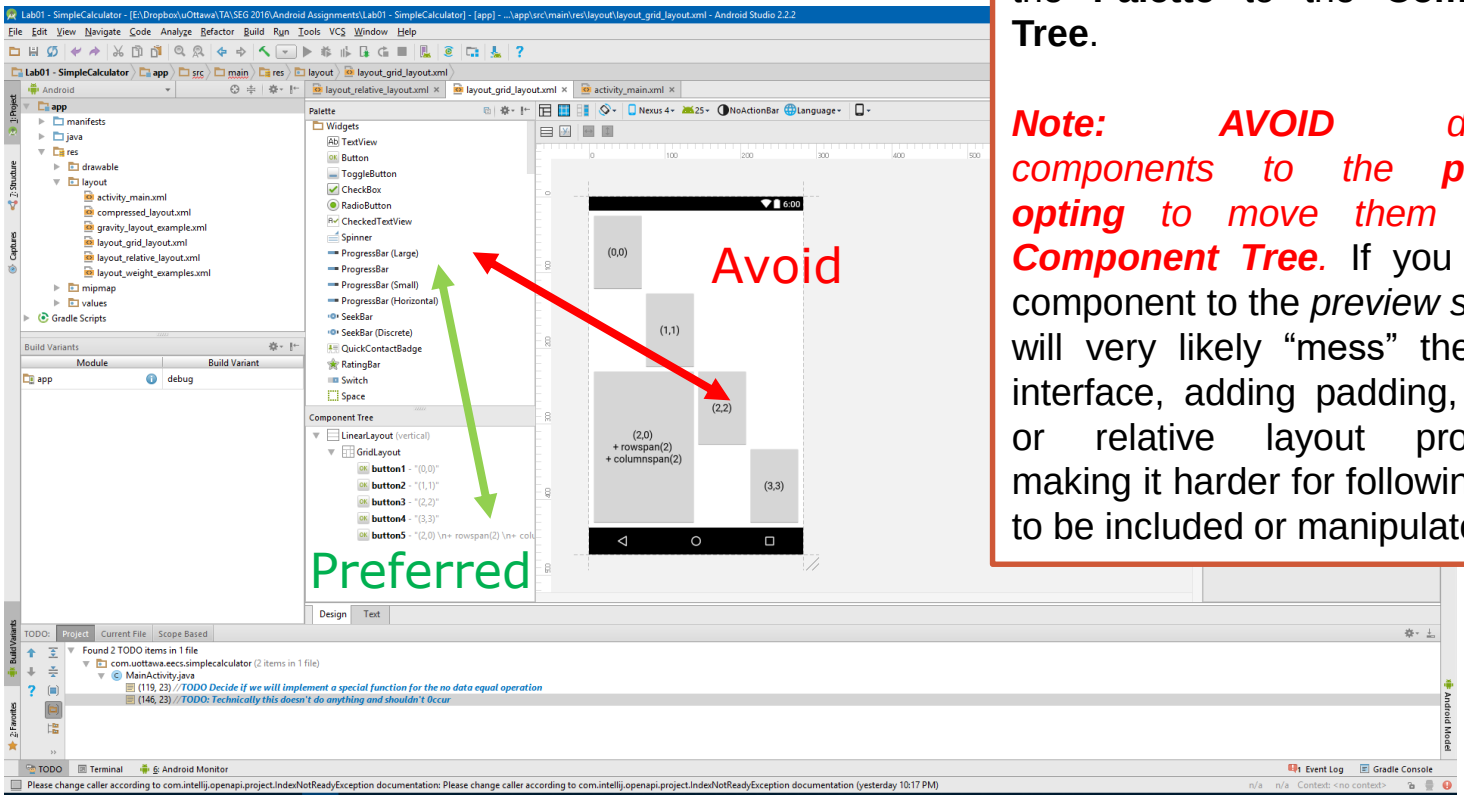
Toasts

- Simple feedback about an operation in a small popup
- They automatically disappear after a timeout
- You can position the toast differently or even use your own layout instead of a simple text message.

Dialogs

- Dialogs prompt the user for decisions or additional information required by the app to continue a task.
- Different dialogs
  - AlertDialog
    - You can also add lists
  - DatePickerDialog
  - TimePickerDialog
  - Custom Layout

Working with your UI



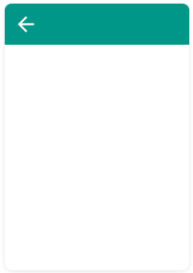
To add a new object to your interface, drag the component from the **Palette** to the **Component Tree**.

Note: ***AVOID*** *dragging components to the* ***preview****,* ***opting*** *to move them to the* ***Component Tree****.* If you drag a component to the *preview screen* it will very likely "mess" the whole interface, adding padding, anchor or relative layout properties, making it harder for following items to be included or manipulated.

Android Studio

CALCULATOR (LIVE DEMO)

Simple Calculator : Step 01



1. Follow the steps in the first tutorial to create an application. Name your application mnemonically (e.g.: "Calculator") and select the "Empty Activity" layout.

2. Remove the TextView "***Hello World***" *(if your layout is not empty)*

3. Add a **TextView** to the *MainLayout* and change its id to "***resultText***" or something meaningful.
    1. *To edit a property in an object, select it in the preview panel or component tree and change the desired property in the properties panel.*

4. Add a (Horizontal) Layout under "***resultText***" and assign Id "***buttonRow01***". *This layout will have the first row of buttons*

Note: Only use other layout options (such as the GridLayout) if you are confident *in your skills.*

Simple Calculator : Button Rows

1. Add four buttons to the ***HorizontalLayout*** you just created.

2. Rename each button *Id* to the appropriate "***btn0X***" value.
    1. *(e.g.: Button 1 will be called "btn01" and the Equals button "btnEQ")*

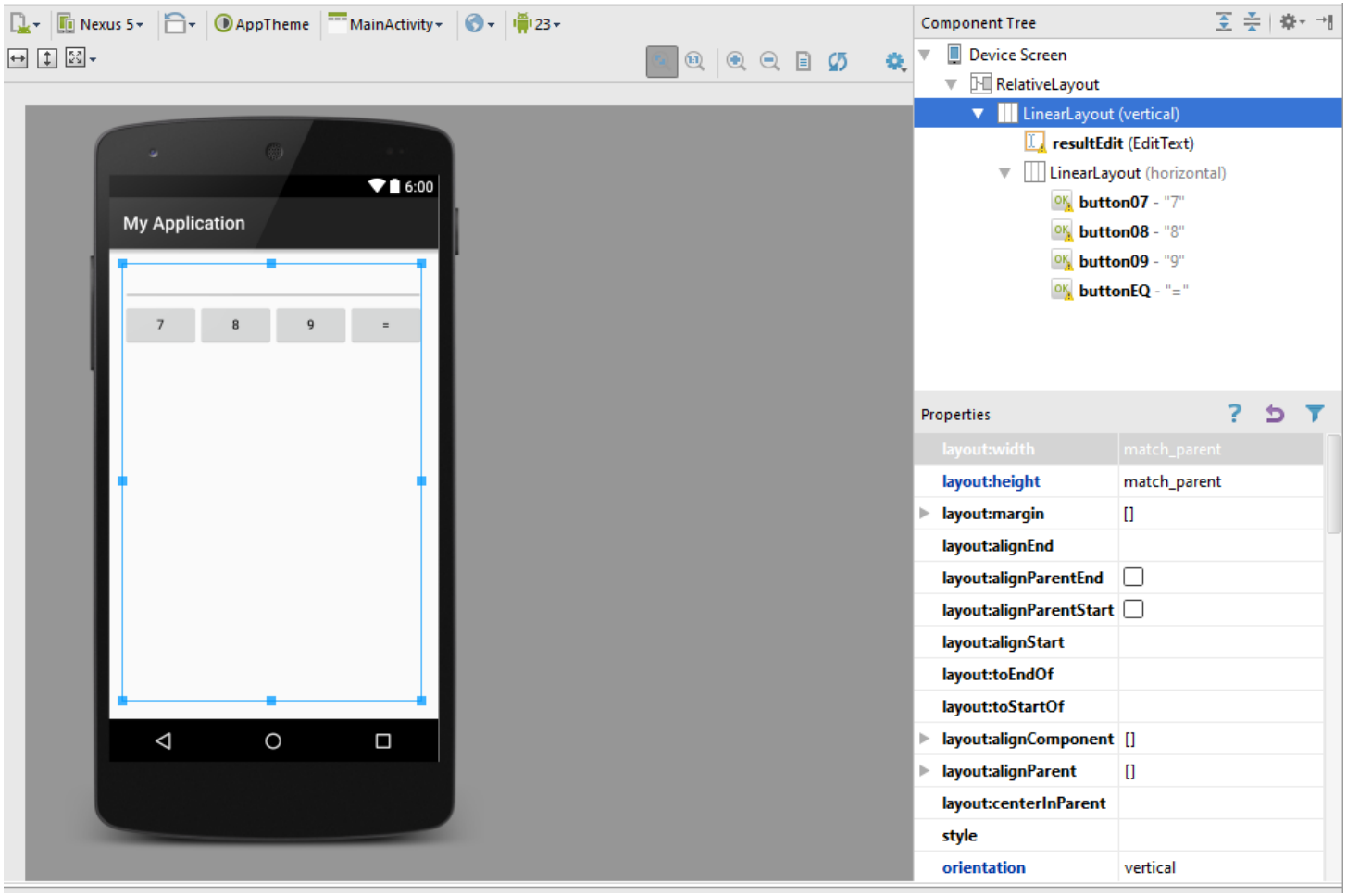
At this point your buttons might look unbalanced, with the later ones added being flattened on the right side of the screen. That is because the other buttons are taking all the space available. To fix this we must change the properties of the views (layout weight, width, height).

1. Change the ***layout weight*** property in each button to 1.

3. *Change the* **layout width** of the buttons to ***0dp***.

4. *Now all buttons should have the same space on the screen regardless of the text in them.*

Simple Calculator : Current Status



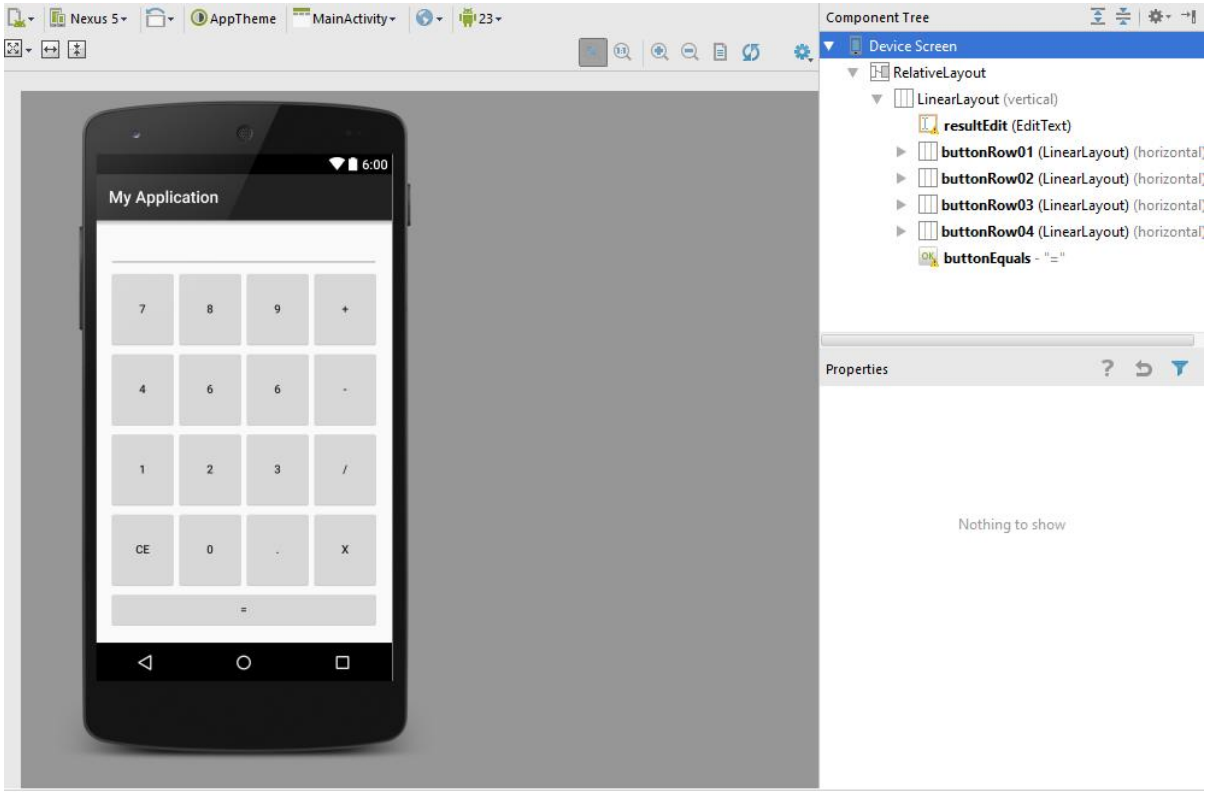
Simple Calculator : More Buttons

1. Repeat the steps in the previous slide 3 times to create a 4 by 4 button matrix.

2. You will notice that the later rows of buttons get flattened. Update their ***layout_height*** accordingly.

2. Add an extra button bellow the lowest button row (HorizontalLinearLayout). This will be the "Equal Sign"
   2. Set the ***layout:width*** to ***fill_parent.***

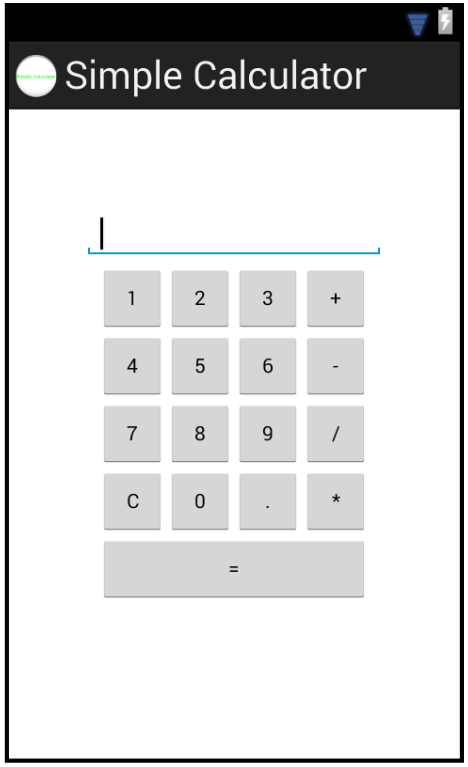
Simple Calculator : Step 03 Result



The final design of your personal project does not have to match the example exactly so long as you include all basic functions.

Simple Calculator : Buttons

- Change the text of all buttons so the application resembles a calculator.
- Change the id of all buttons as following:
  - The numbers' id should be started with "btn0"+its number. For example, for 1 button's id should be "btn01", for 0 should be "btn00", and so on.
  - For others see the table to the side.



Symbol	ID
"C"	btnClear
"."	btnDot
"+"	btnAdd
"-"	btnMinus
"*"	btnMultiply
"/"	btnDivide
"="	btnResult

Simple Calculator : Strings

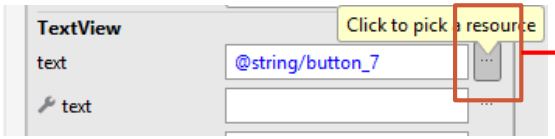
In real-world scenario, applications need to be flexible allowing for localization. To allow this we must decouple the string text in our elements from the layout files.

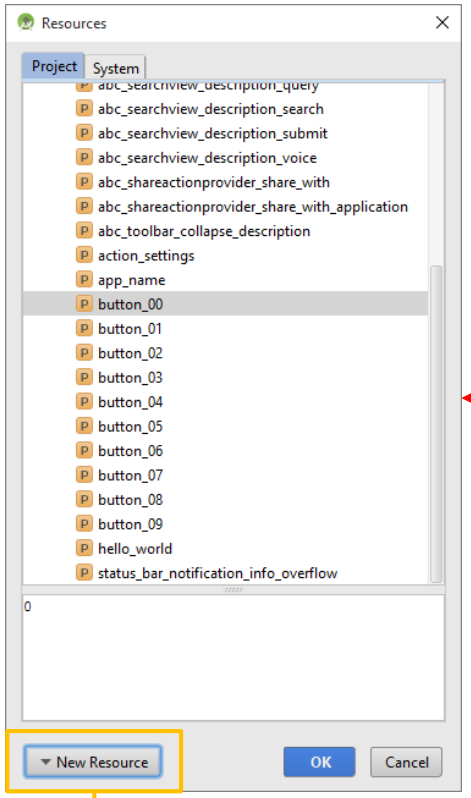
To separate strings from the layout code, follow these steps:

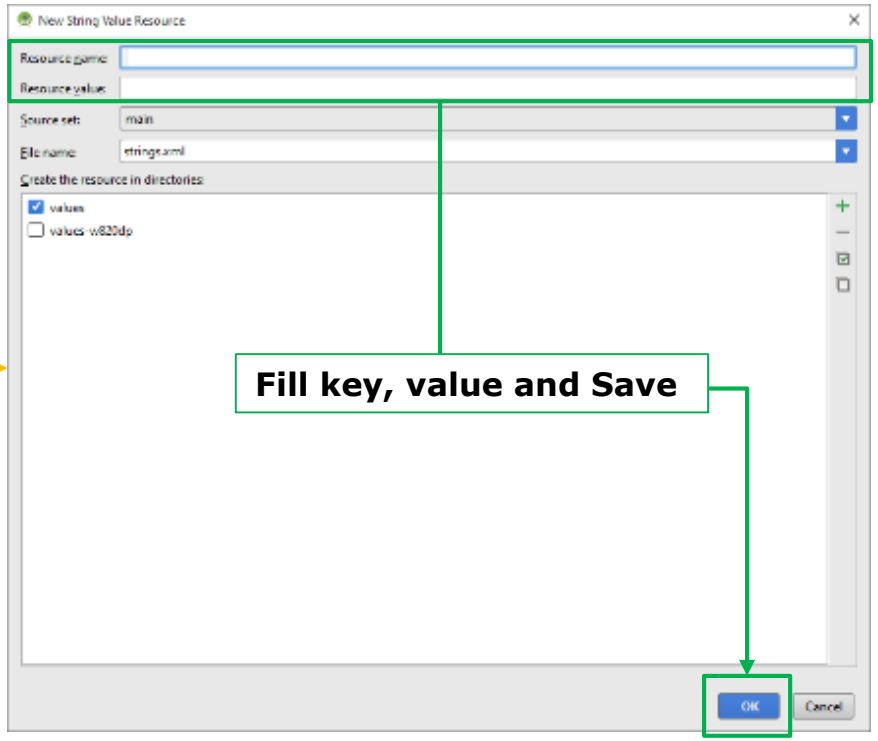
1. Select a button in **component tree**.
2. Navigate to the **"Text" property** in the properties panel.
3. Select the text property and click the triple dots button.
4. Select **"New Resource"** and then **"New String Resource"** in the Resources Window.
5. Define a name for the new button name (e.g.: button_07), add the actual button string value (e.g.: "7"), save the new resource.
6. Repeat this process for all numbers and operands.

1. Alternatively, open the strings.xml file and include the texts manually.

Creating a new String Value







Simple Calculator (cont.)

Now write code for each button to your ACTIVITY JAVA FILE! The next slides have samples code

- Add all onClick methods to MainActivity.java file.
- The methods' signature should be as following:

```
Public void <<methodName>>(View view) {
        //<METHOD CODE GOES HERE>>
}
```

- **E.g.:** onClick method for equal button (btnResult):

```
Public void btnResultClick (View view) {
        //<YOUR CODE GOES HERE>
}
```

- ***Note:*** *Don't forget to import* ***"android.view.View"*** *as well as other necessary classes (Button, TextView, etc).*
- ***Note:*** *Copy-pasting code can cause formatting errors. Pay close attention!*

Simple Calculator: Code (01)

```
package com.uottawa.eecs.simplecalculator;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private enum Operator {none, add, sub, mul, div, eq}
    private double data01=0;
    private Operator opp = Operator.none;
    private boolean hasDot = false;
    private boolean requiresCleaning = false;

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

Simple Calculator: Code (02)

```
public void btn00Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "0");
}

public void btn01Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "1");
}

public void btn02Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "2");
}

public void btn03Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "3");
}

public void btn04Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "4");
}
```

Simple Calculator: Code (03)

```
public void btn05Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "5");
}

public void btn06Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "6");
}

public void btn07Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "7");
}

public void btn08Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "8");
}

public void btn09Click(View view) {
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + "9");
}
```

Simple Calculator: Code (04)

```
public void btnAddClick(View view) {     //Sum
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    opp = Operator.add;
    data01 = Double.parseDouble(resultTextView.getText().toString());
    resultTextView.setText("");
}

public void btnSubClick(View view) {     //Subtraction
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    opp = Operator.sub;
    data01 = Double.parseDouble(resultTextView.getText().toString());
    resultTextView.setText("");
}

public void btnMulClick(View view) {     //Multiplication
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    opp = Operator.mul;
    data01 = Double.parseDouble(resultTextView.getText().toString());
    resultTextView.setText("");
}

public void btnDivClick(View view) {     //Division
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    opp = Operator.div;
    data01 = Double.parseDouble(resultTextView.getText().toString());
    resultTextView.setText("");
}
```

Simple Calculator: Code (05)

```
public void btnClearClick(View view) {    //Clear/Reset
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    opp = Operator.none;
    resultTextView.setText("");
    data01 = 0;
    hasDot = false;
}

public void btnDotClick(View view) {    //Fraction (dot)
    if (hasDot) return;     //We only add a fraction marker if we don't have one already
    TextView resultTextView = (TextView) findViewById(R.id.resultText);
    resultTextView.setText(resultTextView.getText() + ".");
    hasDot = true;
}
```

Simple Calculator: Code (05)

```
public void btnResultClick(View view) {
    if (opp != Operator.none) {
        TextView resultTextView = (TextView) findViewById(R.id.resultText);
        double data02 = Double.parseDouble(resultTextView.getText().toString());
        double result = 0;
        switch (opp) {
            case add:
                result = data01 + data02;
                break;
            case sub:
                result = data01 - data02;
                break;
            case mul:
                result = data01 * data02;
                break;
            case div:
                result = data01 / data02;
                break;
            default:
                //Nothing is done (edge case)
                break;
        }
        if ( (result - (int)result) != 0) {
            resultTextView.setText(String.valueOf(result));
            hasDot = true;  //Result is already a fraction, no "dot" button functionality allowed
        } else {
            resultTextView.setText(String.valueOf((int)result));
            hasDot = false;  //Result is not a fraction and therefore we can use the "dot" button
        }
    }
}
```

Simple Calculator : Input Events

- Each view has its own Events (e.g. onTouchEvenet()) and must be override in order to have your own implementation.
- Instead of extending a view to have control events, use even lisetners
- An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework.
  - onClick()
  - onLongClick()
  - onFocusChange()
  - onKey()
  - onTouch()
  - onCreateContextMenu()

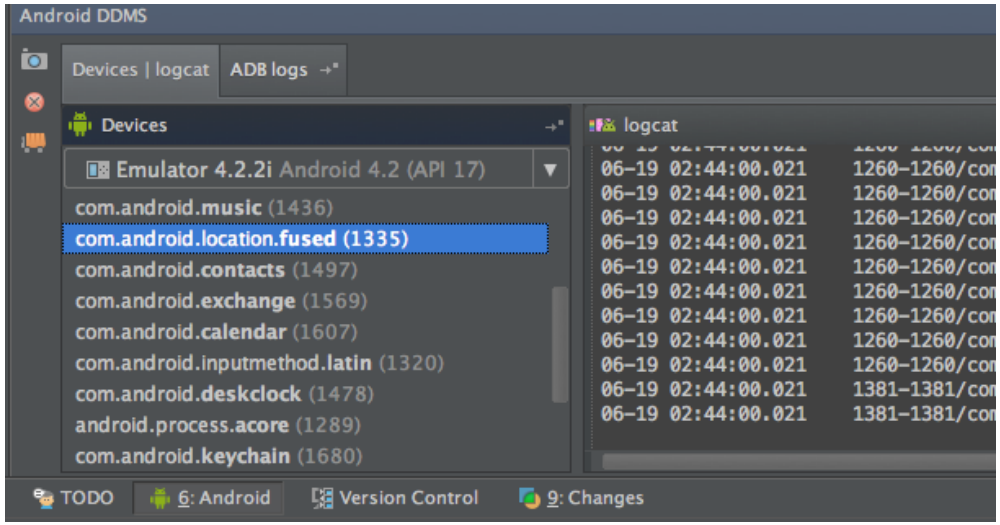
Simple Calculator : Button Functions

In order for your buttons to run the code in the java file they need to know what function they should call. To do so:

- Add ***onClick*** property to all buttons
  - In the "Design" view Set onClick property in the Properties panel to the corresponding function name in the code.
    - E.g.: the function ***btn01click()*** corresponds to **button01**, so the **OnClick** property should read **"btn01click".**
  - If you check the text version of the interface, you will see something similar to: *android:onClick="btn01Click"*
    - *This means that button01 has the correct onclick method assigned to it. Methods that have not been used are colored in grey.*
- Repeat this process for all numerical and operator buttons!

Logcat

- Tool used for debugging purposes.
- Print messages with different tags.
- Displays Error Stack traces if exceptions occur.



Note: HELP! My code isn't working

- Squiggly lines means your expression can't compile.
- It is very likely that there is a ***Typo*** in the code.
  - **onClick()** is different from **onclick()** - (uppercase)
  - **button01** is different from **Button01** and **button_01**
- Remember to include the widgets you use!
  - Text in RED means that Android Studio doesn't know what class you want, probably because a missing include.
  - Alt+Enter on missing dependencies auto-imports the class it thinks you need.
- Yellow and Red lamps give you tips about what to do.

Questions?