

# Android: Session One

## Prerequisite

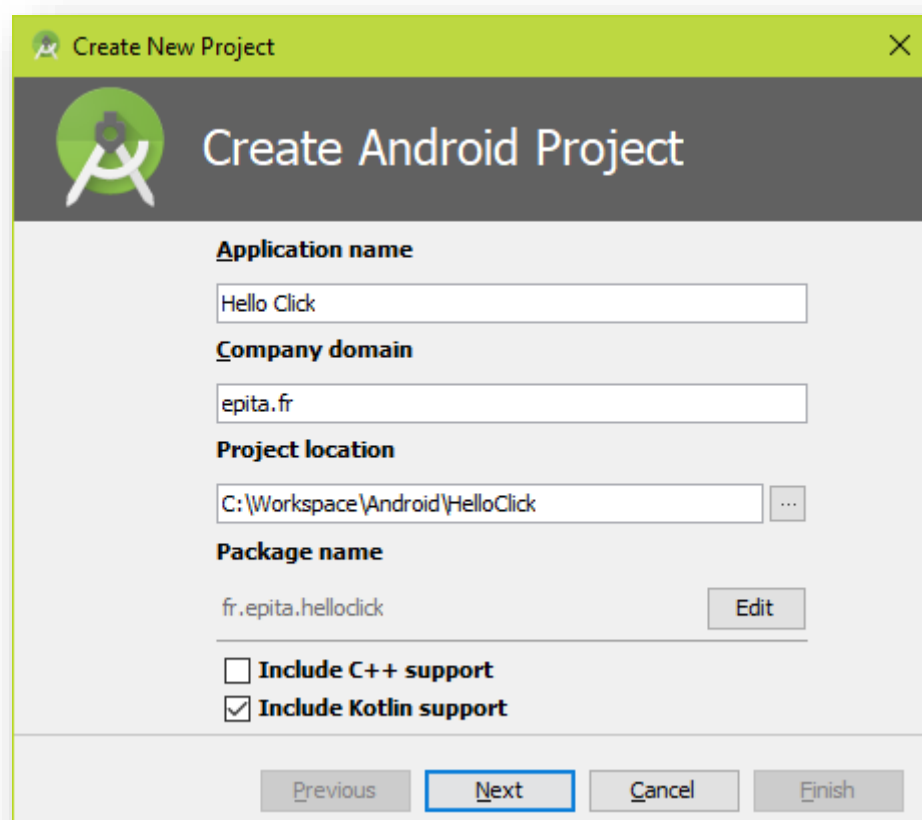
You need to have the complete development environment installed and configured before dosing this practical session. See *Android course preliminaries.pdf* document.

## Part 1: Hello Click

We are going to make a simple HelloWorld application that can also count how many times you click on a button (such an amazing functionality). But at first, we are going to focus on the UI.

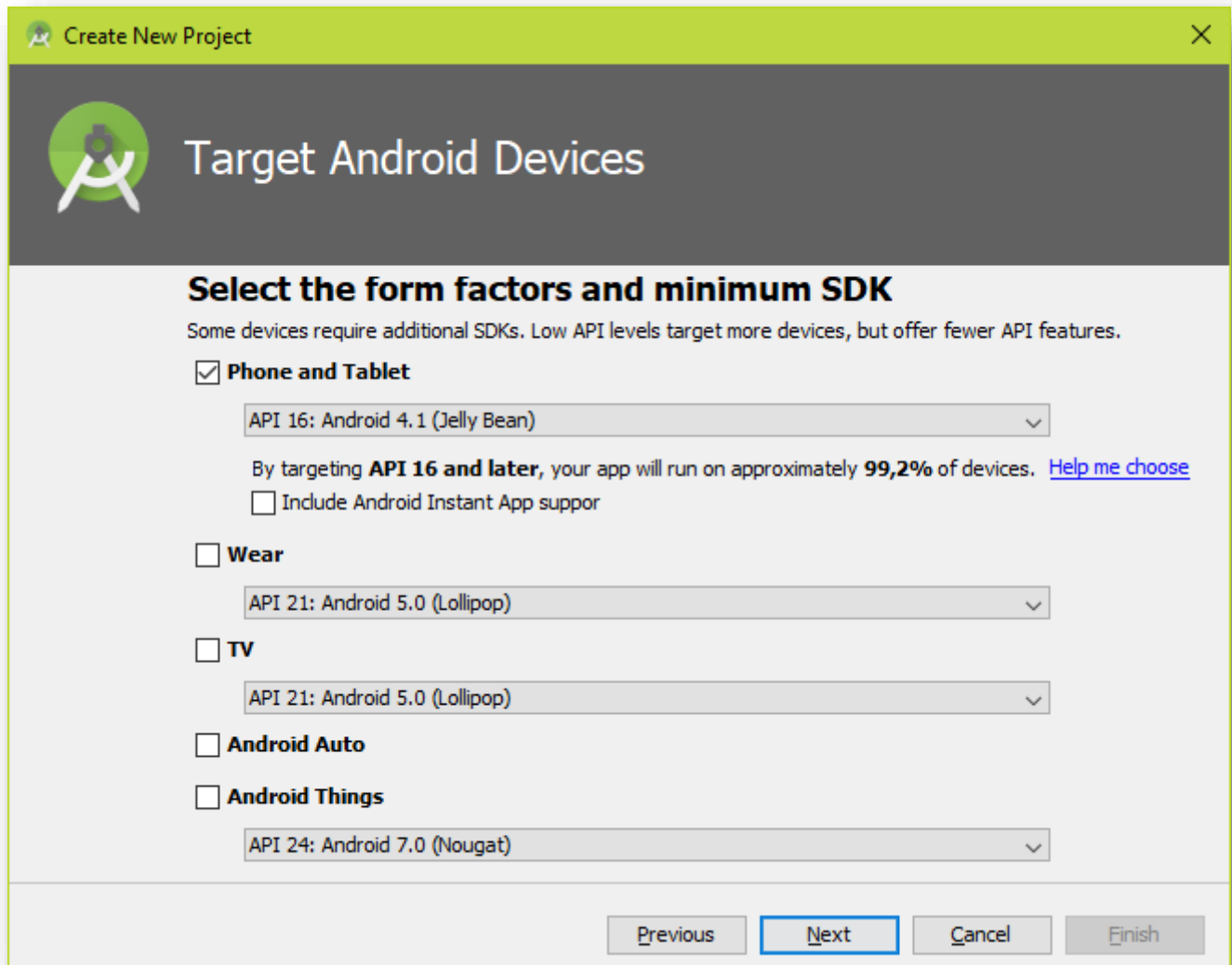
### First step: Create the project

Create a new “Hello Click” android project. Either from Studio starting screen, or if a project is already opened in menu **[File -> New -> New Project]**.



- Application name can be whatever you want, but obviously, it's better if the name fits the app.

- Company domain is used to create a package name. Use **epita.fr**.  
The package name is used as the unique identifier for this app (on the phone, or on the store).  
Package name will also be used as a default package name for our code.
- Project location can be changed
- Do not forget to check the **Include Kotlin** checkbox.



**Create New Project**

**Target Android Devices**

**Select the form factors and minimum SDK**  
Some devices require additional SDKs. Low API levels target more devices, but offer fewer API features.

☒ **Phone and Tablet**

API 16: Android 4.1 (Jelly Bean)

By targeting **API 16 and later**, your app will run on approximately **99,2%** of devices. [Help me choose](#)

☐ Include Android Instant App support

☐ **Wear**

API 21: Android 5.0 (Lollipop)

☐ **TV**

API 21: Android 5.0 (Lollipop)

☐ **Android Auto**

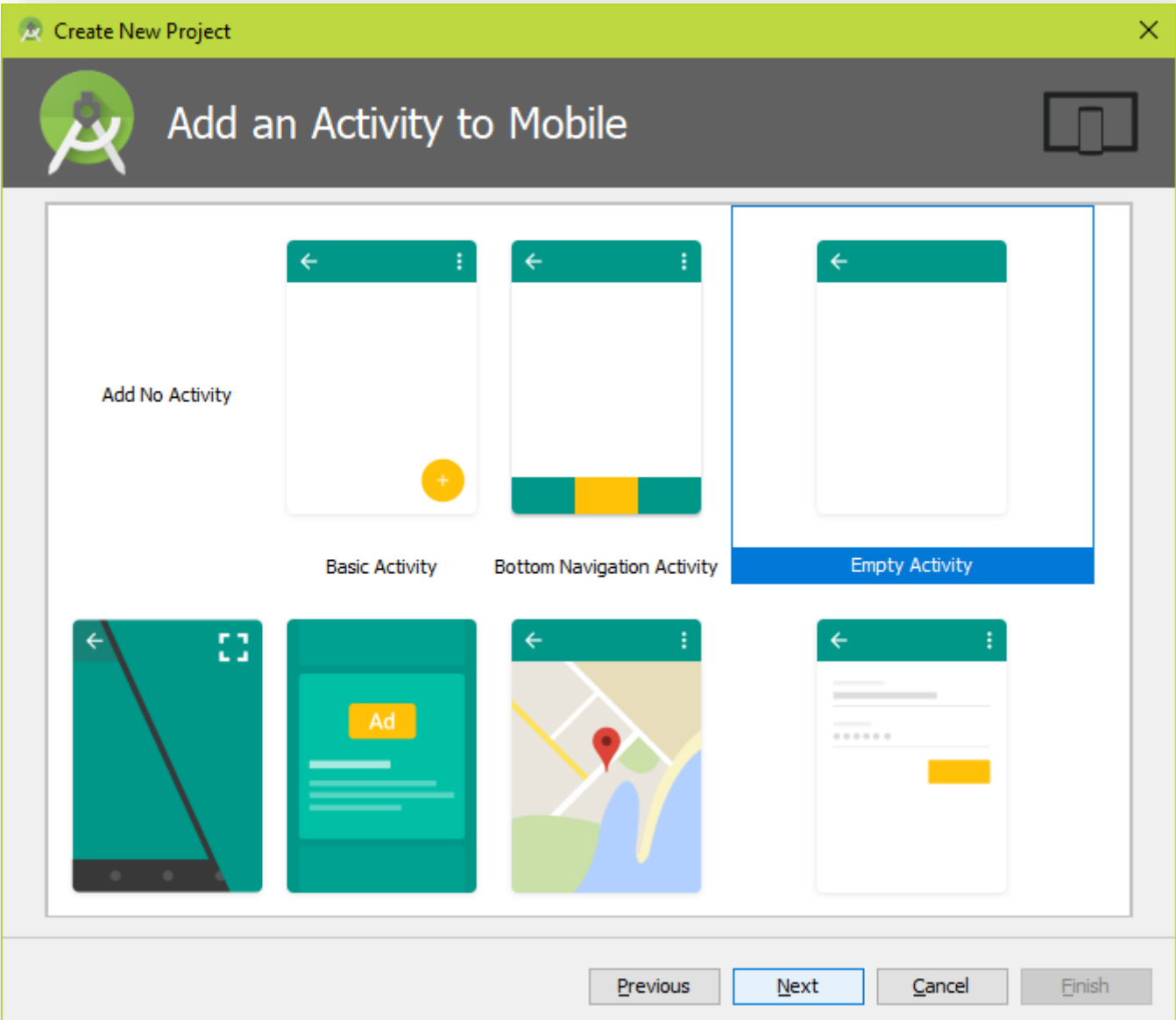
☐ **Android Things**

API 24: Android 7.0 (Nougat)

Previous Next Cancel Finish

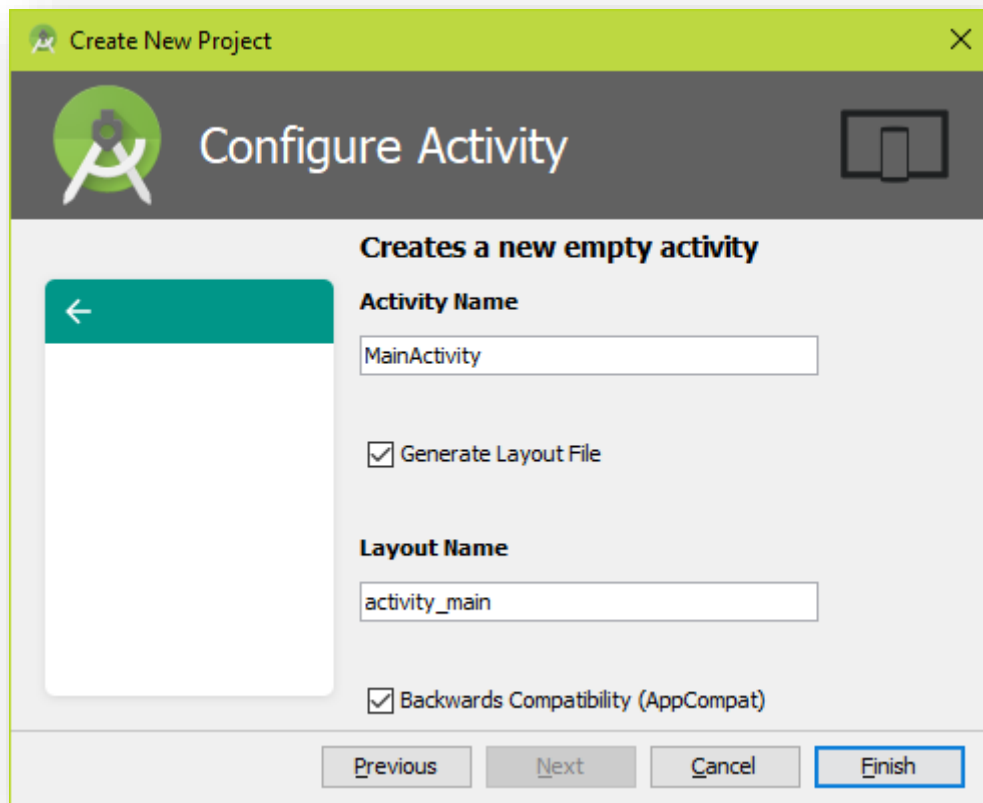
- Create an application for phone and/or tablet, and select android SDK 4.1 (API 16) as a minimum SDK.  
It means our application will only run on devices running Android JellyBean or newer.

This next screen is not actual part of the app creation but is more something of a bootstrap as it enables us to put some initial content in our newborn app. Since our first applications will be quite simple, select the **Empty activity** template.



This will include in our application a first Activity, complete with a layout resource file.

Next screen is designed to customize this new Activity, but default values will do, so do not change anything in here.



- Activity name is, as the name implies, the name for our first Activity.
- Generate layout file will generate for us a layout resource file, and use it as a user interface for this activity
- Layout name is directly related to the previous setting and since we asked for a layout file, we get to name it.
- Be sure to check the AppCompat checkbox, it makes sure our code will run on older Android versions.

Now, click finish to finally create the project.

## Second step: Get acquainted

### Variables

In Kotlin, variables are typed and can be mutable, or read-only respectively declared using the keywords **var** and **val**:

```
var myMutable: Int // mutable variable is type integer
val myReadOnly: String // read-only variable is type string
```

Most of the time, when declaring and assigning a variable at the same time, type can be inferred:

```
var myMutable = 3 // myMutable is inferred as an Int
val myReadOnly = "Hello world" // myReadOnly is inferred as a String
```

### Functions

Functions are defined using the keyword **fun**:

```
// A function returning an integer value
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

```
// A private function that does not return anything. The special return type Unit
// can be omitted
private fun justLog() : Unit {
    Log.d("tag", "A message to log")
}
```

## Classes

A class is defined by a name, which like in Java should start with an uppercase letter, prefixed with the keyword `class`, and of course defined between curly brackets:

```
// This class extends the Gesture class, and implements the OnClickListener interface
// It has one primary constructor with one nullable (?) String parameter
class MyCustomGestureClass(val name: String?) : Gesture(), View.OnClickListener {
    init {
        // Primary constructor code goes here
    }

    constructor() : this("DefaultName") {
        // A Secondary constructor
    }

    override fun onClick(clickedView: View?) {
        // This function overrides the function in the interface OnClickListener
    }
}
```

➡ Take some time to locate your existing Activity ("java/fr.epita.helloclick/MainActivity.kt") code and layout ("res/layout/activity\_main.xml") files:

Our Activity, added by the project creation wizard, is called MainActivity, and it extends the AppCompatActivity from Android framework (which in turn extends the base Activity class).

Classes are packed in packages. This class belongs to the fr.epita.helloclick package.

Other classes can be imported to be used in this one.

```
package fr.epita.helloclick

import android.os.Bundle
import android.support.v7.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

}
```

From its parent, our Activity only override the **onCreate** callback method. This is where we set the layout resource file to use when displaying this activity. This is also where we are going to write most of our activity code.

XML attributes customize the View they are set in.

Some encoding information. XML is text after all...

The root of our document is a ViewGroup: a widget made to organize other widgets. Note the opening and closing tag.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp"
    tools:context="fr.epita.helloclick.MainActivity">

    <TextView
        android:id="@+id/text_hello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

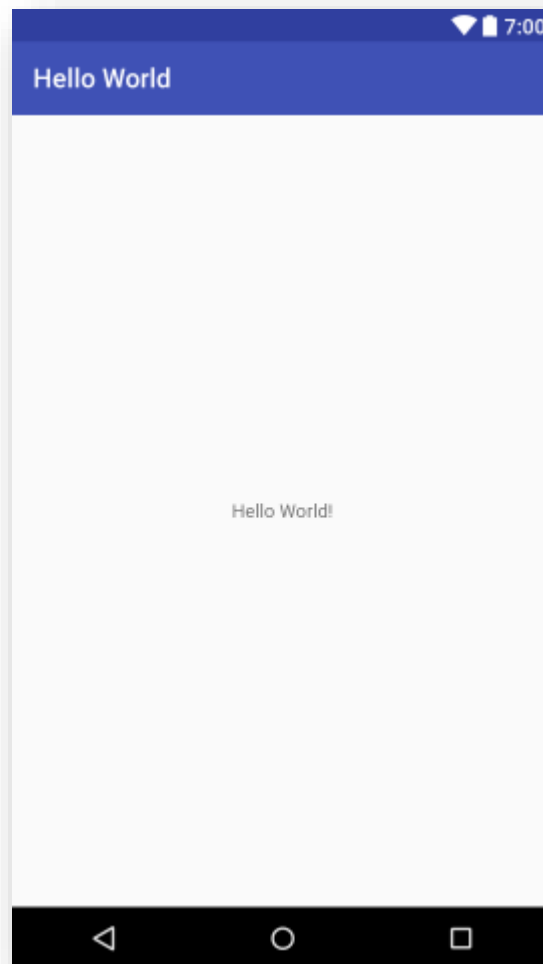
</android.support.constraint.ConstraintLayout>
```

Only one widget can be seen in this layout: a basic TextView to display some hello world text.



## Third step: UI customization and basic views

The UI created by the Blank activity wizard should look like this:



The root of this layout is a `ConstraintLayout` viewgroup (but we will see that later) and the UI comprises only a simple `TextView` widget, with its basic attributes. Also, notice how this textview uses a static string instead of a reference to a string resource.

This is not good. Android uses XML resource files for internationalization (among other things...), that's why you should never use hard coded strings in your activities or layouts, but instead reference a string resource from the `res/values/strings.xml` file.

Inside a layout file (but also from others resource files), you can reference a resource by using the `@<resource_type>/<resource_name>` syntax.

➡ Use examples of widgets below to see how those basics views work.

```
<!-- The TextView displays some text -->
<TextView
    android:id="@+id/text_hello"
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:gravity="center"
    android:text="@string/string_ref_in_strings_xml"
    android:textColor="@android:color/holo_blue_light"
    android:textColor="#FF0000"
    android:textSize="18sp"
    android:textStyle="bold|italic"/>
```

```
<!-- View is the simplest view, just an empty rectangle but can be useful -->
<View
    android:layout_width="80dp"
    android:layout_height="60dp"
    android:layout_margin="8dp"
    android:background="#FF00FF"
    android:paddingLeft="16dp" />
```

```
<!-- A Button is actually a TextView but styled to look like a button -->
<Button
    android:layout_width="180dp"
    android:layout_height="wrap_content"
    android:text="Sample Button" />
```

```
<!-- The EditText is a user-editable area -->
<EditText
    android:layout_width="250dp"
    android:layout_height="40dp"
    android:hint="Firstname"
    android:inputType="number" />
```

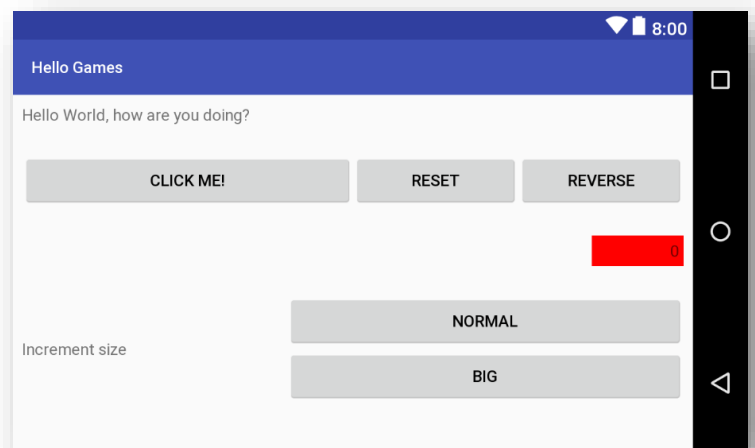
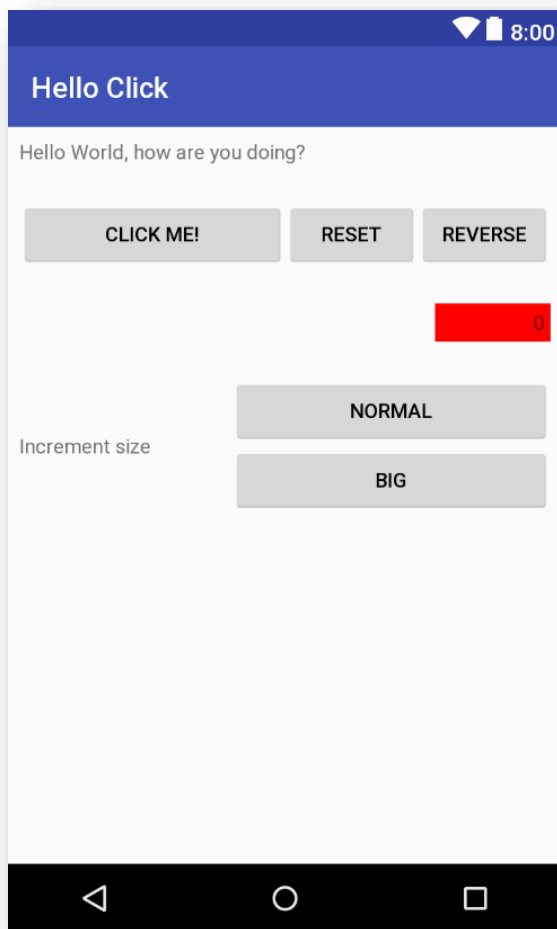
```
<!-- The ImageView displays pictures, such as drawable resources -->
<ImageView
    android:layout_width="300dp"
    android:layout_height="0dp"
    android:adjustViewBounds="true"
    android:scaleType="fitCenter"
    android:src="@drawable/some_picture" />
```

## Fourth step: Create your UI

### Creating the UI

Here is a screenshot of the interface we want to create. You can create your UI directly in the text tab editor or drag and drop your widget in the WYSIWYG editor and then change the few properties that need to in the XML editor.

Remember that you will have warning if you use a static string instead of a resource, so be sure to create new string resources either by modifying your code directly in the strings.xml file or by using the graphical assistant.



- *Click me* button takes half the width of the screen
- *Reset* and *Reverse* equally share the remaining space
- Counter width is fixed to 80 dp
- Counter background is colored (here red but exact color is free)
- There is a small free space around the value in the counter
- Counter is aligned to the right and Counter value is also aligned to the right inside the counter
- Normal and big buttons take 60% of screen width
- "Increment size" label is vertically centered between Normal and Big buttons.

## The Constraint Layout

Since transforming an xml layout resource into a full fledge View that can be displayed (which is called inflating) costs a lot of resources to the system, it is strongly advised to make our layout files as simple as possible, but most of all, as flat as possible. To avoid nesting ViewGroups in ViewGroups, we use the Constraint Layout.

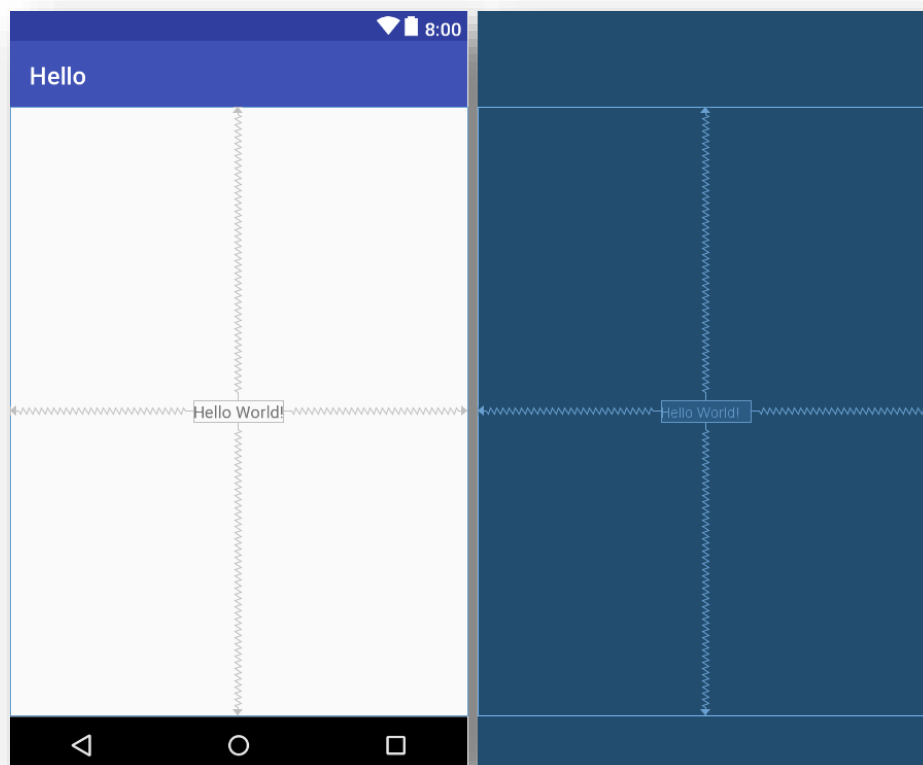
In a Constraint Layout, we set constraints on view borders and attach these constraints to other view borders, or even screen border.

Each View in a Constraint Layout must have at least one horizontal constraint, and one vertical constraint.

In the xml, constraints are attributes set to attach a border to another:

When the dimension of a view depends on its constraints instead of its actual width and/or height attributes, use *0dp* as value, as it means "match\_constraints" for the system.

```
<!-- Horizontal constraint : the left border of this TextView is attached to the
left border of its parent (which here is the screen).
Vertical constraint : the top border is attached to the bottom of some other
view -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Left here"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/some_content_above" />
```

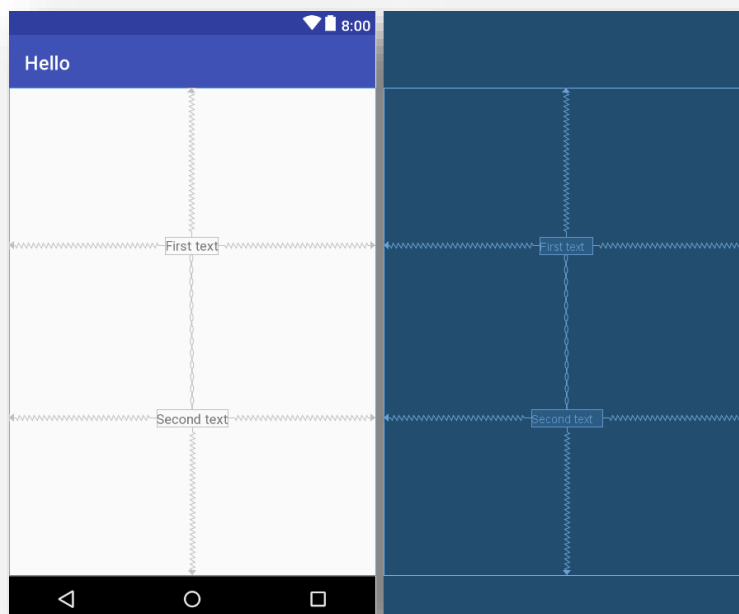


## Chains

Paraphrasing Android documentation here: "A set of widgets are considered a chain if they are linked together via a bi-directional connection. Chains are controlled by attributes set on the first element of the chain (the "head" of the chain). The head is the left-most widget for horizontal chains, and the top-most widget for vertical chains."

```
<TextView
    android:id="@+id/red_square"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_marginTop="16dp"
    android:background="#FF0000"
    android:gravity="center"
    android:text="Red"
    app:layout_constraintHorizontal_bias="0.2"
    app:layout_constraintHorizontal_chainStyle="packed"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/green_square"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/green_square"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:background="#00FF00"
    android:gravity="center"
    android:text="Green"
    app:layout_constraintLeft_toRightOf="@+id/red_square"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="@+id/red_square" />
```

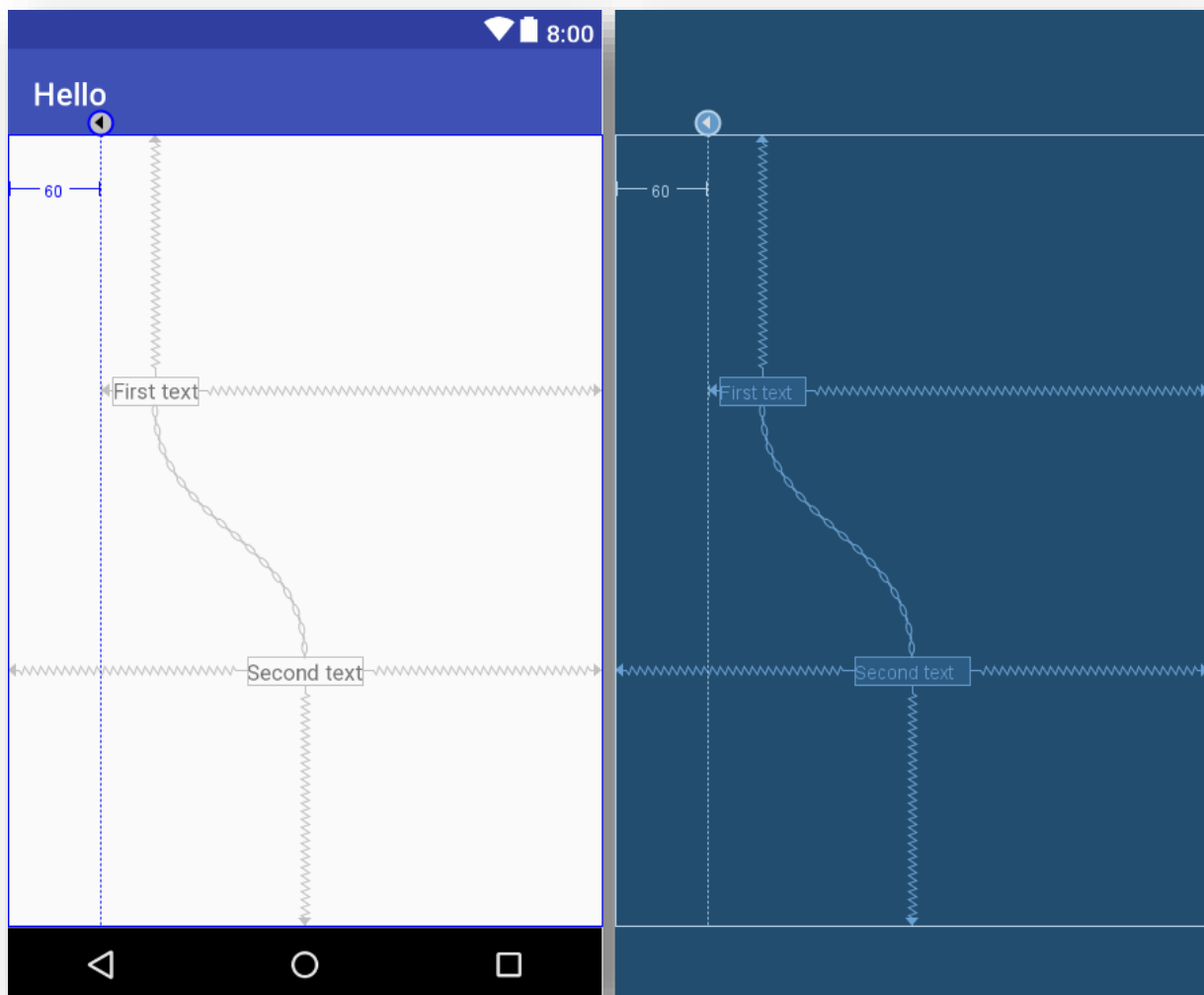


## Guidelines

Guidelines are virtual helpers. They will not appear on the screen but help you position your other widgets.

```
<!-- A vertical guide at 20dp from the left border of the screen -->
<android.support.constraint.Guideline
    android:id="@+id/vertical_guide"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintGuide_begin="20dp"/>

<!-- An horizontal guide in the middle of the screen -->
<android.support.constraint.Guideline
    android:id="@+id/horizontal_guide"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.5"/>
```



## Barriers

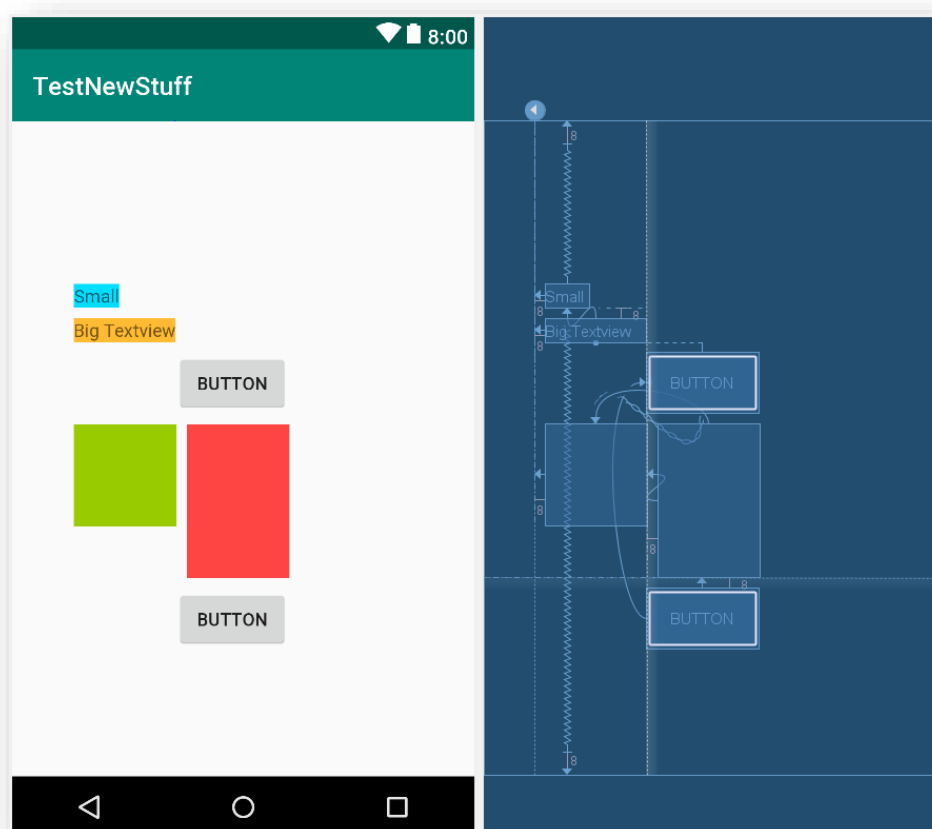
Like Guidelines, Barriers are virtual helpers. A Barrier is like a dynamic guideline that will position itself at the end of a group of views

```
<!--A vertical barrier to the right of both textView1 and textView 2 -->
```

```
<android.support.constraint.Barrier
    android:id="@+id/barrier"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="end"
    app:constraint_referenced_ids="textView1,textView2"/>
```

```
<!--An horizontal barrier below block1 and block 2 -->
```

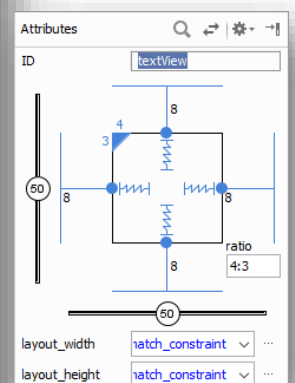
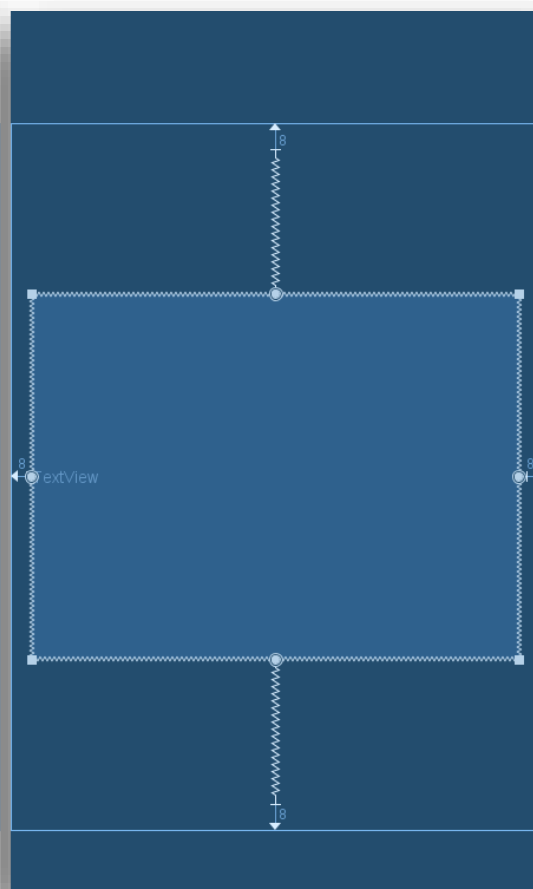
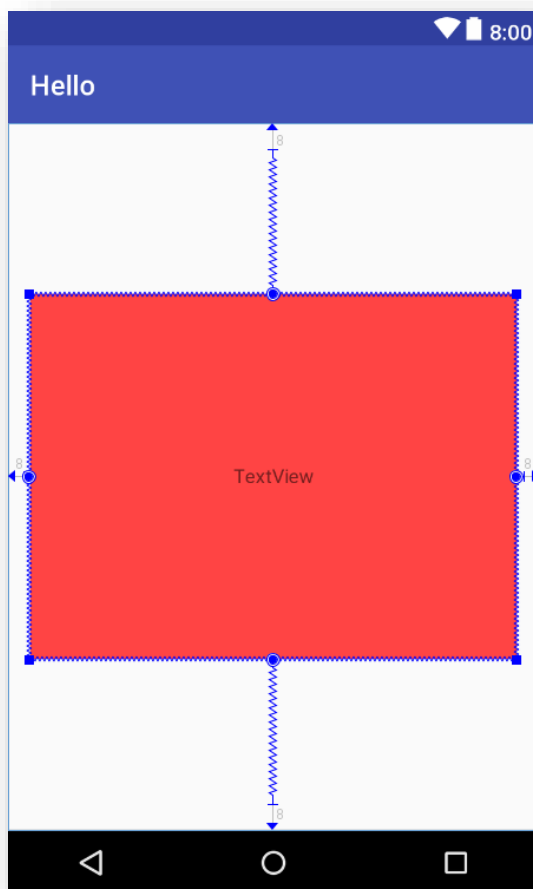
```
<android.support.constraint.Barrier
    android:id="@+id/barrier2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:barrierDirection="bottom"
    app:constraint_referenced_ids="block1,block2"/>
```



## Other stuff

The constraint layout viewgroup also offers many useful tricks, like the ratio attribute

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:background="#FF0000"
    android:text="Some text"
    app:layout_constraintDimensionRatio="3:2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
```





or the weight you can use to size chain elements

```
<TextView
    android:id="@+id/red_square"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:background="#FF0000"
    android:gravity="center"
    android:text="Red"
    app:layout_constraintHorizontal_weight="3"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@+id/green_square" />

<TextView
    android:id="@+id/green_square"
    android:layout_width="0dp"
    android:layout_height="50dp"
    android:background="#00FF00"
    android:gravity="center"
    android:text="Green"
    app:layout_constraintHorizontal_weight="1"
    app:layout_constraintLeft_toRightOf="@+id/red_square"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="@+id/red_square" />
```

➡ Use your Constraint layout knowledge to create the complete UI for the Hello Click app.

For now this is just UI and the screen does nothing, but we will add behavior by code later.