Mobile Applications for Sensing and Control

Professional Master's Program in
Electrical and Computer Engineering
EEP 523

The first 20 hours how to learn anything | *Josh Kaufman* | TEDxCSU

- https://www.youtube.com/watch?v=5MgBikgcWnY

TODAY

Announcements

TODAY

1. Kotlin concepts

2. Summary from Lecture 1

3. App User Interface
   a) Layouts
   b) Widgets

4. Multiple activities (Intents)

Lecture 2 - Practice

- **Class exercise (30min)**: practice with layout and widgets
- **Assignment 1**: GUIs and multiple activities
- **Challenge 1**: a) add pictures to widgets
  b) swipe event

WEEK	Topics	Assignments
1 April 2nd	Introduction My first Android App	Assign. #0 (not-graded)
2 April 9th	Android Programming (I)	Assign. #1
3 April 16th	Android Programming (II)	Assign. #2
4 April 23th	Smartphone Sensors	Assign. #3 **Face Detector**
5 April 30th	**TinyML**	Assign. #4 **Gesture Recognition**
6 May 7th	Arduino programming (I)	Assign. #5
7 May 14nd	Arduino programming (II) Android-Arduino Interaction	*Final project proposal deadline*
8 May 21st	Databases/??? Remote data bases, Firebase, Web services	--
9 May 28th	Special topics - Web services - Hybrid Mobile App Frameworks	--
10, 11 June 4th, 11th	Final projects presentation	*Final project deadline*

Kotlin: Classes properties

- Classes in Kotlin can have **properties**. These can be declared as mutable, using the ***var*** keyword or read-only using the ***val*** keyword.

```
class Address {
    var name: String = ...
    var state: String? = ...
    var zip: String = ...
}
```

- To use a property, we simply refer to it by name, as if it were a field in Java:

```
fun copyAddress(address: Address): Address {
    val result = Address() // there's no 'new' keyword in Kotlin!
    result.name = address.name
    result.street = address.street
    // ...
    return result
}
```

Kotlin: Null Safety

NullPointerException (NPE)

thrown when program attempts to use an object reference that has the null value

- Kotlin type system: eliminates the Null References

  The billion Dollar Mistake:
  Tony Hoare, invented the null reference in 1965. Speaking at a software conference called QCon London in 2009, he apologised for inventing the null reference

  > *"But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years."*

Kotlin: Null Safety

- Nullable references: can hold a null
- Non-null references: e.g. String type

```
var a: String = "abc"
a = null // compilation error

var a: String? = "abc"
a = null // ok
val t = a.length()
```

a can hold a null

Error : variable can't be null

- **Safe calls:**

```
val t = a?.length
```

Is equivalent to

```
If (a!=null)
        val t = a?.length
else
        null
```

reduces this complexity and execute an action only when the specific reference holds a non-null value. It allows us to combine a null-check and a method call in a single expression

Kotlin: Null Safety

- Nullable references: can hold a null
- Non-null references: e.g. String type

```
var a: String = "abc"
a = null // compilation error

var a: String? = "abc"
a = null // ok
```

a can hold a null

- Safe calls: `val t = a?.length` t = null

- The **!! Operator**: converts any value to a non-null type and throws an exception if the value is null

`val t = a!!.length` Throws a NPE

Kotlin: Late-Initialized Properties and Variables

- Properties declared as having a non-null type must be initialized in the constructor.
- Fairly often this is not convenient. Example:

```
class FlagsActivity : AppCompatActivity() {
    var mFlagPic:  ImageView
    var mFlagDesc: TextView
}
```

Compiling error: property must be initialized

ImageView and ***TextView*** are <u>not nullable</u> type

Cannot use ***?*** ***!!***

```
class FlagsActivity : AppCompatActivity() {
    lateinit var mFlagPic:  ImageView
    lateinit var mFlagDesc: TextView
```

Using **lateinit**, the initial value does not need to be assigned.

Kotlin: Functions

- Functions in Kotlin are declared using the ***fun*** keyword:

```
fun double(x: Int): Int {
        return 2 * x
}
```

- **Parameters**: *name: type*

```
fun powerOf(number: Int, exponent: Int) { ... }
```

- **Default Arguments**: function parameters can have default values, which are used when a corresponding argument is omitted

```
fun read(b: Array<Byte>, off: Int = 0, len: Int
= b.size) { ... }
```

```
fun foo(bar: Int = 0, baz: Int) { ... }
foo(baz = 1) // The default value bar =
0 is used
```

Kotlin: Functions

- **Single-Expression functions** when a function returns a single expression, the curly braces can be omitted and the body is specified after a = symbol

```
fun double(x: Int): Int = x * 2
fun double(x: Int) = x * 2
```

Explicitly declaring the return type is optional when this can be inferred by the compiler

- **Unit-returning** (*void* in Java)

optional

```
fun printHello(name: String?): Unit {
        if (name != null)
                println("Hello ${name}")
        else
                println("Hi there!")
 }
```

Kotlin: Lambdas expressions

- '**function-literal**' – function that is not declared, but passed immediately as an expression

```
val sum: (Int , Int) -> Int = { x, y -> x + y }
```

- Surrounded by curly braces
- Parameter declarations inside curly braces
- Optional type annotations

```
val sum = { x: Int, y: Int  -> x + y }
```

- Body goes after **->** sign

The last (single) expression inside the body is the return value

Notes

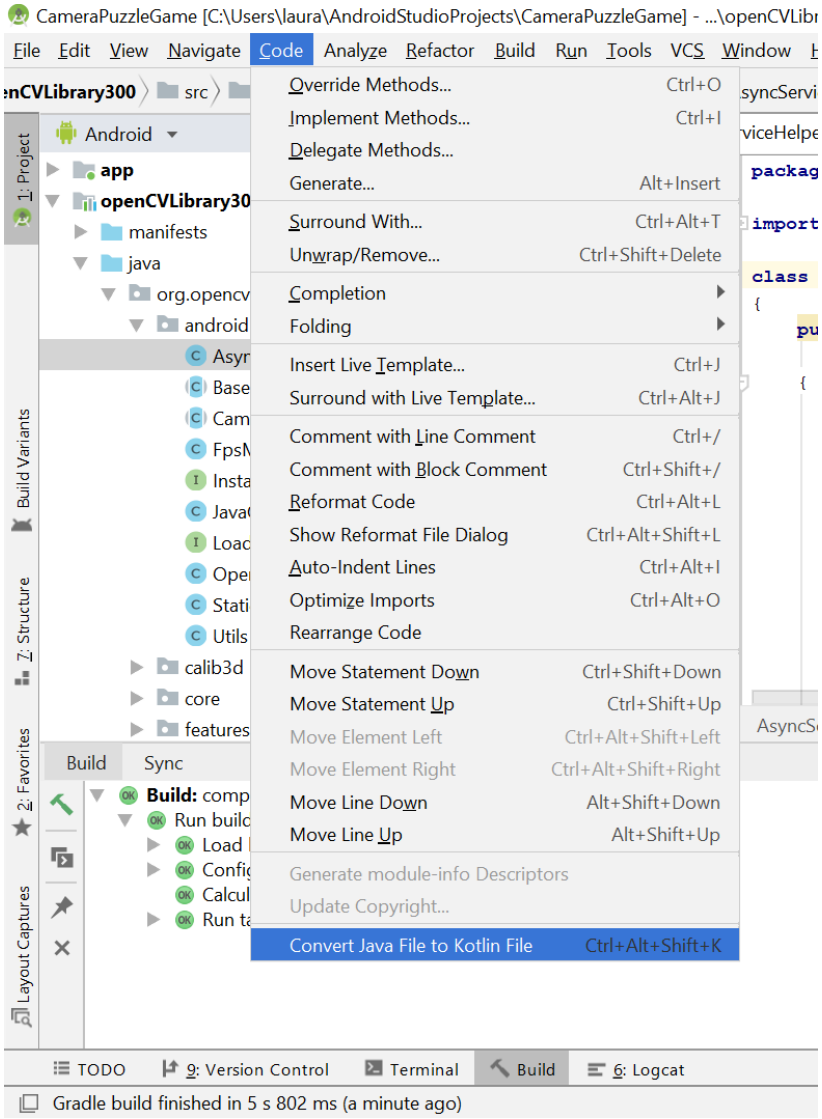
Companion objects

- Unlike Java or C#, Kotlin doesn't have static members or member functions
- function or a property to be tied to a class rather than to instances of it (similar to @staticmethod in Python)

```
companion object {
        private val PICK_KNOTE_REQUEST = 1234
        private val KEY = "knoteselected"
}
```

Convert Java Class to Kotlin

- Select Java class
- Code ->
  Convert Java File
- to Kotlin File
- It might ask you to
  also convert related classes



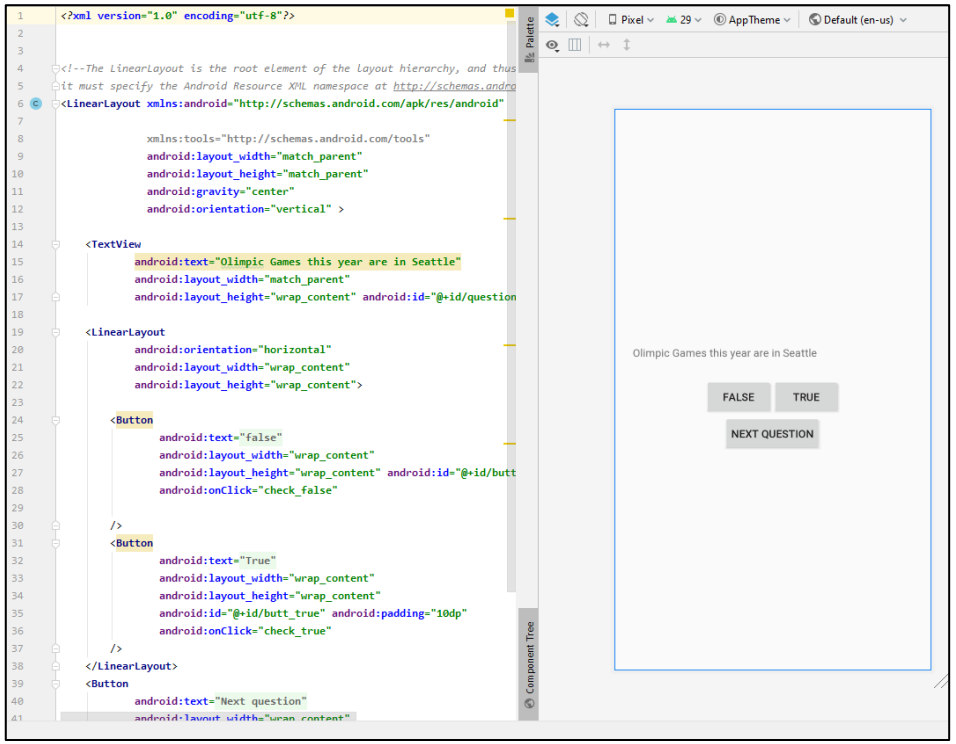
From Lecture 1: App Basics

The most basic App consists of

an *activity (.kt)* and a *layout (.xml)*

```kotlin
class MainActivity : AppCompatActivity() {

    private var mCurrentIndex = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mCurrentIndex = 1
    }

    private val questionBank = listOf(
        Question("This year the Olympic Games are in Seattle", answer: false),
        Question("First modern Olympic games were in 1796", answer: false),
        Question("This year games there will a total of 40 sports", answer: false),
        Question("The Olympic flag contains 5 colors", answer: true),
        Question("Women were allowed to participate in the Olimpics in 1900", answer: true),
        Question("Skateboarding is an Olympic sport", answer: true),
        Question("Gold medals are made of 50% gold", answer: false),
        Question("Michael Phelps is the athlete with the most Olympic medals", answer: true),
        Question("Men and women do NOT compete against each other in any ...", answer: true),
        Question("Swimming obstacle race was Olympic in one Game", answer: false)
    )


    fun updateQuestion(view: View){
        mCurrentIndex ++
        if (mCurrentIndex > questionBank.size) {
            mCurrentIndex = 1
        }
        val questionTextResID = questionBank[mCurrentIndex].textResId
        questionView.setText(questionTextResID)
    }
```

```xml
<?xml version="1.0" encoding="utf-8"?>


<!--The LinearLayout is the root element of the layout hierarchy, and thus
it must specify the Android Resource XML namespace at http://schemas.andro
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

              xmlns:tools="http://schemas.android.com/tools"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:gravity="center"
              android:orientation="vertical" >

    <TextView
            android:text="Olimpic Games this year are in Seattle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" android:id="@+id/question

    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content">

        <Button
                android:text="false"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:id="@+id/butt
                android:onClick="check_false"

        />
        <Button
                android:text="True"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/butt_true" android:padding="10dp"
                android:onClick="check_true"
        />
    </LinearLayout>
    <Button
            android:text="Next question"
            android:layout_width="wrap_content"
```



From Lecture 1: App Basics

The most basic App consists of

an *activity (.kt)*

```
class MainActivity : AppCompatActivity() {

    private var mCurrentIndex = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mCurrentIndex = 1
    }

    private val questionBank = listOf(
        Question("This year the Olympic Games are in Seattle", answer: false),
        Question("First modern Olympic games were in 1796", answer: false),
        Question("This year games there will a total of 40 sports", answer: false),
        Question("The Olympic flag contains 5 colors", answer: true),
        Question("Women were allowed to participate in the Olimpics in 1900", answer: true),
        Question("Skateboarding is an Olympic sport", answer: true),
        Question("Gold medals are made of 50% gold", answer: false),
        Question("Michael Phelps is the athlete with the most Olympic medals", answer: true),
        Question("Men and women do NOT compete against each other in any ...", answer: true),
        Question("Swimming obstacle race was Olympic in one Game", answer: false)
    )


    fun updateQuestion(view: View){
        mCurrentIndex ++
        if (mCurrentIndex > questionBank.size) {
            mCurrentIndex = 1
        }
        val questionTextResID = questionBank[mCurrentIndex].textResId
        questionView.setText(questionTextResID)
    }
```

A basic Activity

- Inherits from class AppCompaActivity
- Needs to override the onCreate function
- Set the layout (scree view) of that activity

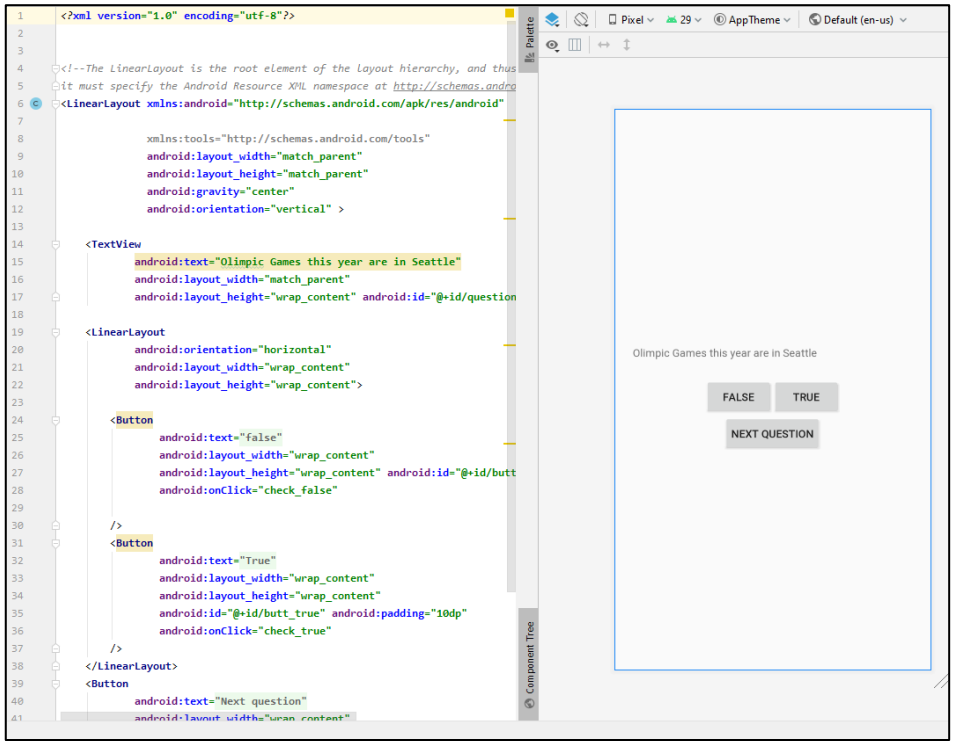
From Lecture 1: App Basics

The most basic App consists of

a *layout (.xml)*

A basic layout (xml file)

- Contains a *ViewGroup* object (the main layout or container)
- Contains one or more widgets or *View* objects (buttons, textViews, …)



MyActivity.kt

- An **activity** represents a single screen with a User Interface. Basic application startup logic that should happen only once for the entire life of the activity.

- `class MainActivity : AppCompatActivity() {…}`
  subclass of Android's **Activity** class that provides compatibility support for older versions of Android.

MyActivity.kt

- An **activity** represents a single screen with a User Interface. Basic application startup logic that should happen only once for the entire life of the activity.

- ***onCreate*() :**
  function to **initialize** the activity -

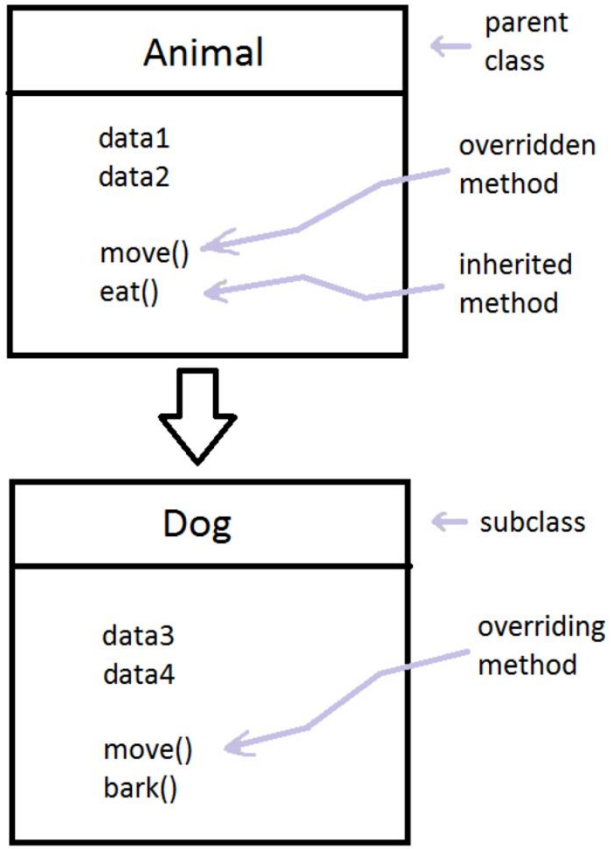
Will be covered in Lecture 3
Save data between activity re-initialization

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

→ Set the layout for the activity

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

- It's an annotation that you can use to tell the compiler and your IDE that the method is an override of a super class method.



Resources and resource ID

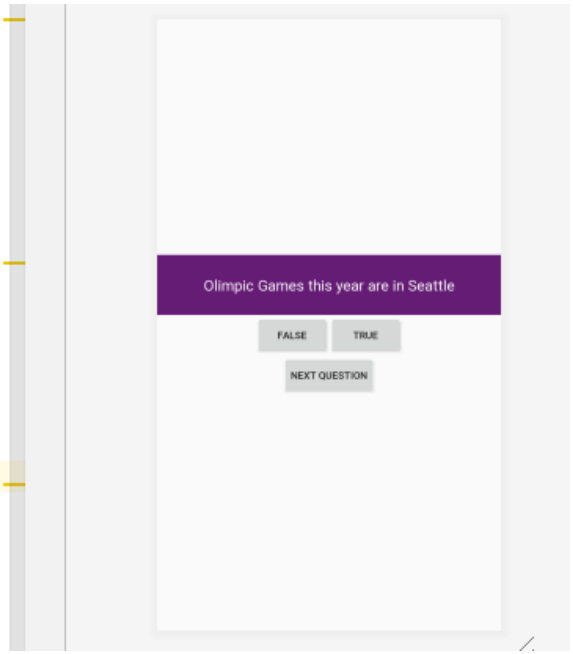
- A layout is a resource.
- A resource is a piece of your application that is not code – things like image files, audio files, and XML files.
- Resources for your project live in a subdirectory of the app/res directory.
- To access a resource in code, you use its **resource ID**. The resource ***ID*** for your layout is R.layout.activity_main.

Refer to widgets from xml (use ID)

```
class MainActivity : AppCompatActivity() {
.....
....


//Change the text in the buttons: two options
//First option
butt_false.text = "WRONG"

//Second option
var falseButtom : Button = findViewById(R.id.butt_false)
falseButtom.setText(" WRONG ")


}
```

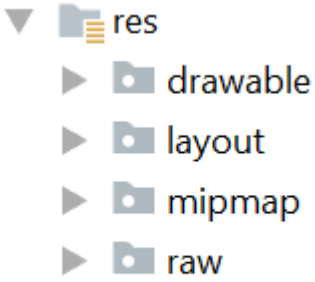
Change the text to display in the False button in our kotlin Activity

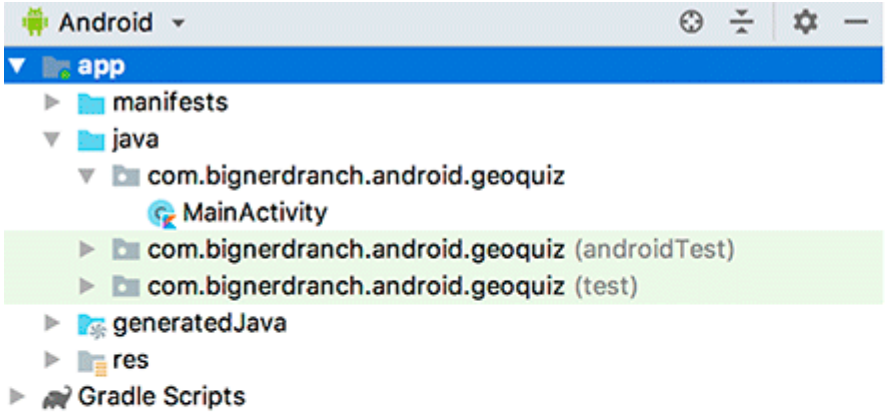
```
<Button
        android:text="false"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/butt_false"
        android:padding="10dp"
        android:onClick="check_false"
        android:textSize="12sp"

    />
    <Button
        android:text="True"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/butt_true"
        android:padding="10dp"
        android:onClick="check_true"
        android:textSize="12sp"
    />
```

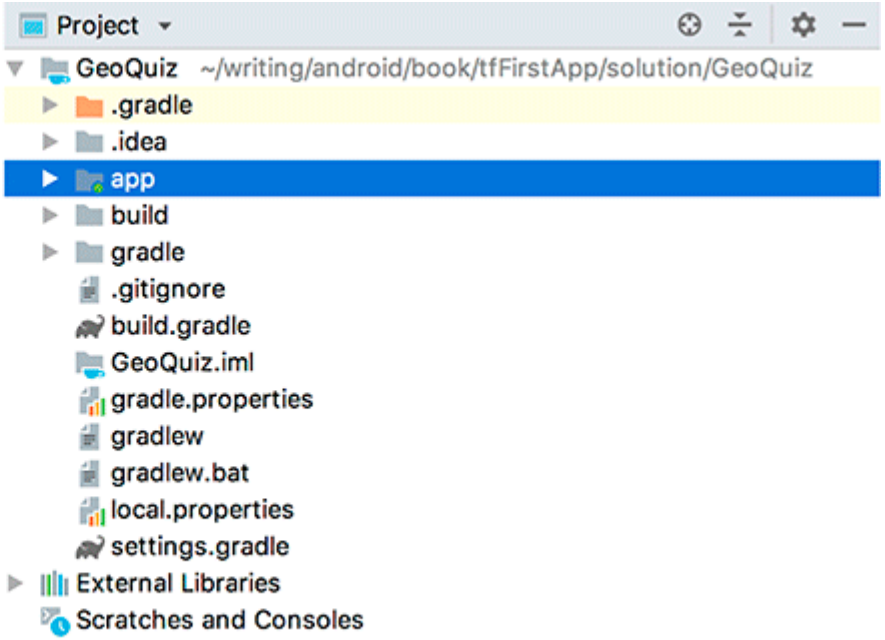


Notes

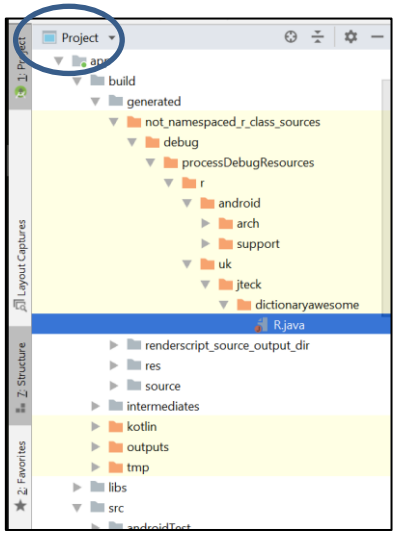
1. What is R? (ex: *R.drawable.myimage*)

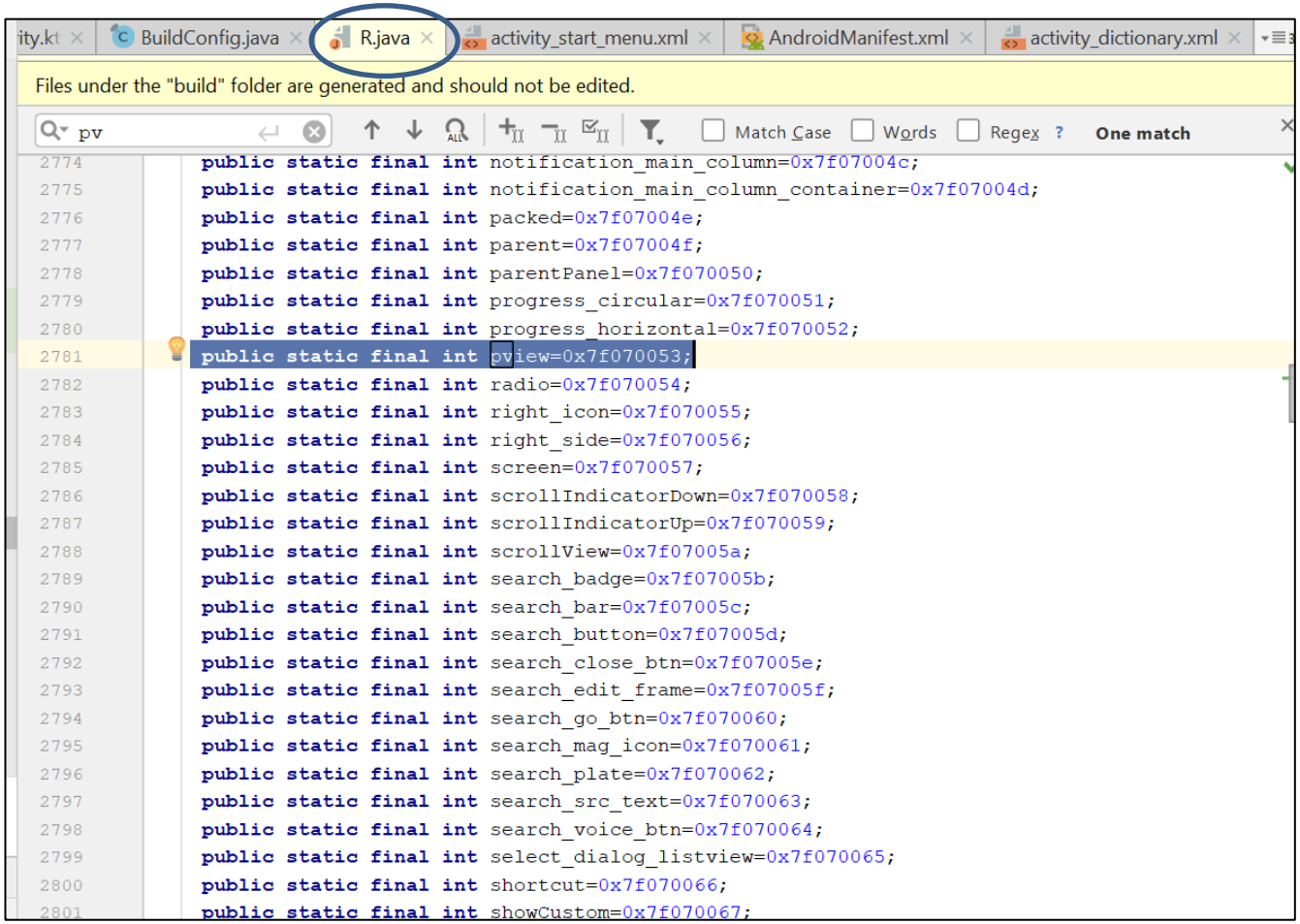




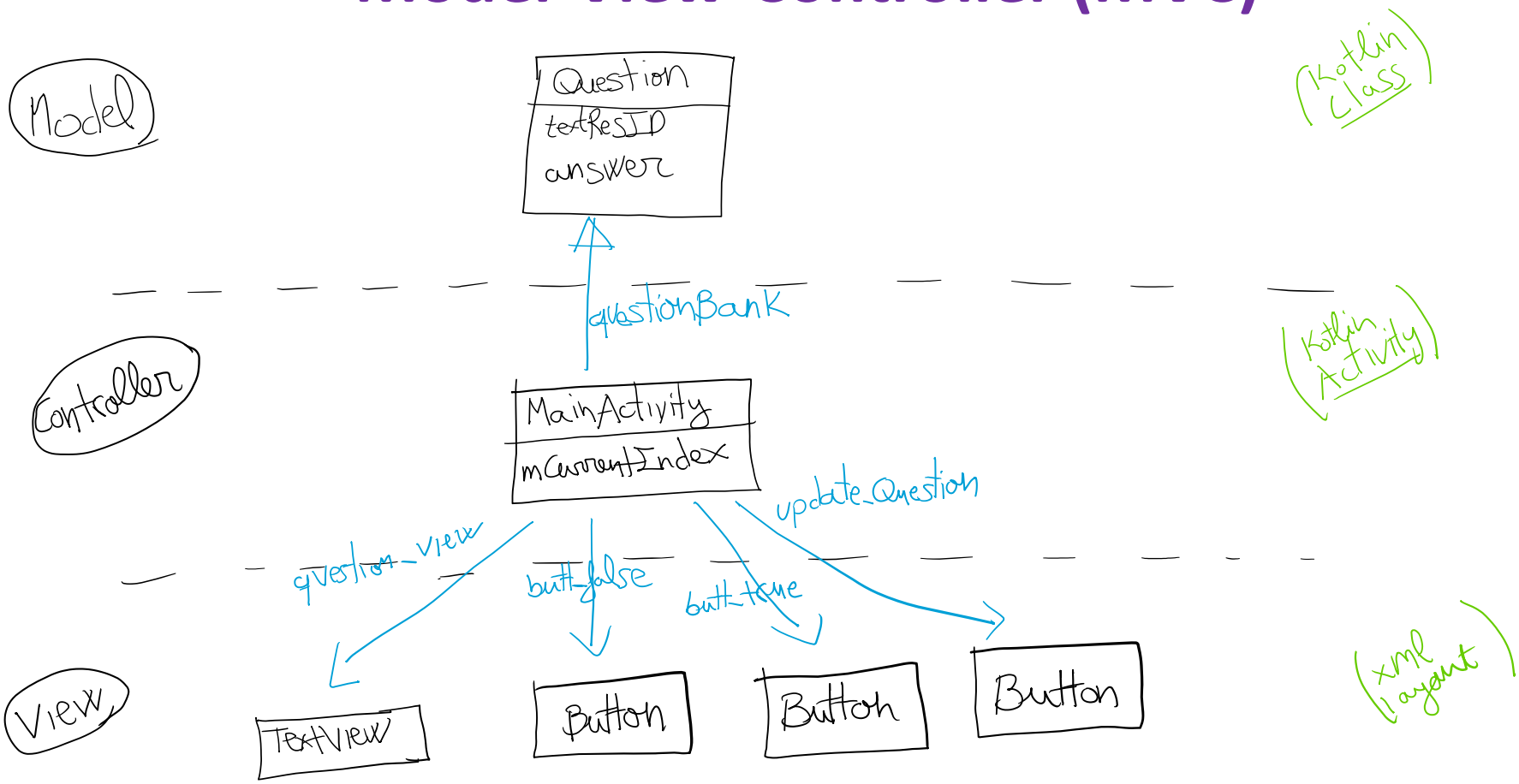


Notes


Project
build
generated
not_namespaced_r_class_sources
debug
processDebugResources
r
android
arch
support
uk
jteck
dictionaryawesome
R.java
renderscript_source_output_dir
res
source
intermediates
kotlin
outputs
tmp
libs
src

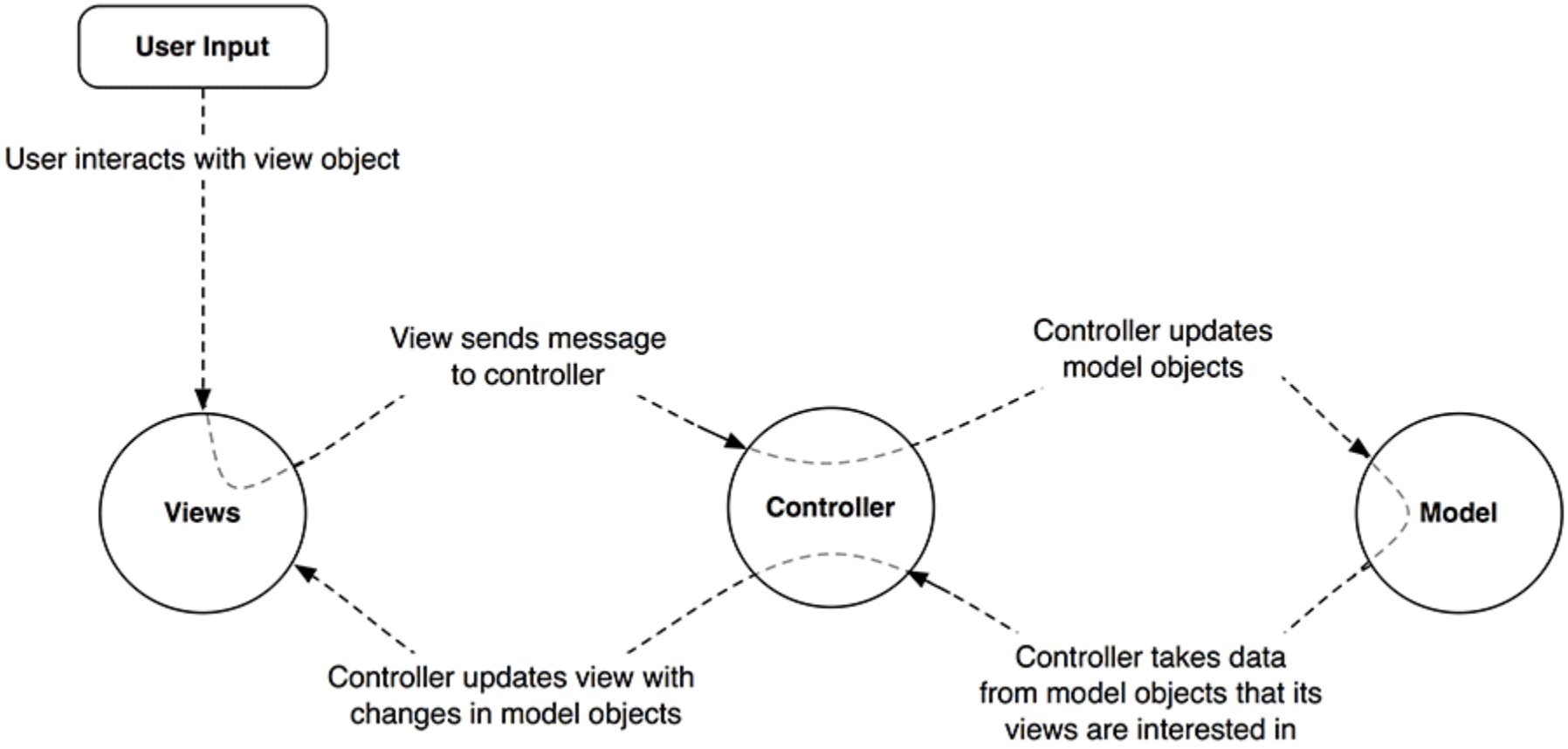

BuildConfig.java
R.java
activity_start_menu.xml
AndroidManifest.xml
activity_dictionary.xml
Files under the "build" folder are generated and should not be edited.
pv
Match Case
Words
Regex
One match
public static final int notification_main_column=0x7f07004c;
public static final int notification_main_column_container=0x7f07004d;
public static final int packed=0x7f07004e;
public static final int parent=0x7f07004f;
public static final int parentPanel=0x7f070050;
public static final int progress_circular=0x7f070051;
public static final int progress_horizontal=0x7f070052;
public static final int pview=0x7f070053;
public static final int radio=0x7f070054;
public static final int right_icon=0x7f070055;
public static final int right_side=0x7f070056;
public static final int screen=0x7f070057;
public static final int scrollIndicatorDown=0x7f070058;
public static final int scrollIndicatorUp=0x7f070059;
public static final int scrollView=0x7f07005a;
public static final int search_badge=0x7f07005b;
public static final int search_bar=0x7f07005c;
public static final int search_button=0x7f07005d;
public static final int search_close_btn=0x7f07005e;
public static final int search_edit_frame=0x7f07005f;
public static final int search_go_btn=0x7f070060;
public static final int search_mag_icon=0x7f070061;
public static final int search_plate=0x7f070062;
public static final int search_src_text=0x7f070063;
public static final int search_voice_btn=0x7f070064;
public static final int select_dialog_listview=0x7f070065;
public static final int shortcut=0x7f070066;
public static final int showCustom=0x7f070067;

Model-View-Controller (MVC)



MVC for example App "OlimpicsQuizz"
from Lecture 1

Model-View-Controller



Model-View-Controller

- ***Model*** objects hold the application's data and "business logic."
  - Model classes are typically designed to model the things your app is concerned with, such as a user, a product in a store, a photo on a server, a television show – or a true/false question.
  - Model objects have no knowledge of the UI; their sole purpose is holding and managing data.
  - In Android applications, model classes are generally custom classes you create. All of the model objects in your application compose its model layer.
  - OlympicQuiz's model layer consists of the **Question** class.

Model-View-Controller

- ***View*** objects know how to draw themselves on the screen and how to respond to user input, like touches.
  - A simple rule of thumb is that if you can see it onscreen, then it is a view.
  - Android provides a wealth of configurable view classes. You can also create custom view classes. An application's view objects make up its view layer. OlympicQuiz's view layer consists of the widgets that are inflated from res/layout/activity_main.xml.
- ***Controller*** objects tie the view and model objects together.
  - They contain "application logic."
  - Designed to respond to various events triggered by view objects and to manage the flow of data to and from model objects and the view layer.
  - In Android, a controller is typically a subclass of Activity or Fragment.(You will learn about fragments in Lecture 3.)
  - OlimpicsQuiz's controller layer, at present, consists solely of MainActivity.

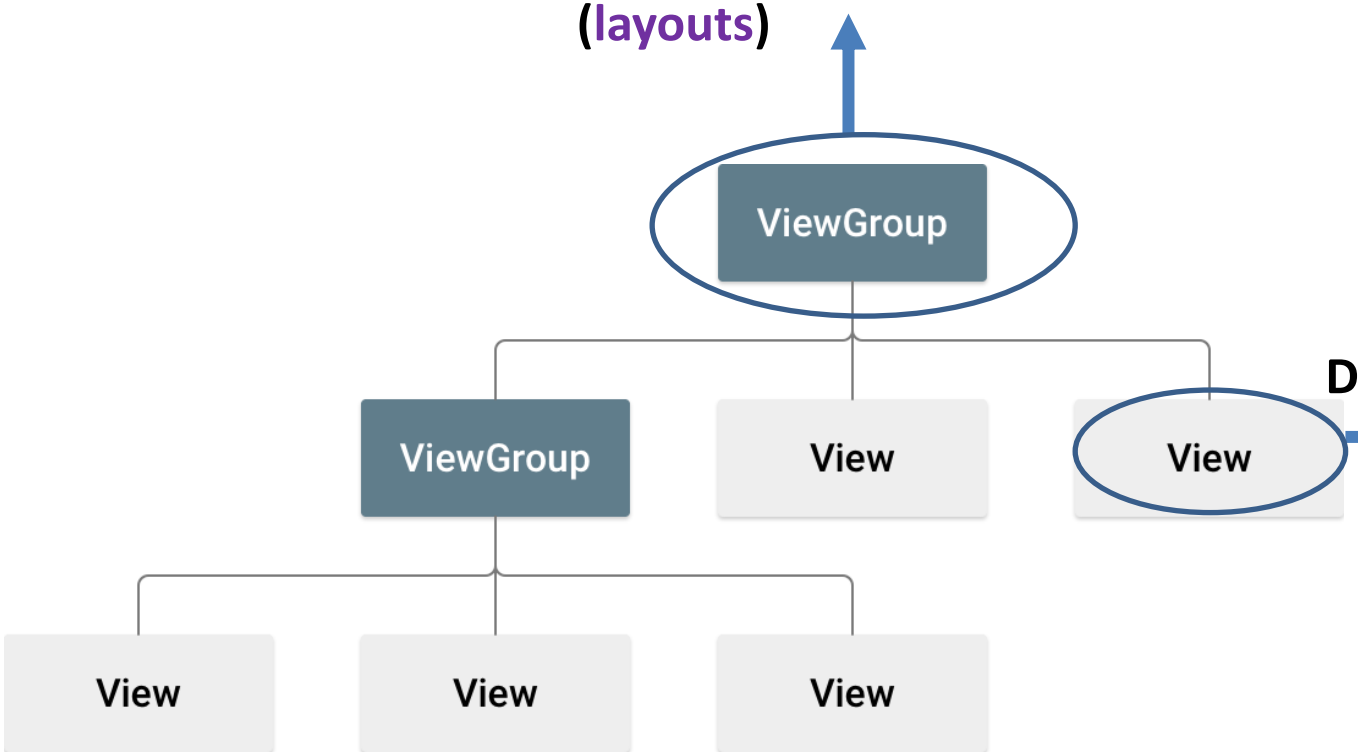
Layout

- A layout defines a structure for a user interface in your app, such as in an activity

- All elements in the layout are built using a *hierarchy* of ***View*** and ***ViewGroup*** objects

Layout

Invisible container that defines the layout structure for *View* and other *ViewGroup* Objects (layouts)



Draws sth the user can interact with (widgets)

MainActivity.kt

```
package edu.uw.pmpee590.myapplication

import ...

class MainActivity : AppCompatActivity() {

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

}
```

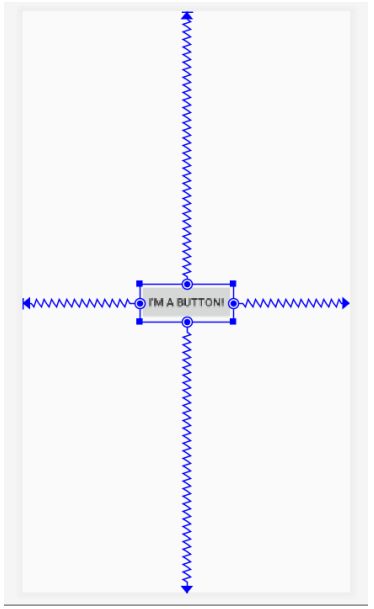
activity_main.xml

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

    <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I'm a button!"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            android:id="@+id/butt"/>

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

ViewGroup

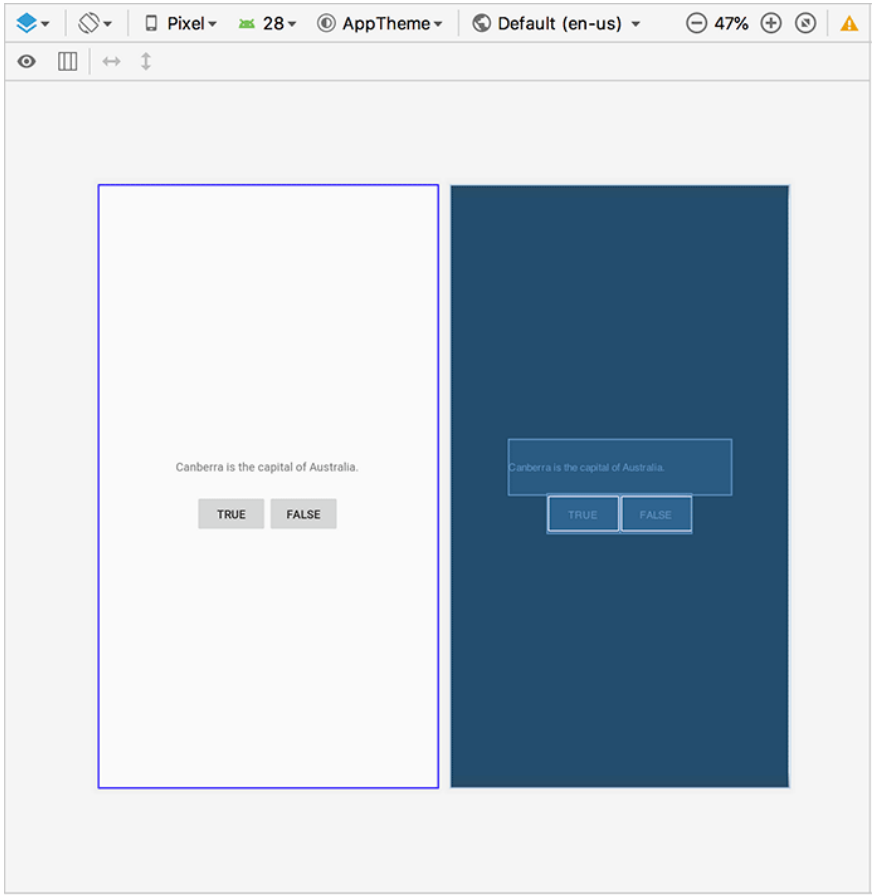
View

Layout Editor



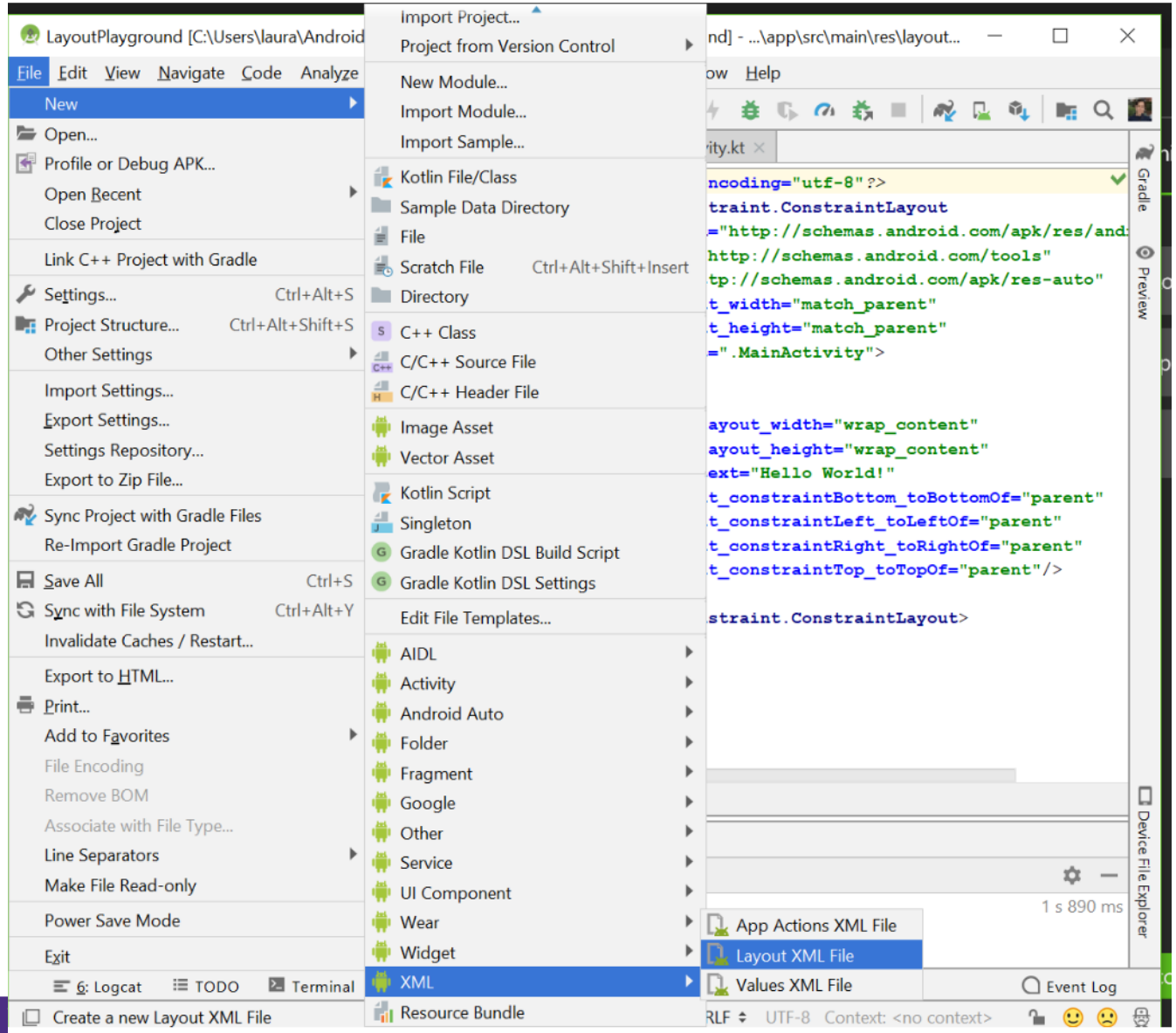
Layout: Design/Blueprint preview

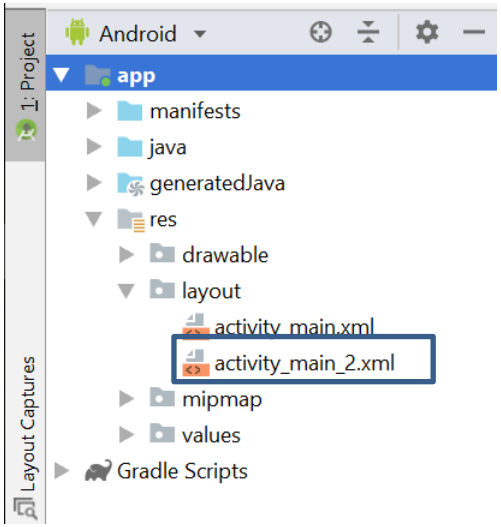
Design preview how the layout would look on a device, including theming.



Blueprint preview focuses on the size of widgets and the relationships between them

Create additional Layouts





Write the XML

XML: language to describe hierarchical text data

- scalable and simple to develop
- Readable by both human and machine
- Lightweight language

```
<Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I'm a button!"
        android:id="@+id/butty"
        android:onClick="goAway"
        android:visibility="gone"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>
```

size

Layouts in Android

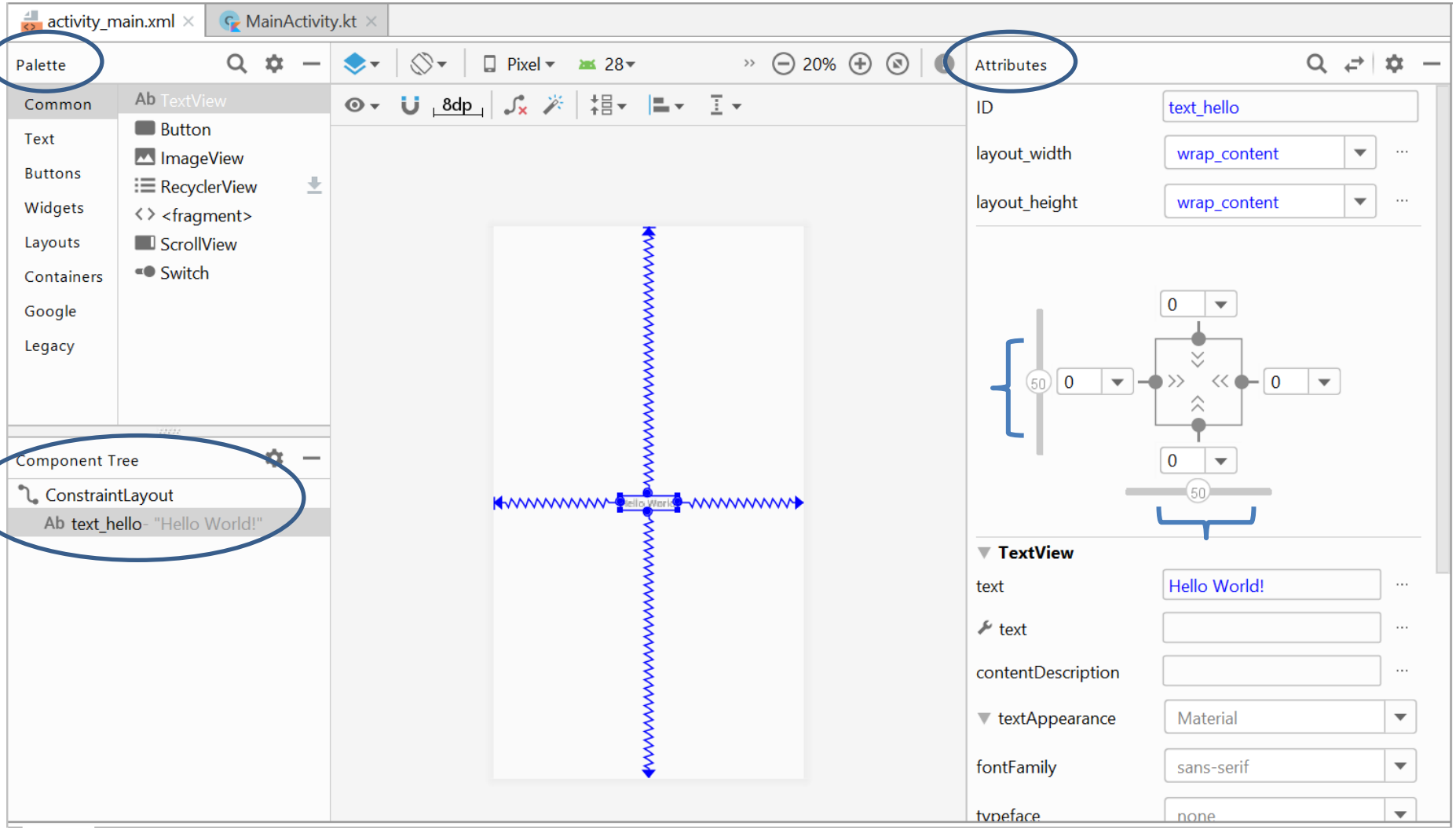
You can declare a layout elements in two ways:

1. **Declare UI elements in XML**. (declarative UI)

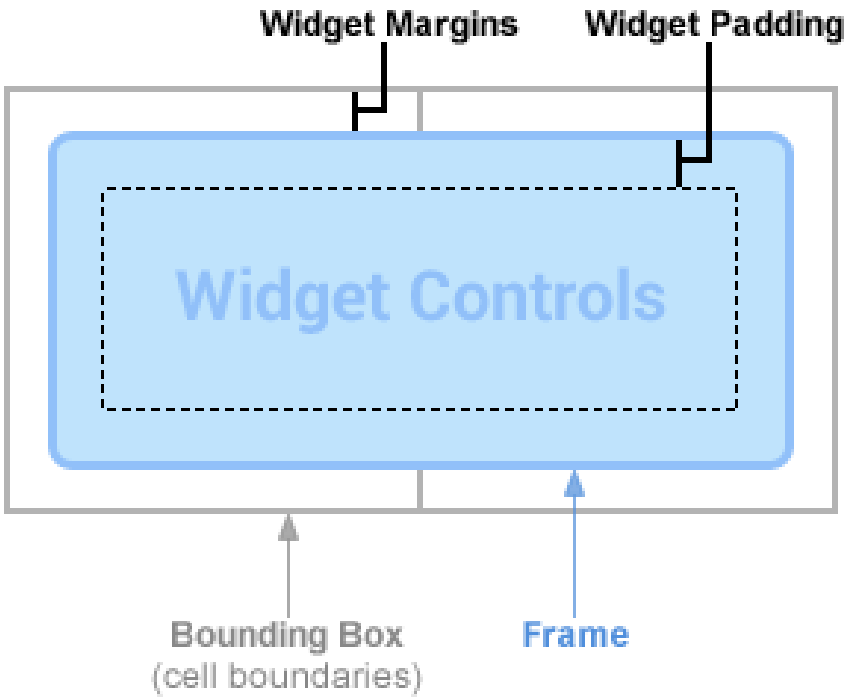
   Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

   You can also use Android Studio's **Layout Editor** to build your XML layout using a drag-and-drop interface.

Layouts Editor



Linear Layout: The box model



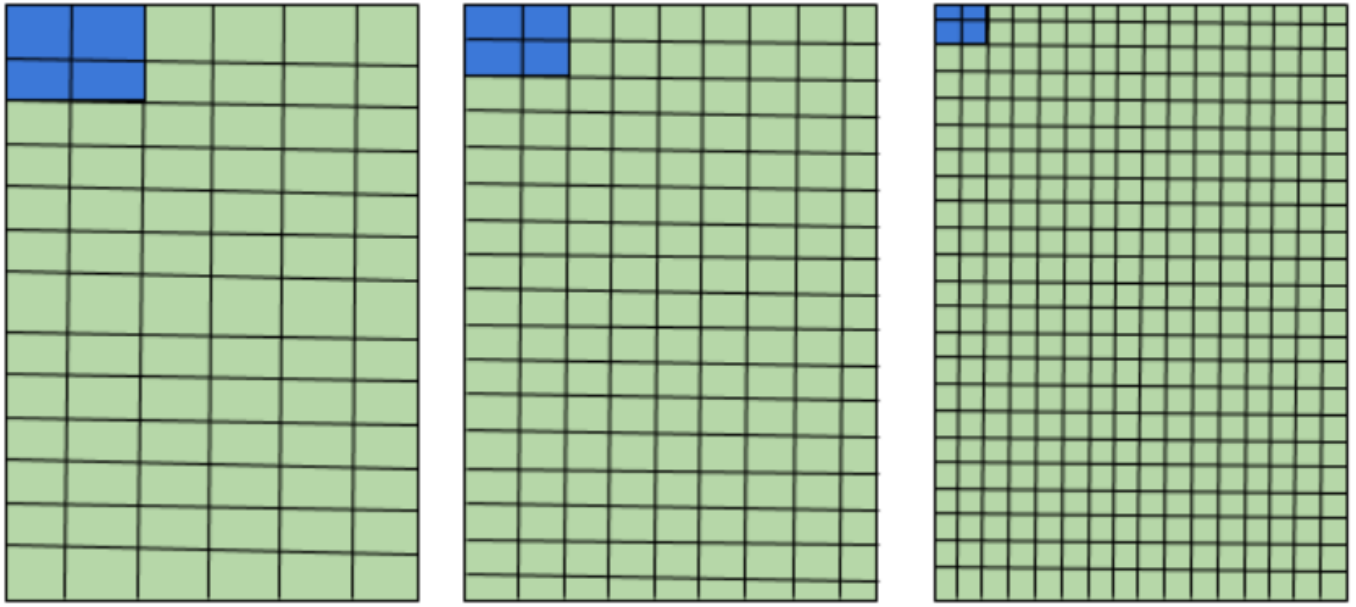
Padding: artificial increase to widget size outsize of content
Margin: invisible separation from neighboring widgets
Frame: outsize padding, a line around edge of widget

Dimension: Pixel density (dpi)

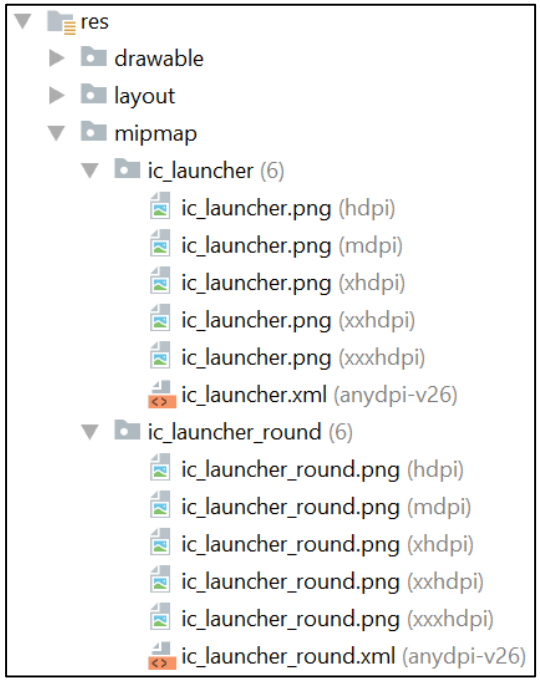
- Number of pixels within a physical area of the screen and is referred to as **dpi (dots per inch).**

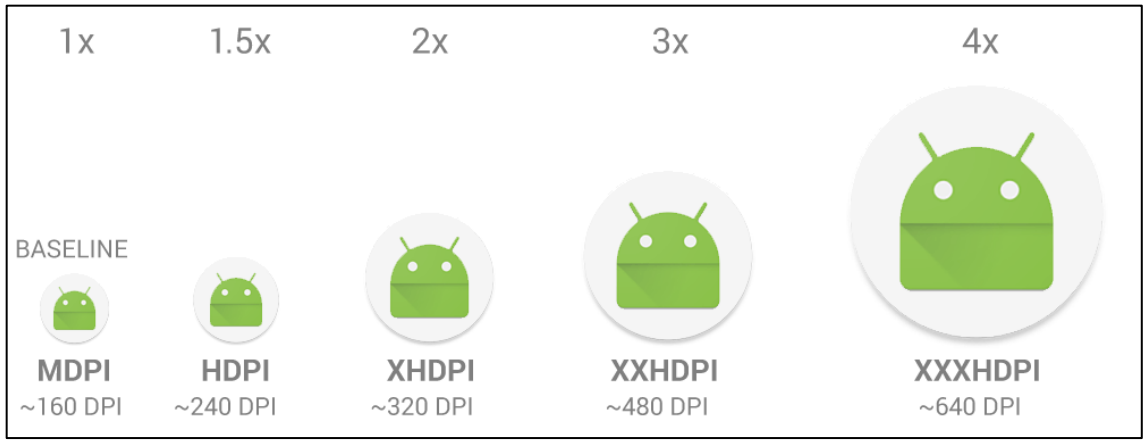
- This is different from the resolution, which is the total number of pixels on a screen

same size but have different pixel densities



Supporting different screen densities





`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
`hdpi`	Resources for high-density (*hdpi*) screens (~240dpi).
`xhdpi`	Resources for extra-high-density (*xhdpi*) screens (~320dpi).
`xxhdpi`	Resources for extra-extra-high-density (*xxhdpi*) screens (~480dpi).
`xxxhdpi`	Resources for extra-extra-extra-high-density (*xxxhdpi*) uses (~640dpi).

Solution: Dp

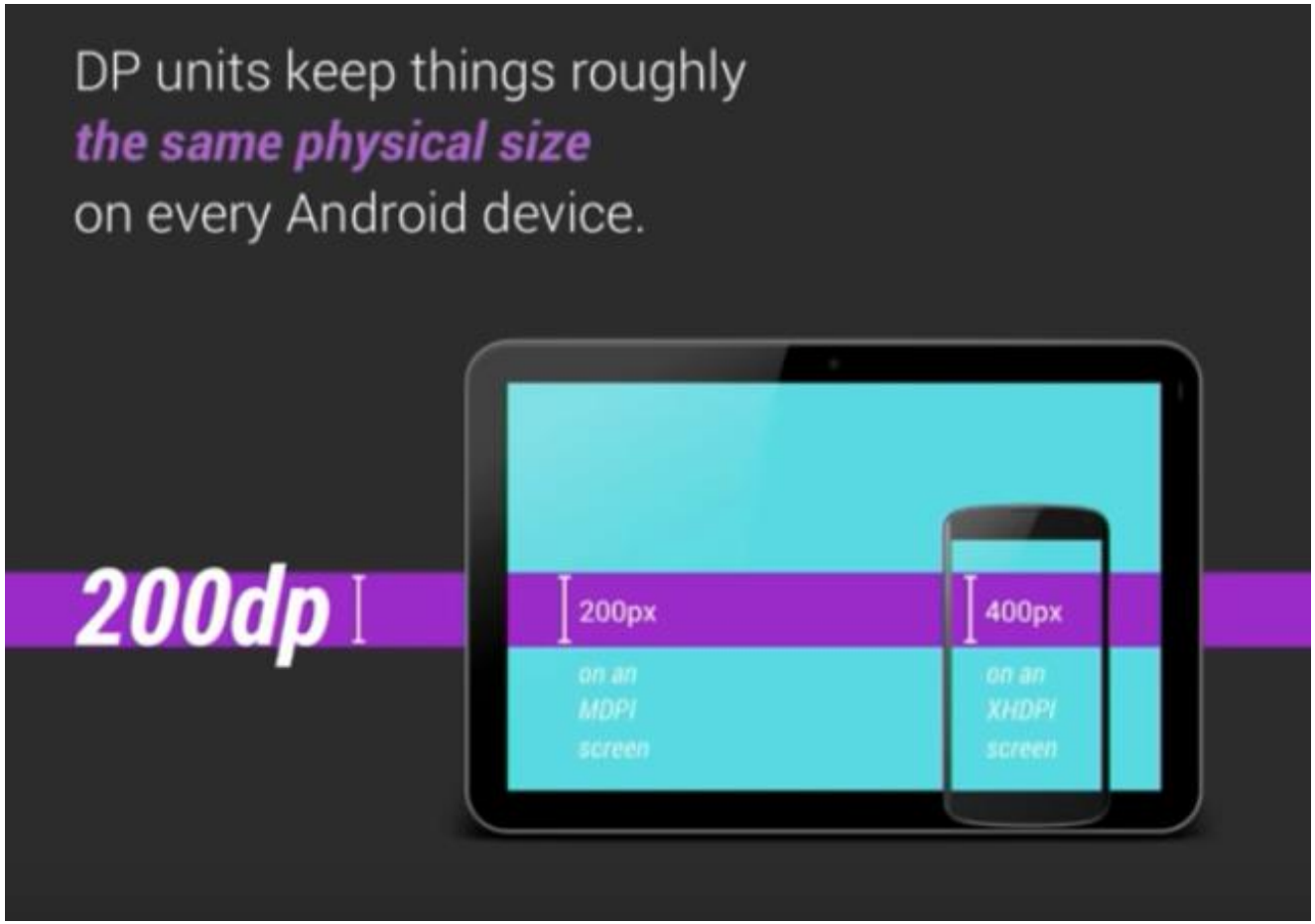
Dp - Density-independent Pixels

- An abstract unit that is based on the physical density of the screen.

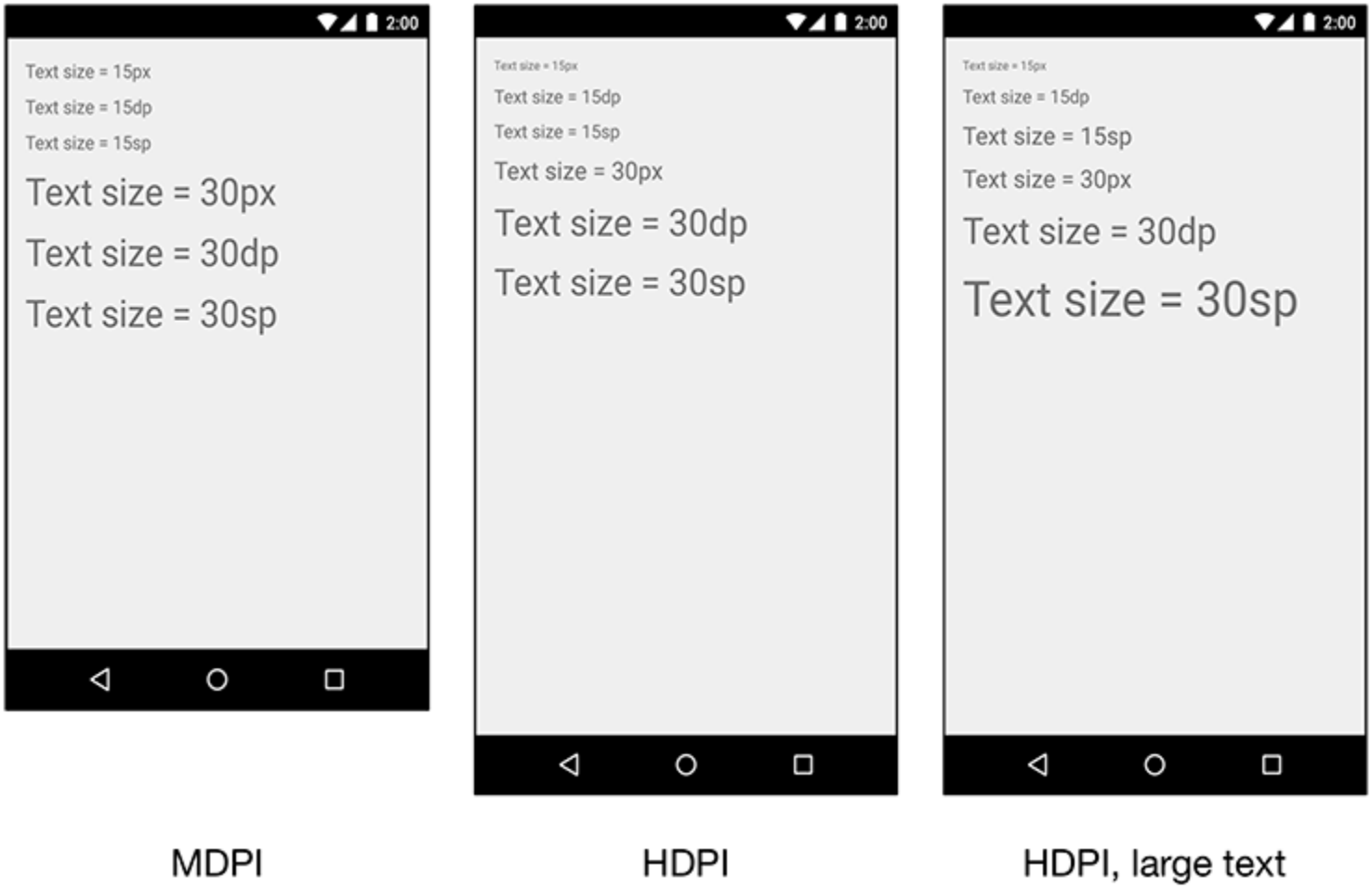
- One dp is a virtual pixel unit that's roughly equal to one pixel on a medium-density screen (**160dpi; the "baseline" density**). Android translates this value to the appropriate number of real pixels for each other density.

- Use to preserve the visible size of your UI on screens with different densities.

Solution: Dp


DP units keep things roughly
the same physical size
on every Android device.
200dp
200px
on an
MDPI
screen
400px
on an
XHDPI
screen

Dimension units



MDPI HDPI HDPI, large text

Dimension: Dp

Example

dpi = dots per inch

Imagine an app in which a scroll or fling gesture is recognized after the user's finger has moved by at least **16 pixels**.

- Phone A : screen with 160 dpi

$$\frac{16 \text{ pixels}}{160 \text{ pixels/inch}} = 1/10^{th} \text{ inch} \sim 2.5\text{mm}$$

- Phone B : screen with 240 dpi

$$\frac{16 \text{ pixels}}{240 \text{ pixels/inch}} = 1/15^{th} \text{ inch} \sim 1.7\text{mm}$$

→ The app in this phone appears more sensitive to the user than with phone A

solution: use dp!!

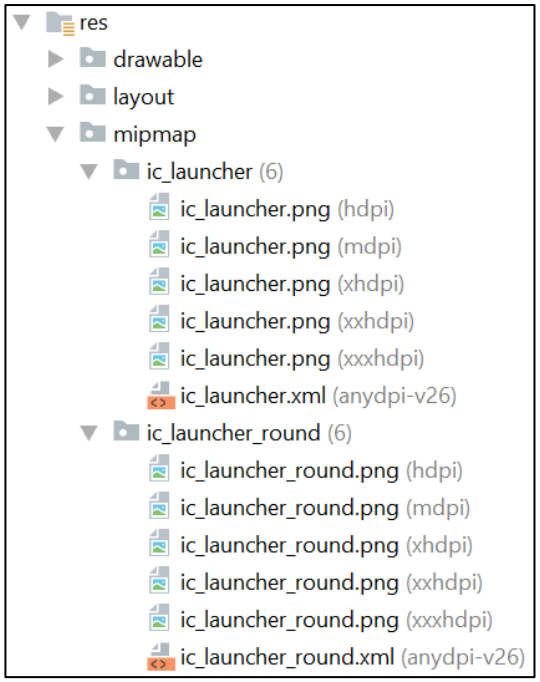
→ see next slide for code implementation

Example

```
// The gesture threshold expressed in dp
private const val dp_threshold = 16.0f
...
private var gesture_threshold: Int = 0
...
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Get the screen's density scale
    val scale: Float = resources.displayMetrics.density

    // Convert the dps to pixels, based on density scale
    gesture_threshold = (dp_threshold * scale).toInt()

  }
```

px = dp * (dpi / 160)

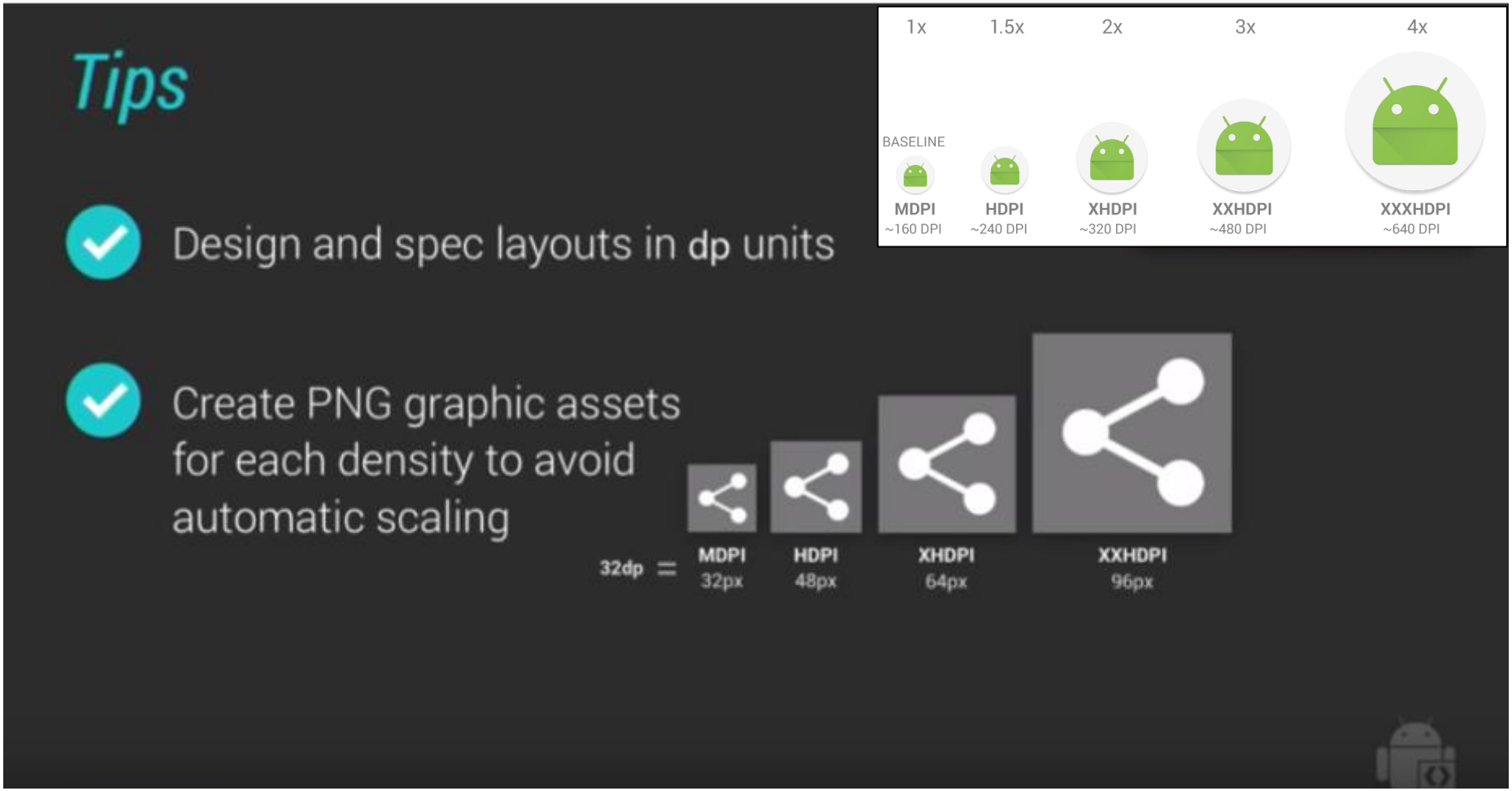
Supporting different screen densities



- To provide good graphical qualities on devices with different pixel densities, you should provide multiple versions of each bitmap in your app—one for each density bucket, at a corresponding resolution

- Then, place the generated image files in the appropriate subdirectory under res/ and the **system will pick the correct one automatically based on the pixel density of the device your app is running on**

Tips


Tips
Design and spec layouts in dp units
Create PNG graphic assets for each density to avoid automatic scaling
32dp =
MDPI
32px
HDPI
48px
XHDPI
64px
XXHDPI
96px
1x
1.5x
2x
3x
4x
BASELINE
MDPI
~160 DPI
HDPI
~240 DPI
XHDPI
~320 DPI
XXHDPI
~480 DPI
XXXHDPI
~640 DPI

Dimension: sp

When defining text sizes: use ***scalable pixels*** **(sp)** (but never use sp for layout sizes!).

The ***sp*** unit is the same size as dp, by default, but it resizes based on the user's preferred text size.

When specifying spacing between two views, use **dp**:

```
<Button android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/clickme"
    android:layout_marginTop="20dp" />
```

When specifying text size, always use **sp**:

```
<TextView android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="20sp" />
```

Other dimensions

Pt - Points

1/72 of an inch based on the physical size of the screen,
assuming a 72dpi density screen.

px - Pixels

Corresponds to actual pixels on the screen.
This unit of measure is not recommended because the actual representation can vary across devices; each devices may have a different number of pixels per inch and may have more or fewer total pixels available on the screen.

mm - Millimeters

Based on the physical size of the screen.

in - Inches

Based on the physical size of the screen.

15min Break


ONE PER WORKING HOUR
TO KEEP YOU MOVING
1ST HOUR
Touch your toes
5 seconds, 3 times
2ND HOUR
Tip-toes and stretch up
5 seconds, 3 times
3RD HOUR
The twist
5 seconds, 3 times
4TH HOUR
The March
Knee towards chest. 10 times
5TH HOUR
Neck stretch
Turn head backwards and sideways. 5 seconds, 3 times
6TH HOUR
The Ballerina
Feet apart. Arm over head reach towards opposite side.
5 seconds, 3 times
7TH HOUR
The Hand Hold
Grab hand and pull towards you until you feel stretch.
5 seconds, 3 times
8TH HOUR
The Banana
Lean, arching your back.
5 seconds, 3 times
9TH HOUR
The Shrug
Roll your shoulders. 5 times
10TH HOUR
Scratch my back
Touch opposite shoulder blade.
5seconds, 3 times
11TH HOUR
Squat
10 times.
12TH HOUR
The penguin
Walk sideways around chair without swaying.
3 times
SIT LESS
MOVE MORE OFTEN
1.Drink More
2.Stand & Call
3.Read Standing
4.Take Stairs
5.Park & Walk

Layout

You can declare a layout in two ways:

1. Declare UI elements in XML

2. **Instantiate layout elements at runtime** (*procedural UI*)

   You can create View and ViewGroup objects (and manipulate their properties) programmatically.

Initiate elements at runtime

When we may need to need it?

Initiate elements at runtime

e.g.: Draw a button on the center of the screen at runtime

1. Create a new Button variable
2. Define the size, and constraints of the Button. Define where to draw the button on the screen
3. Additional configuration of the button
4. Add listener → event when the user clicks the button
5. Add the button to the main layout

Initiate elements at runtime

Draw a button on the center of the screen at runtime

MainActivity.kt

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

    // 1. Create a new Button variable
    val butt = Button(this)
```

Name of xml file

Initiate elements at runtime

Draw a button on the center of the screen at runtime

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout

        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity"
        android:id="@+id/main_layout">
    <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="I'm a button!"
            android:id="@+id/butt"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"  />

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

Initiate elements at runtime

What would be a different strategy to displaying a button at runtime?

-> Create the button on the xml, and set it to be invisible.
-> then make it visible in the kotlin file when needed.

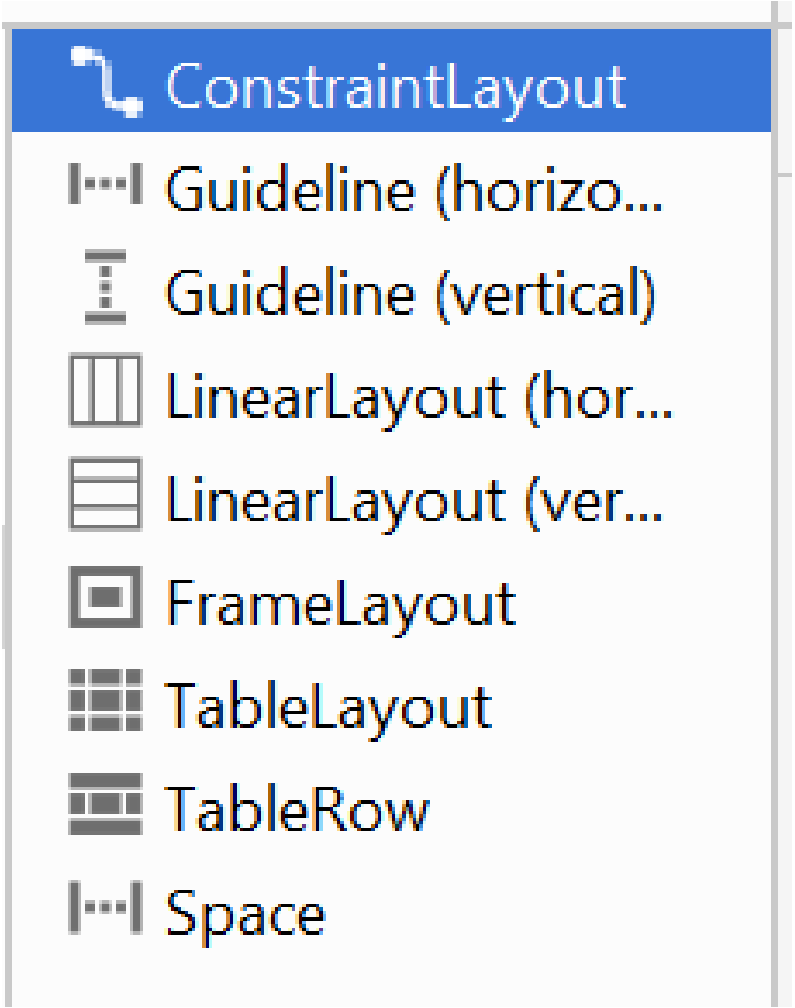
In xml file

```
android:visibility="gone"
```

In the kotlin Class:

```
butty.setVisibility(View.VISIBLE)
```

Common Layouts



Common Layouts

Linear Layout

Organizes its children into **a single horizontal for vertical row**

Scrollbar if length of windows exceeds length of the screen

Relative Layout

Location of child objects **relative to each other or to the parent**

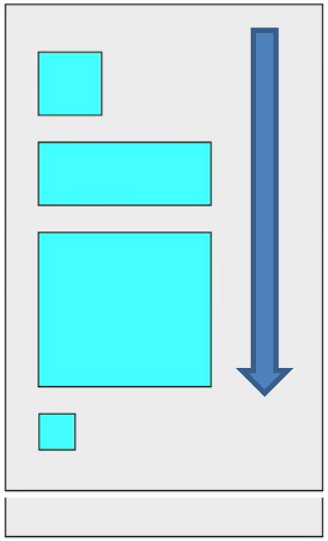
Web View

Displays **web pages**

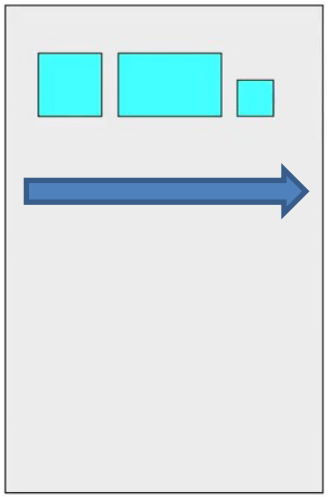
Linear Layout

ViewGroup (*container*) that aligns all children in a single direction

Vertically



Horizontally



All children are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding).

Linear Layout: Gravity

Alignment direction that widgets are pulled

– top, bottom, left, right, center

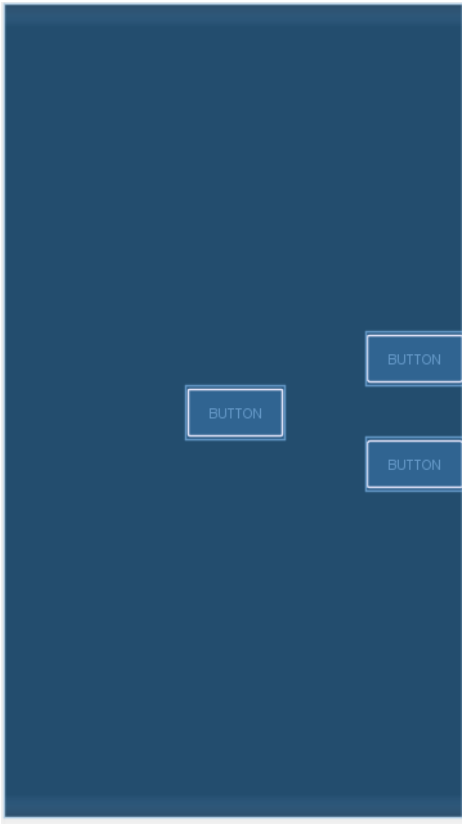
– combine multiple with |

– *gravity* vs *layout_gravity*

In the xml file:

- *android:gravity*
  set gravity on the layout (container) to adjust all widgets (eg. Buttons)

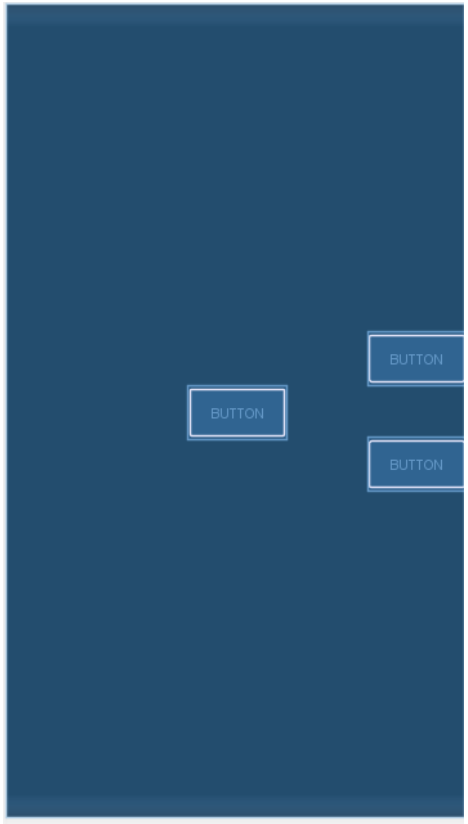
- *android:layout_gravity:*
  adjust individual widgets within the layout



Linear Layout: Gravity

Alignment direction that widgets are pulled

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical"
              android:gravity="center|right">
    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
             android:id="@+id/v_1"     />
    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
             android:id="@+id/v_2“     />

    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
             android:id="@+id/v_3"/>

</LinearLayout>
```

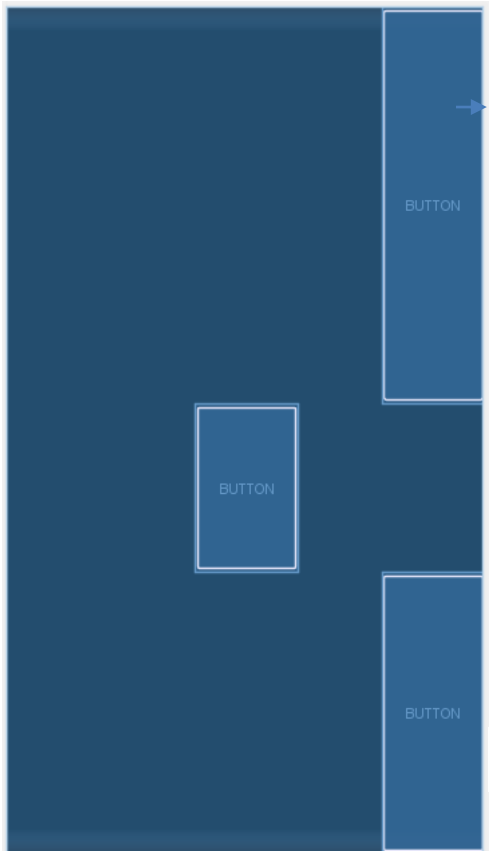


Linear Layout: Weight

Gives elements relative sizes by integers

- widget with weight K gets K/total fraction of total size

Can you guess the weights?




```
android:layout_weight="3"
```

```
android:layout_weight="1"
```

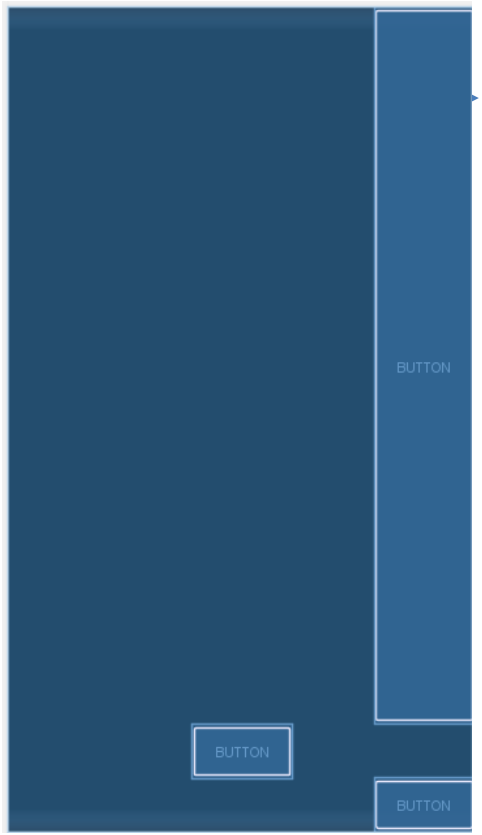
```
android:layout_weight="2"
```

Linear Layout: Weight

Gives elements relative sizes by integers

- widget with weight K gets K/total fraction of total size

Can you guess the weights?

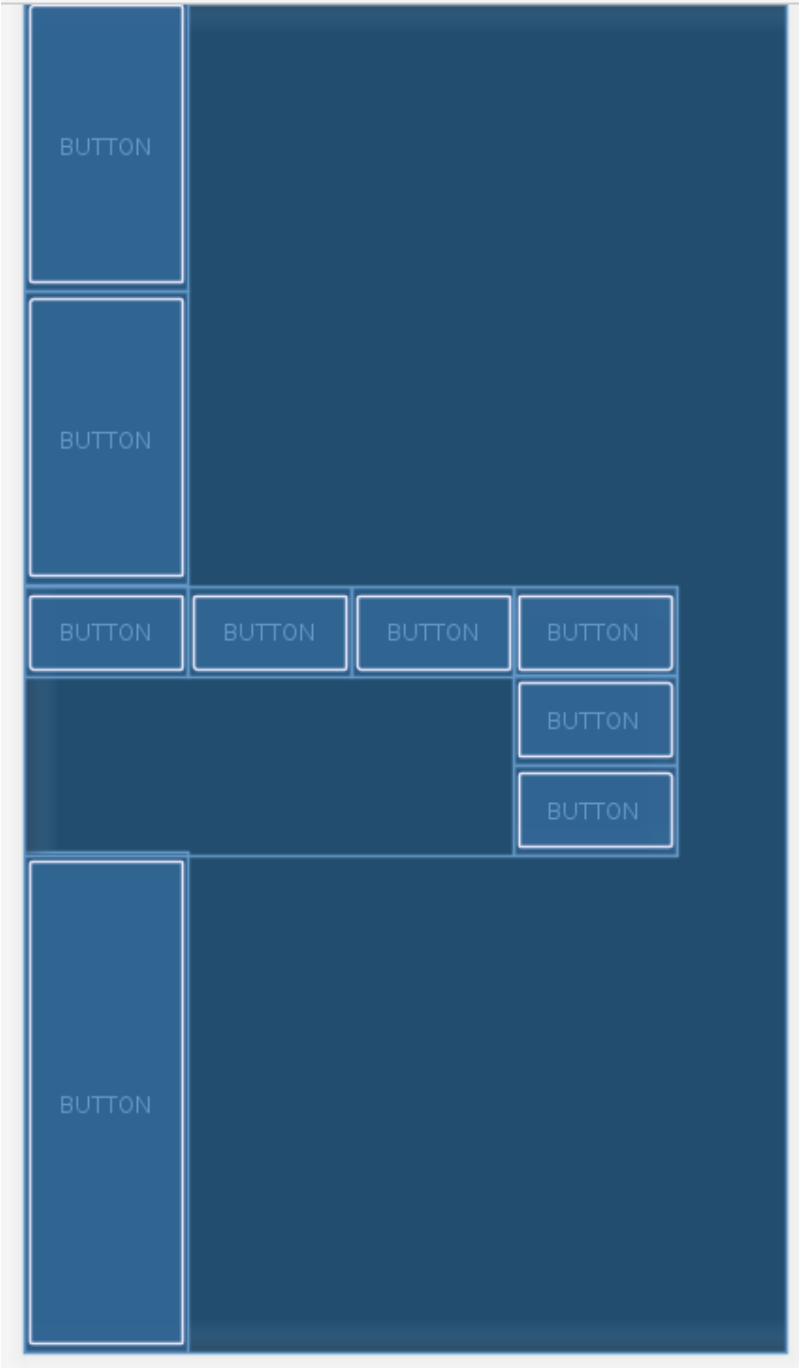



```
android:layout_weight="1"
```

```
android:layout_weight="0"
```

```
android:layout_weight="0"
```

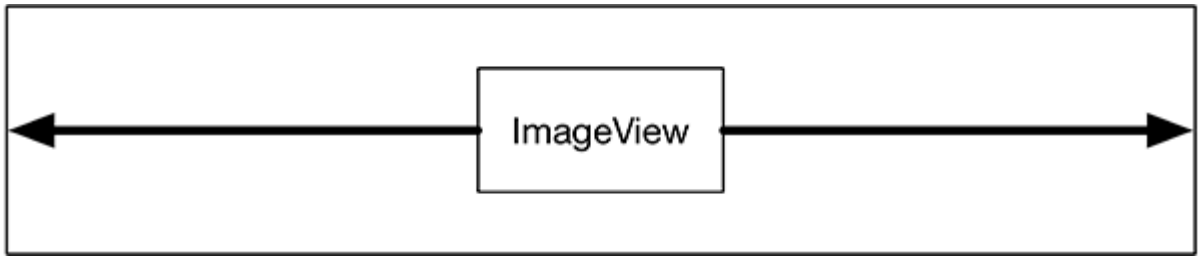
How to generate this layout?



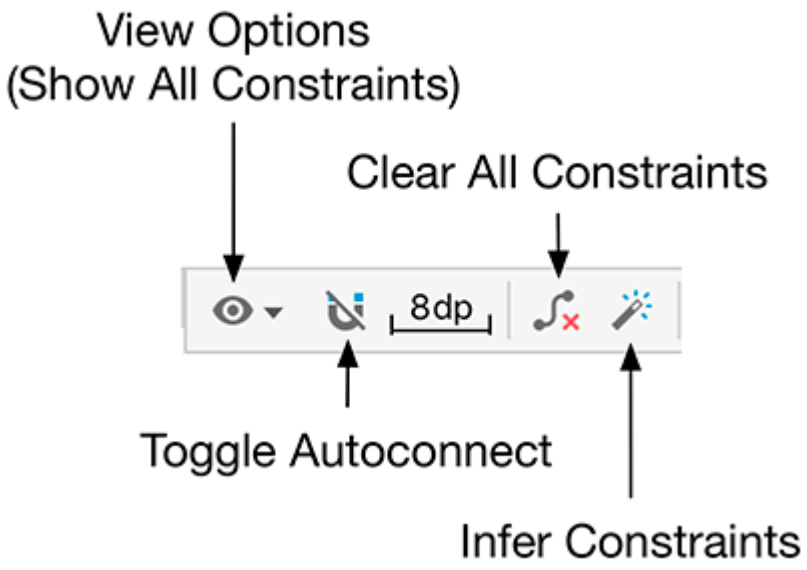
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical">
    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/v_1" android:layout_weight="1“  />
    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/v_2" android:layout_weight="1"/>
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" >
        <Button
                android:text="Button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:id="@+id/vh_1" android:layout_weight="1"/>
        <Button
                android:text="Button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:id="@+id/vh_2" android:layout_weight="1"/>
        <Button
                android:text="Button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:id="@+id/vh_3" android:layout_weight="1"/>
        <LinearLayout
                android:orientation="vertical"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" >
            <Button
                    android:text="Button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content" android:id="@+id/vhv_1" android:layout_weight="1"/>
            <Button
                    android:text="Button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content" android:id="@+id/vhv_2" android:layout_weight="1"/>
            <Button
                    android:text="Button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content" android:id="@+id/vhv_3" android:layout_weight="1"/>
        </LinearLayout>
    </LinearLayout>
    <Button
            android:text="Button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/v_3"
            android:layout_weight="2" />
</LinearLayout>
```

Constraint Layout

A constraint is like a rubber band: It pulls two things toward each other.



Constraint Options

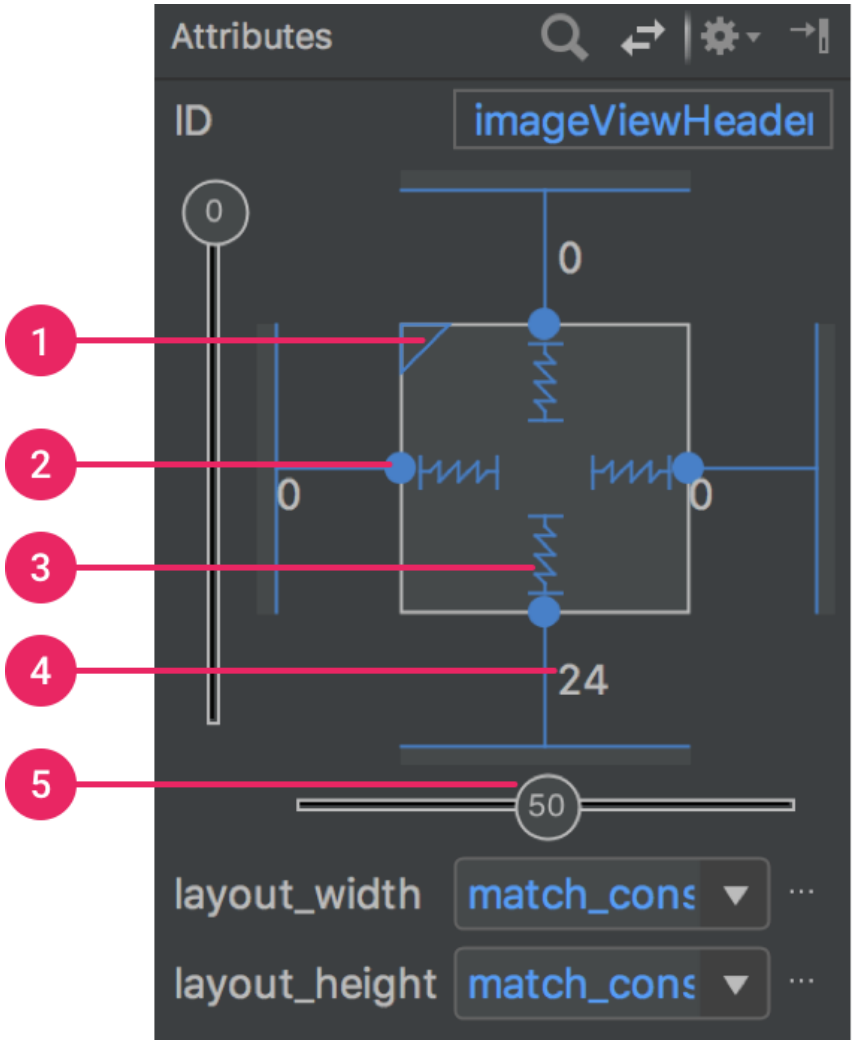


View Options	View Options → Show All Constraints reveals the constraints that are set up in the preview and Blueprint views. You will find this option helpful at times and unhelpful at others. If you have many constraints, this setting will trigger an overwhelming amount of information. View Options includes other useful options, such as Show Layout Decorations. Selecting Show Layout Decorations displays the app bar as well as some system UI (such as the status bar) the user sees at runtime. You will learn more about the app bar in Chapter 14.
Toggle Autoconnect	When autoconnect is enabled, constraints will be automatically configured as you drag views into the preview. Android Studio will guess the constraints that you want a view to have and make those connections on demand.
Clear All Constraints	This button removes all existing constraints in the layout file. You will use this shortly.
Infer Constraints	This option is similar to autoconnect in that Android Studio will automatically create constraints for you, but it is only triggered when you select this button. Autoconnect is active any time you add a view to your layout file.

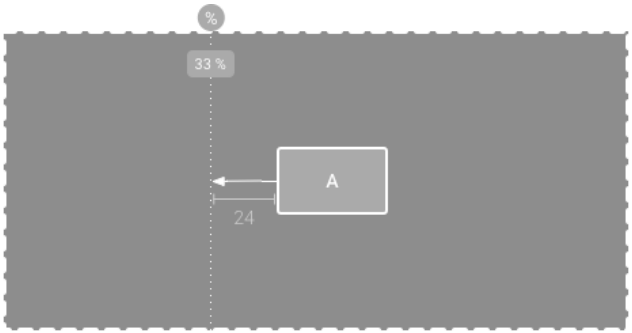
Constraint Layout: Attributes

The **Attributes** window includes controls for

1. size ratio,
2. Delete constraint,
3. height/width mode
4. Margins,
5. constraint bias.



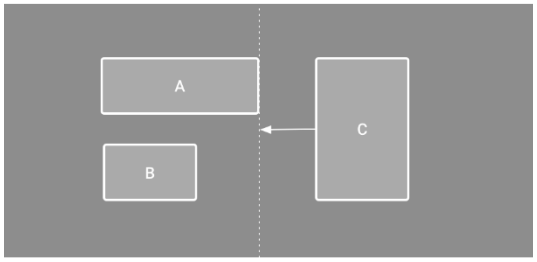
Constraint Layout

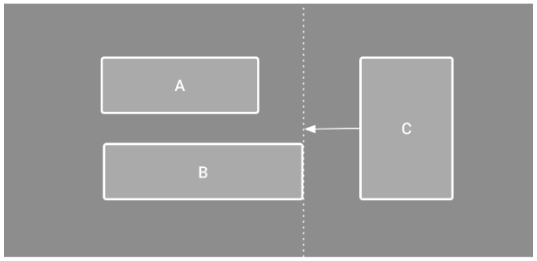


Guideline

Add a vertical or horizontal guideline to which you can constrain views, and the guideline will be invisible to app users.

To create a guideline, click Guidelines in the toolbar, and then click either Add Vertical Guidelineor Add Horizontal Guideline.



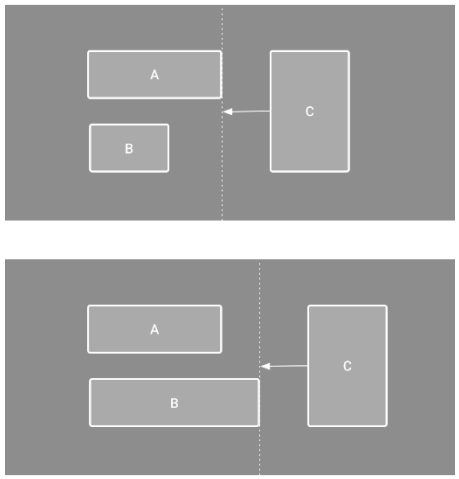


Barrier

A barrier does not define its own position; instead, the barrier position moves based on the position of views contained within it

Constraint Layout: Barrier

A **Constrain to a Barrier** is useful when you want to constrain a view to the a set of views rather than to one specific view.



- View C is constrained to the right side of a barrier.
- The barrier is set to the "end" of both A and B.
- The barrier moves depending on whether the right side of view A or view B is is farthest right.

To create a barrier

1. Click **Guidelines** and then click **Add Vertical Barrier** or **Add Horizontal Barrier**.
2. In the **Component Tree** window, select the views you want inside the barrier and drag them into the barrier component.
3. Select the barrier from the **Component Tree**, open the **Attributes** window, and then set the **barrierDirection**.

Constraint Layout: Chains

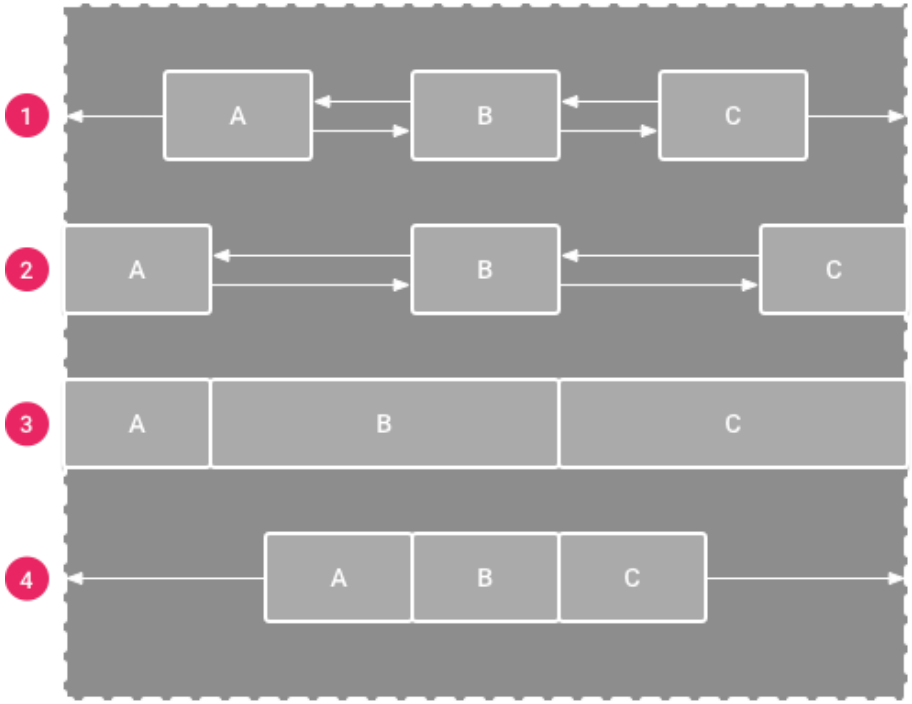
Chains can be styled in one of the following ways:

1.Spread: The views are evenly distributed (after margins are accounted for).
This is the default.

2.Spread inside: The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed.

3.Weighted: the view with the highest weight value gets the most amount of space; views that have the same weight get the same amount of space.

4.Packed: The views are packed together (after margins are accounted for). You can then adjust the whole chain's bias (left/right or up/down)



Constraint Layout: automatic

Instead of adding constraints to every view as you place them in the layout, you can move each view into the positions you desire, and then click **Infer Constraints** to automatically create constraints.

- **Infer Constraints** scans the layout to determine the most effective set of constraints for all views.
- It makes a best effort to constrain the views to their current positions while allowing flexibility.
- You might need to make some adjustments to be sure the layout responds as you intend for different screen sizes and orientations.

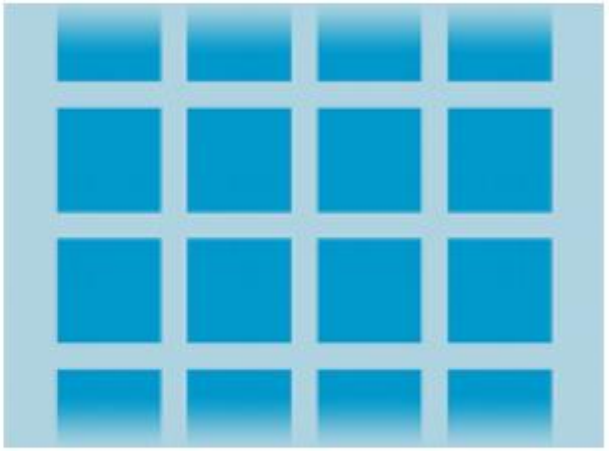
Layout with adapter

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses AdapterView to populate the layout with views at runtime

List View

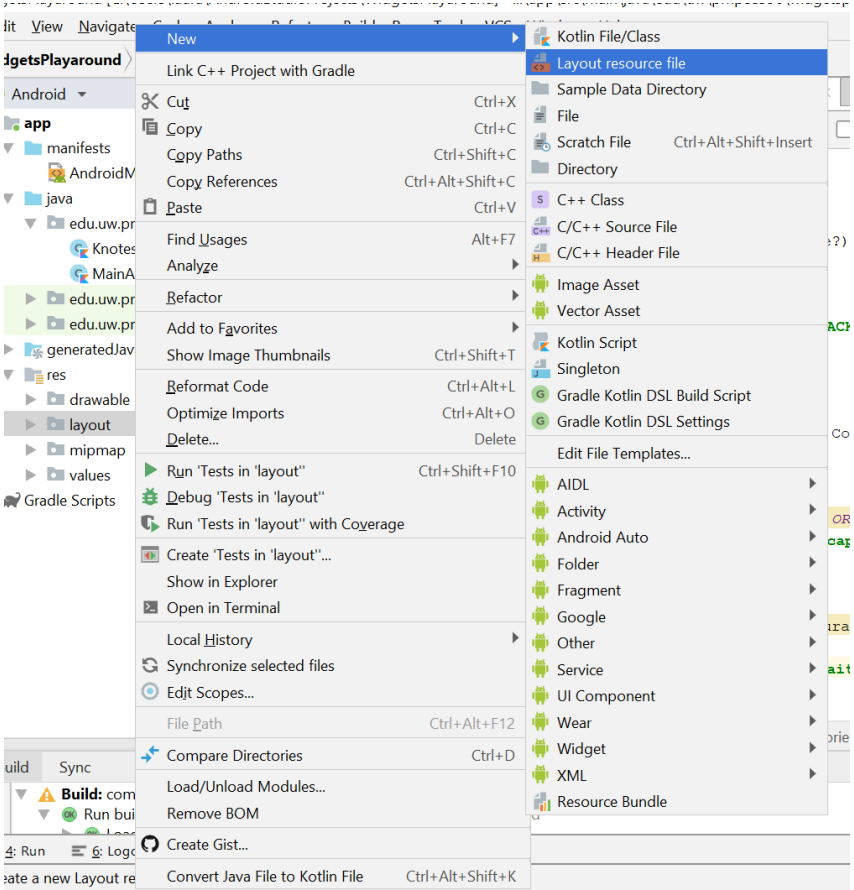
Displays a scrolling single column list.

Grid View

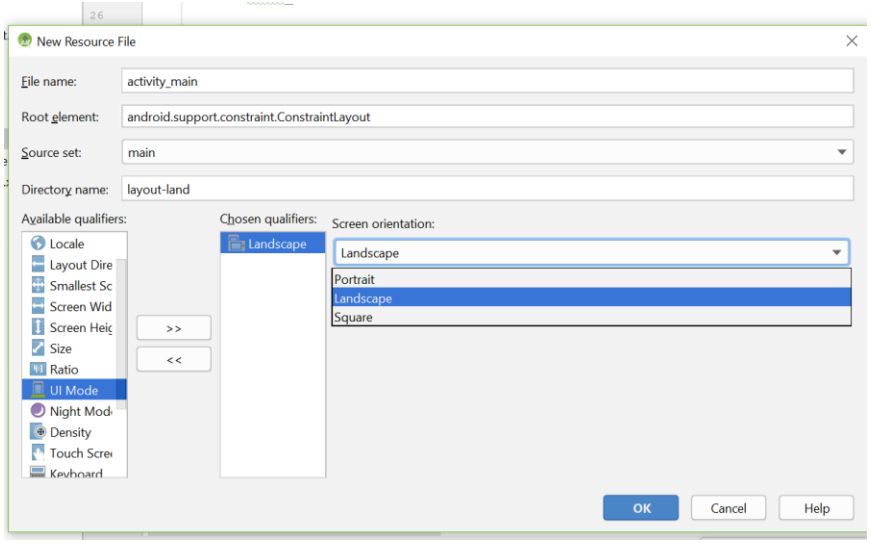


Displays a scrolling grid of columns and rows.

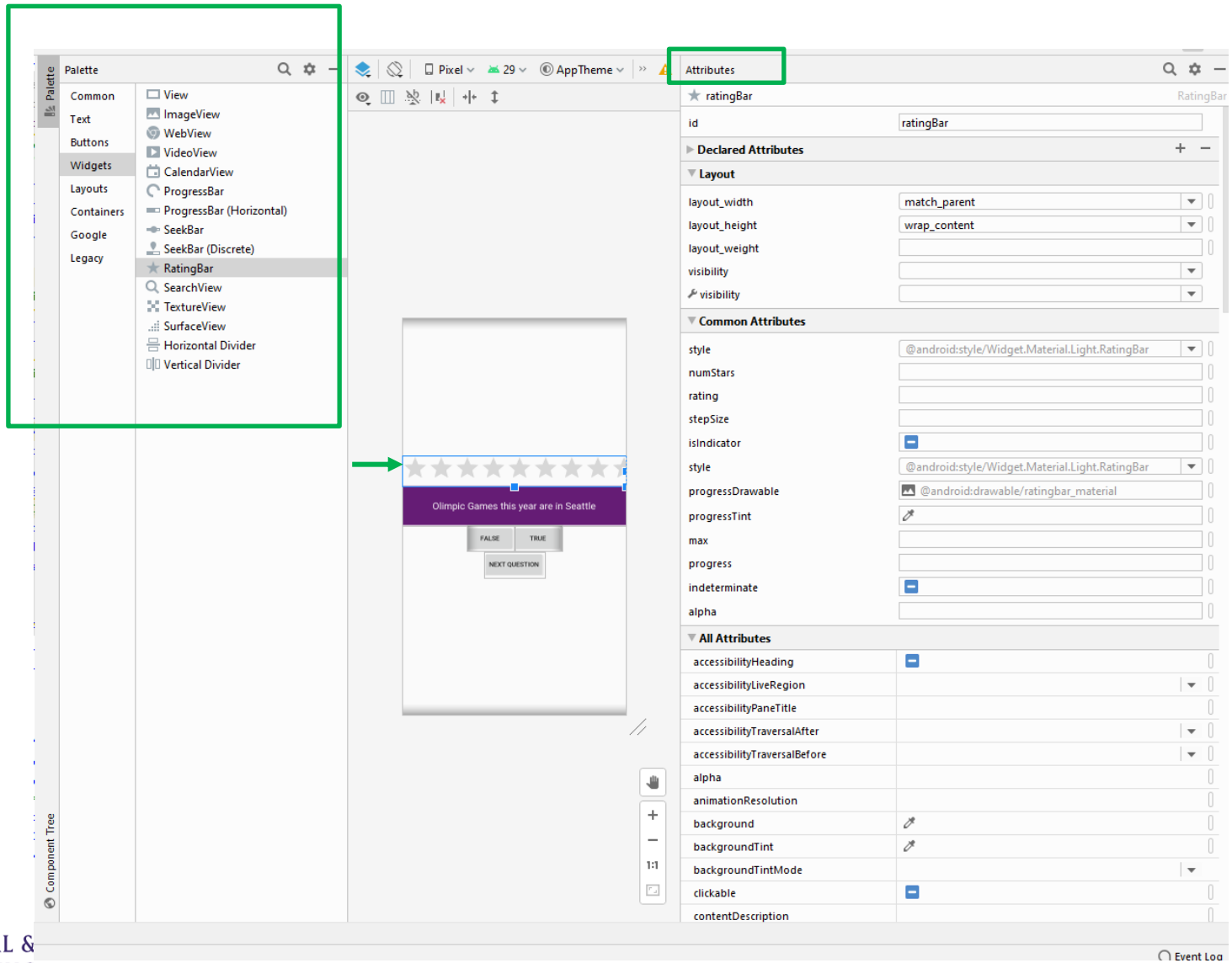
Landscape Layout



- Name the landscape xml with the same name as the portrait xml
- Android will pick the correct xml file automatically depending on the screen orientation



Widgets (View objects)



Widgets Atrributes (I)

`android:layout_width` and **`android:layout_height`**

The `android:layout_width` and `android:layout_height` attributes are required for almost every type of widget.

They are typically set to either `match_parent` or `wrap_content`:

`match_parent`	view will be as big as its parent
`wrap_content`	view will be as big as its contents require

Widgets Atrributes (II)

`android:orientation`

attribute on the two LinearLayout widgets determines whether their children will appear vertically or horizontally.

- The order in which children are defined determines the order in which they appear onscreen.
- In a vertical LinearLayout, the first child defined will appear topmost.
- In a horizontal LinearLayout, the first child defined will be leftmost.(Unless the device is set to a language that runs right to left, such as Arabic or Hebrew. In that case, the first child will be rightmost.)

`android:visibility="visible“`

- set widgets to be “visible” or “invisible”
- you can set the initial (default) value in the xml fine and
  then change the value of this attribute from the Activity as a response to an event

Widgets Atrributes (III)

`android:text`

The TextView and Button widgets have `android:text` attributes.

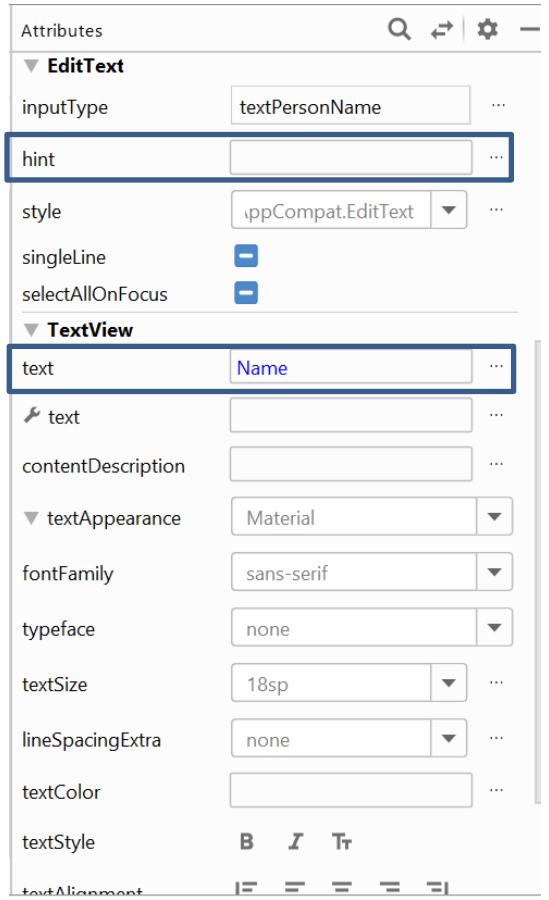
- This attribute tells the widget what text to display.

- Notice that the values of these attributes are not literal strings.
- They are references to string resources, as denoted by the @string/ syntax.

- A **string resource** is a string that lives in a separate XML file called a strings file.

- You can give a widget a hardcoded string, like android:text="True", but it is usually not a good idea. Placing strings into a separate file and then referencing them is better because it makes localization (which you will learn later in the course) easy.

Widgets: EditText



Example to read the text entered in a EditText from a user:

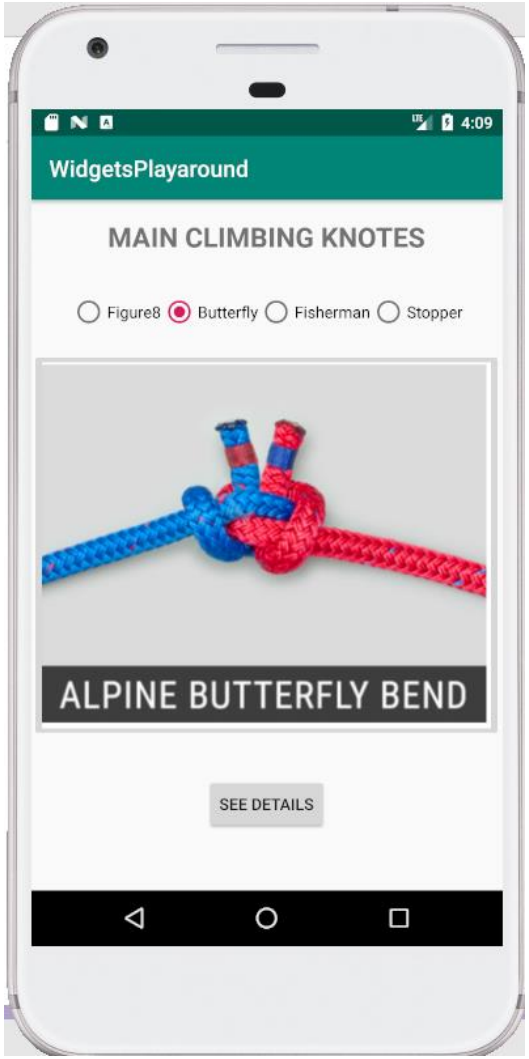
2 alternatives to refer to control the widget form the activity.

```
val editText = findViewById<EditText>(R.id.editText)
val message = editText.text.toString()
```

equivalent

```
import kotlinx.android.synthetic.main.activity_main.*

val message = editText.text.toString()
```

Widgets: RadioGroup




```
private fun updateKnoteImage(view:View) {
    val id = when (view) {
        figure8 -> R.drawable.figure8
        butterfly -> R.drawable.butterfly
        fisherman -> R.drawable.fisherman
        stopper -> R.drawable.stopper
        else
            -> R.drawable.figure8
    }
    knote.setImageResource(id)
}
```

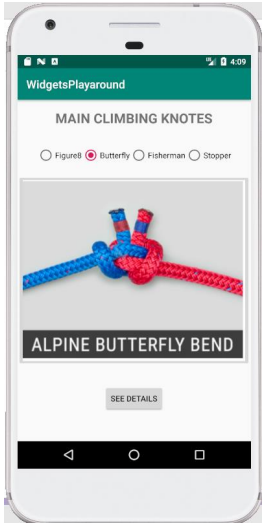
When expression

Replaces the ***switch*** operator of Java

In the simplest form it looks like this

```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> {
        print("x is neither 1 nor 2")
    }
}
```

Widgets: RadioGroup




```
<ImageButton
        android:id="@+id/knote"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/butterfly"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.41000003"
/>
```

Called when user selects a radio button

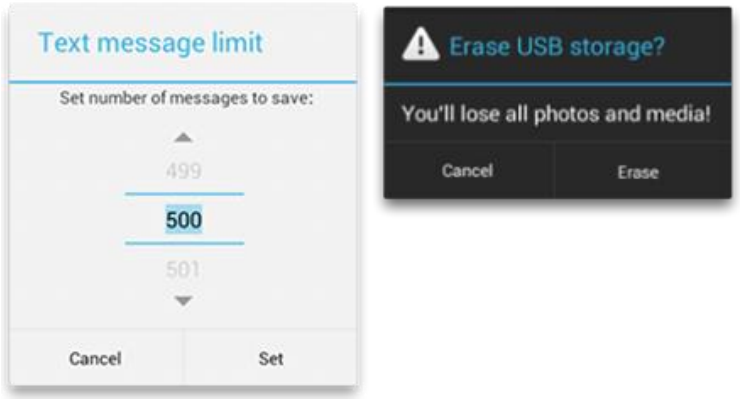
```
private fun updateKnoteImage(view:View) {
    val id = when (view) {
        figure8 -> R.drawable.figure8
        butterfly -> R.drawable.butterfly
        fisherman -> R.drawable.fisherman
        stopper -> R.drawable.stopper
        else
             -> R.drawable.figure8
    }
    knote.setImageResource(id)
}
```

Widgets: CheckBox

```
Refer to Lecture 2 class activity
```

Dialogs

- A dialog is a small window that prompts the user to make a decision or enter additional information.



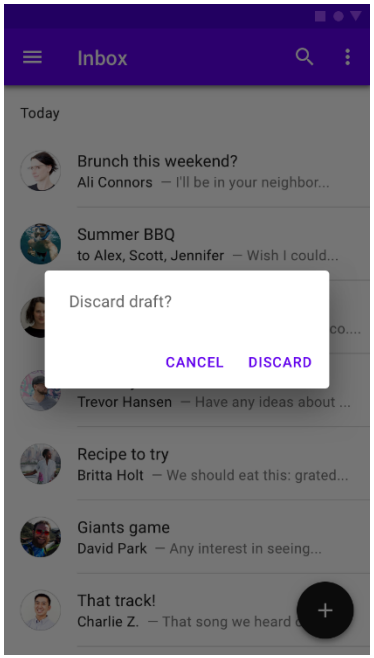
- A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed

- Dialogs are purposefully interruptive, so they should be used sparingly.

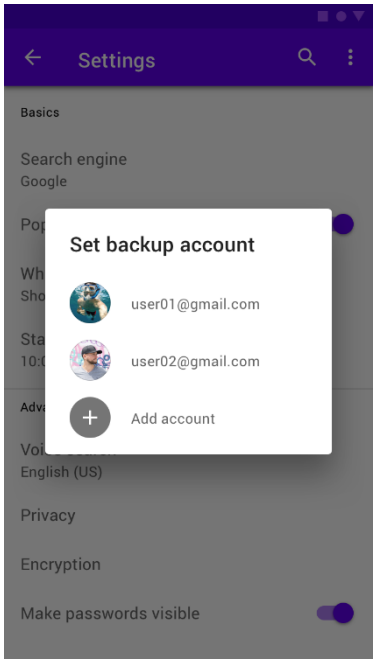
- Dialogs should be used for:
  - Errors that block an app's normal operation
  - Critical information that requires a specific user task, decision, or acknowledgement

Types of Dialogs

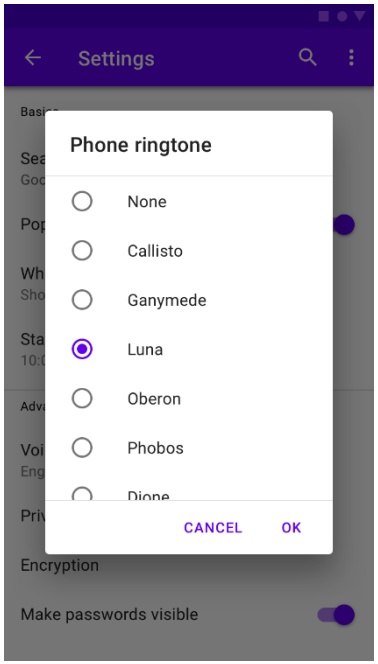
Alert



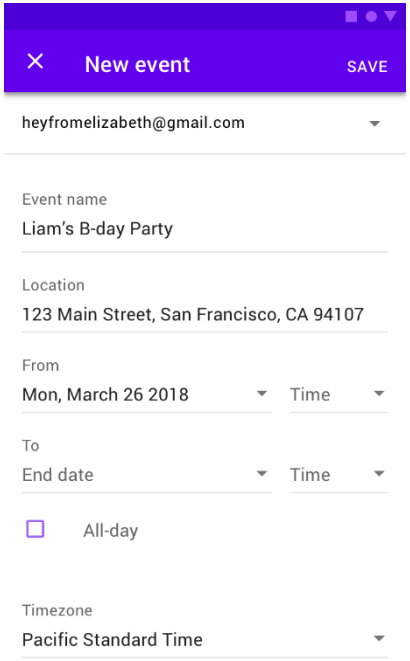
Simple



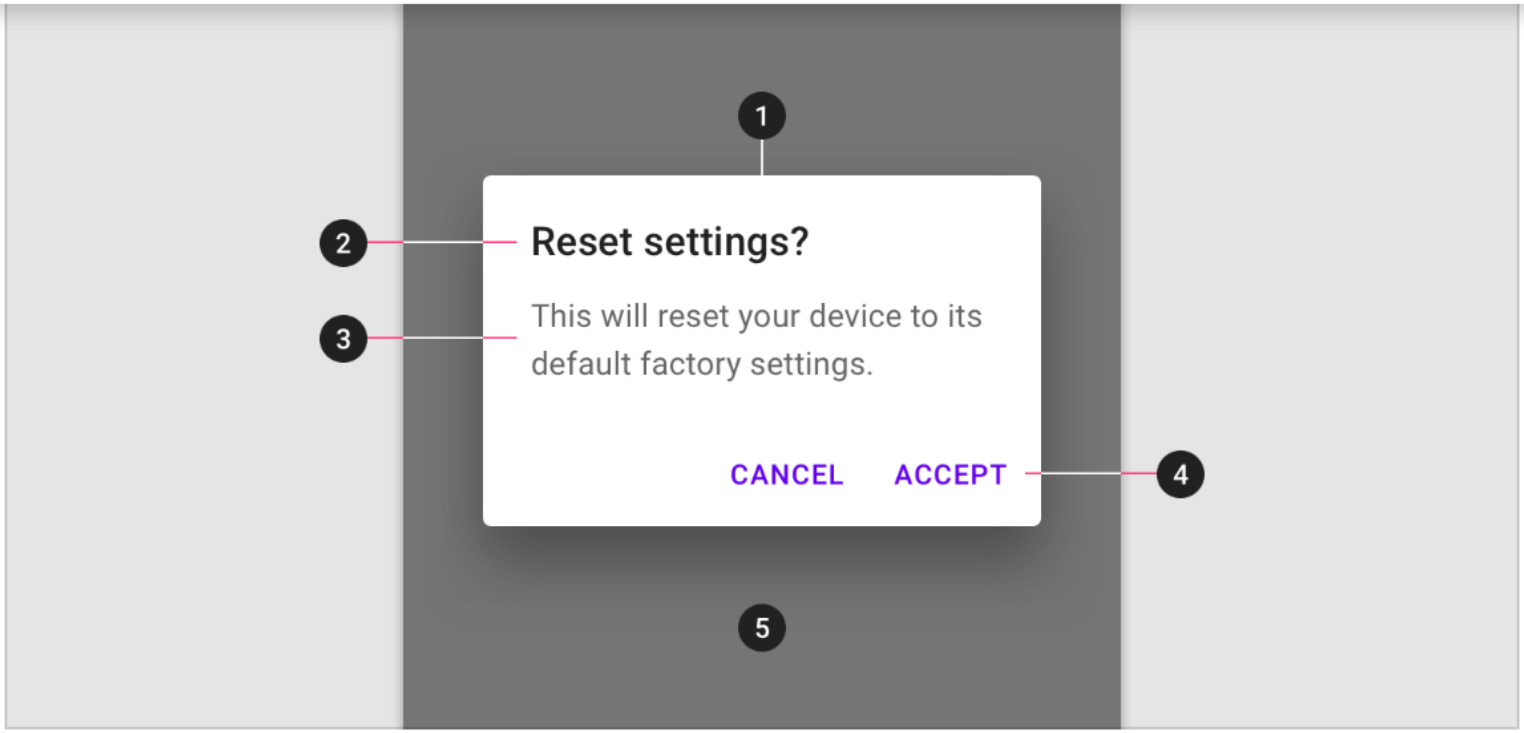
Confirmation



Full Screen



Dialogs: Anatomy



1. Container
2. Title (optional)
3. Supporting text
4. Buttons
5. Scrim

AlertDialog

There are three different action buttons you can add:

- **Positive**
  You should use this to accept and continue with the action (the "OK" action).

- **Negative**
  You should use this to cancel the action.

- **Neutral**
  You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel.
  It appears between the positive and negative buttons.
  For example, the action might be "Remind me later."

You can add only one of each button type to an AlertDialog. That is, you cannot have more than one "positive" button.

Build an Alert Dialog

1. Create a dialog in your activity class with dialog *builder*
2. The builder has many *set* methods to customize the dialog
3. When ready, *create()* the dialog and *show()* it
4. You can attach listener to the buttons

```
//make a dialog

val mDialog = AlertDialog.Builder(this)
mDialog.setTitle("This is the title")
mDialog.setMessage("This is a Dialog!")
mDialog.setPositiveButton("OK"){_, _ ->} //you can attach a
listener to respond to the OK button
mDialog.show()
```

Build an Alert Dialog

1. Create a dialog in your activity class with dialog *builder*
2. The builder has many *set* methods to customize the dialog
3. When ready, *create()* the dialog and *show()* it
4. You can attach listener to the buttons

```
//make a dialog

val mDialog = AlertDialog.Builder(this)
mDialog.setTitle("This is the title")
mDialog.setMessage("This is a Dialog!")
mDialog.setPositiveButton("OK"){_, _ ->} //you can attach a
listener to respond to the OK button
mDialog.show()
```

Using String Resources

Define the string in the *resources/Strings* directory

```
<resources>
    <string name="app_name">myKnotes</string>
    <string name="initial_info">Click see details to add your notes here</string>
</resources>
```

Create a widget/element to write the string

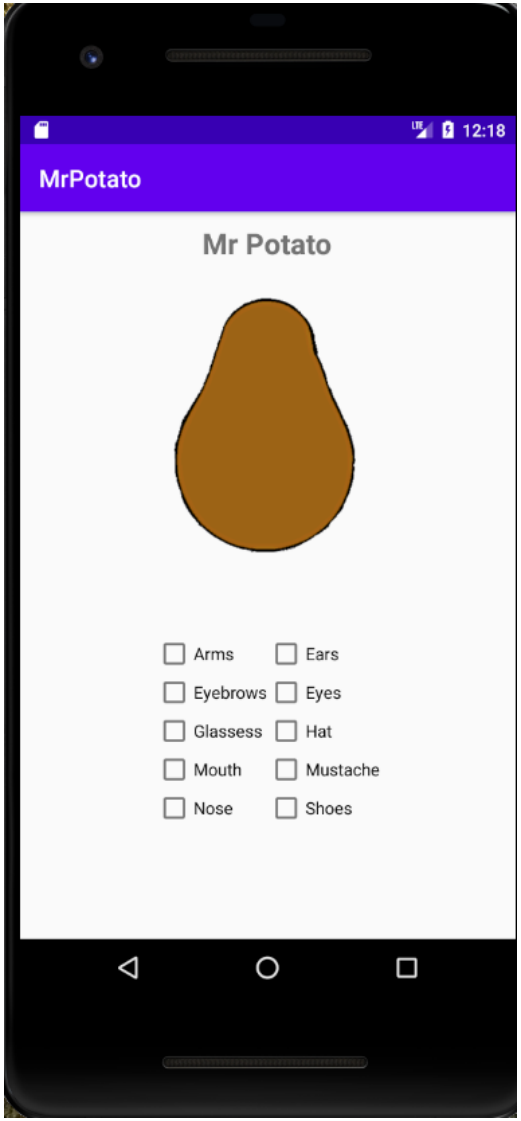
```
<TextView
    android:id="@+id/user_info"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content“
          …
/>
```

Recover the string defined in the Resources directory

```
user_info.text=getString(R.string.initial_info)
```

Practice time!

Practice time!



15min Break


ONE PER WORKING HOUR
TO KEEP YOU MOVING
1ST HOUR
Touch your toes
5 seconds, 3 times
2ND HOUR
Tip-toes and stretch up
5 seconds, 3 times
3RD HOUR
The twist
5 seconds, 3 times
4TH HOUR
The March
Knee towards chest. 10 times
5TH HOUR
Neck stretch
Turn head backwards and sideways. 5 seconds, 3 times
6TH HOUR
The Ballerina
Feet apart. Arm over head reach towards opposite side.
5 seconds, 3 times
7TH HOUR
The Hand Hold
Grab hand and pull towards you until you feel stretch.
5 seconds, 3 times
8TH HOUR
The Banana
Lean, arching your back.
5 seconds, 3 times
9TH HOUR
The Shrug
Roll your shoulders. 5 times
10TH HOUR
Scratch my back
Touch opposite shoulder blade.
5seconds, 3 times
11TH HOUR
Squat
10 times.
12TH HOUR
The penguin
Walk sideways around chair without swaying.
3 times
SIT LESS
MOVE MORE OFTEN
1.Drink More
2.Stand & Call
3.Read Standing
4.Take Stairs
5.Park & Walk

Multiple Activities

Many apps have **multiple activities**.

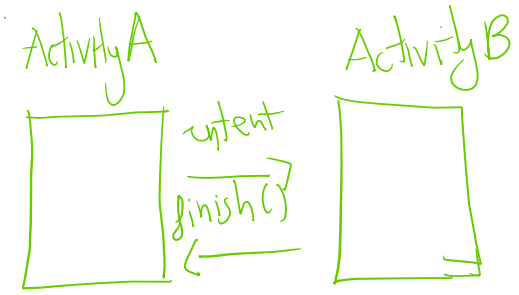
- An activity A can launch another activity B in response to an event.
- The activity A can pass data to B.
- The second activity B can send data back to A when it is done.
- Example: class Appp KnowYourKnotes

Class example

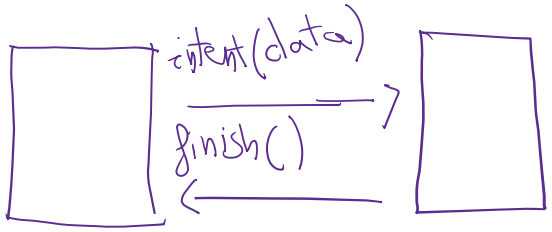
Multiple Activities

We are going to analyze 3 scenarios:

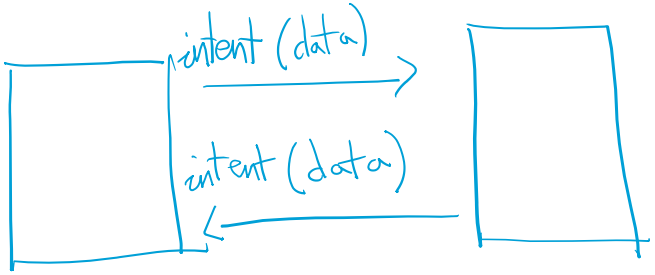
SCENARIO 1



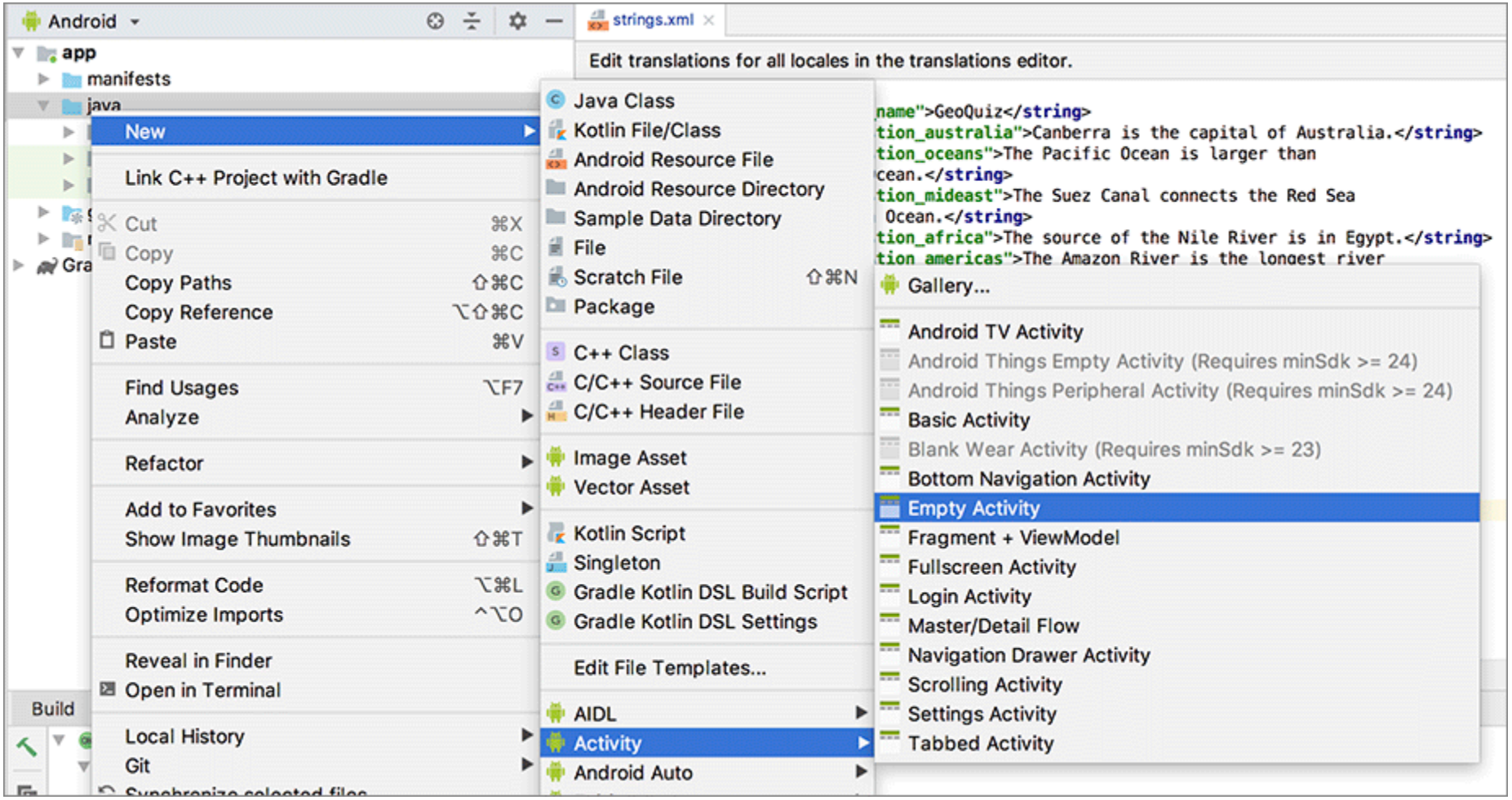
SCENARIO 2



SCENARIO 3

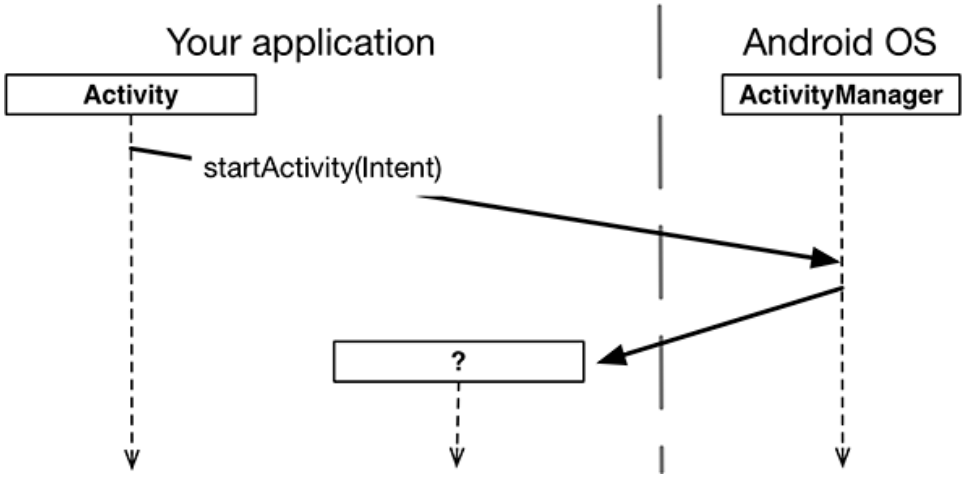


Start an activity



Start an activity: Intents

- The simplest way one activity can start another is with the `startActivity(Intent)` function.



- You might guess that `startActivity(Intent)` is a static function that you call on the Activity subclass that you want to start. But it is not.

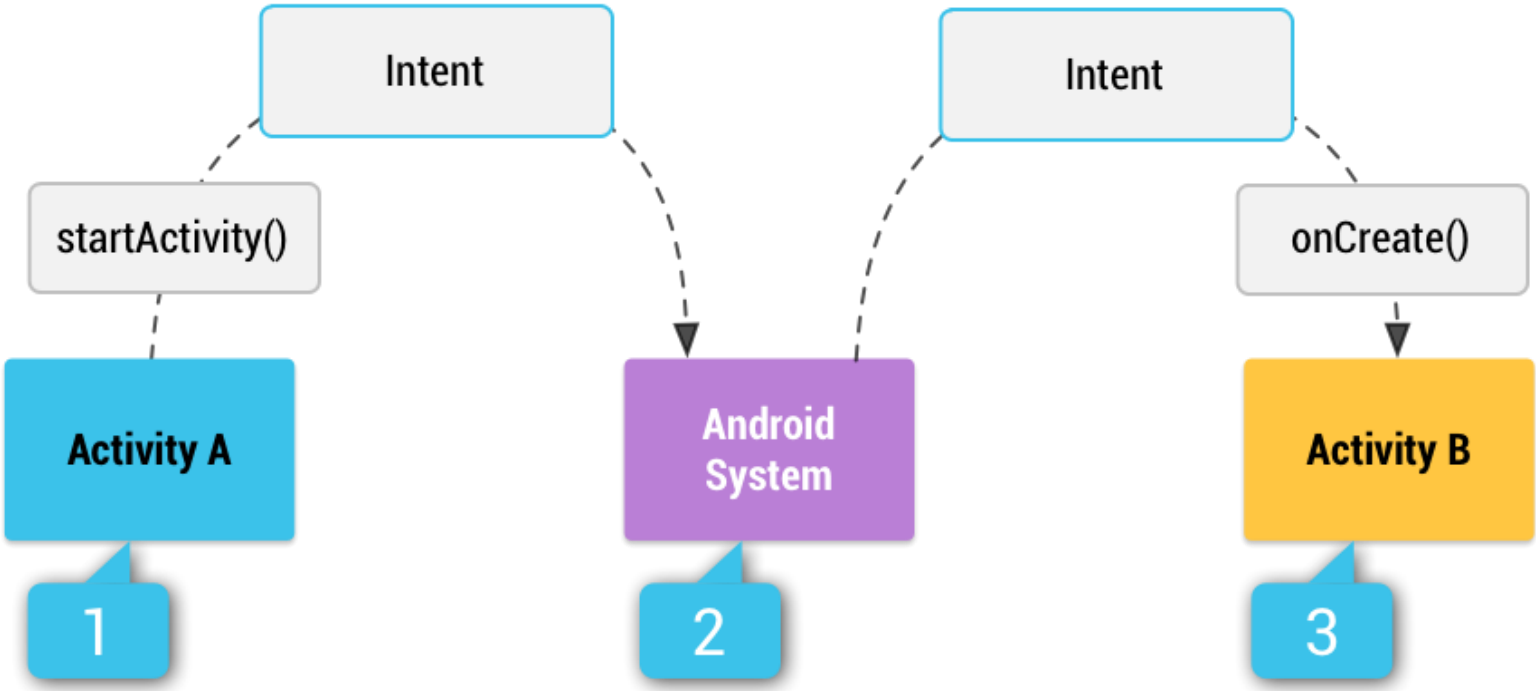
- When an activity calls `startActivity(Intent),` this call is sent to the OS. In particular, it is sent to a part of the OS called the ActivityManager. The ActivityManager then creates the Activity instance and calls its `onCreate(Bundle?)` function,

Multiple Activities: Intents

An ***Intent*** is a messaging object you can use to request an action from another app component.

- **Explicit intents**
  - Specify which application will satisfy the intent
  - You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start
- **Implicit intents**
  - Do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
  - Example: show the user a location on a map: use an implicit intent to request that another capable app show a specified location on a map.

Implicit Intent

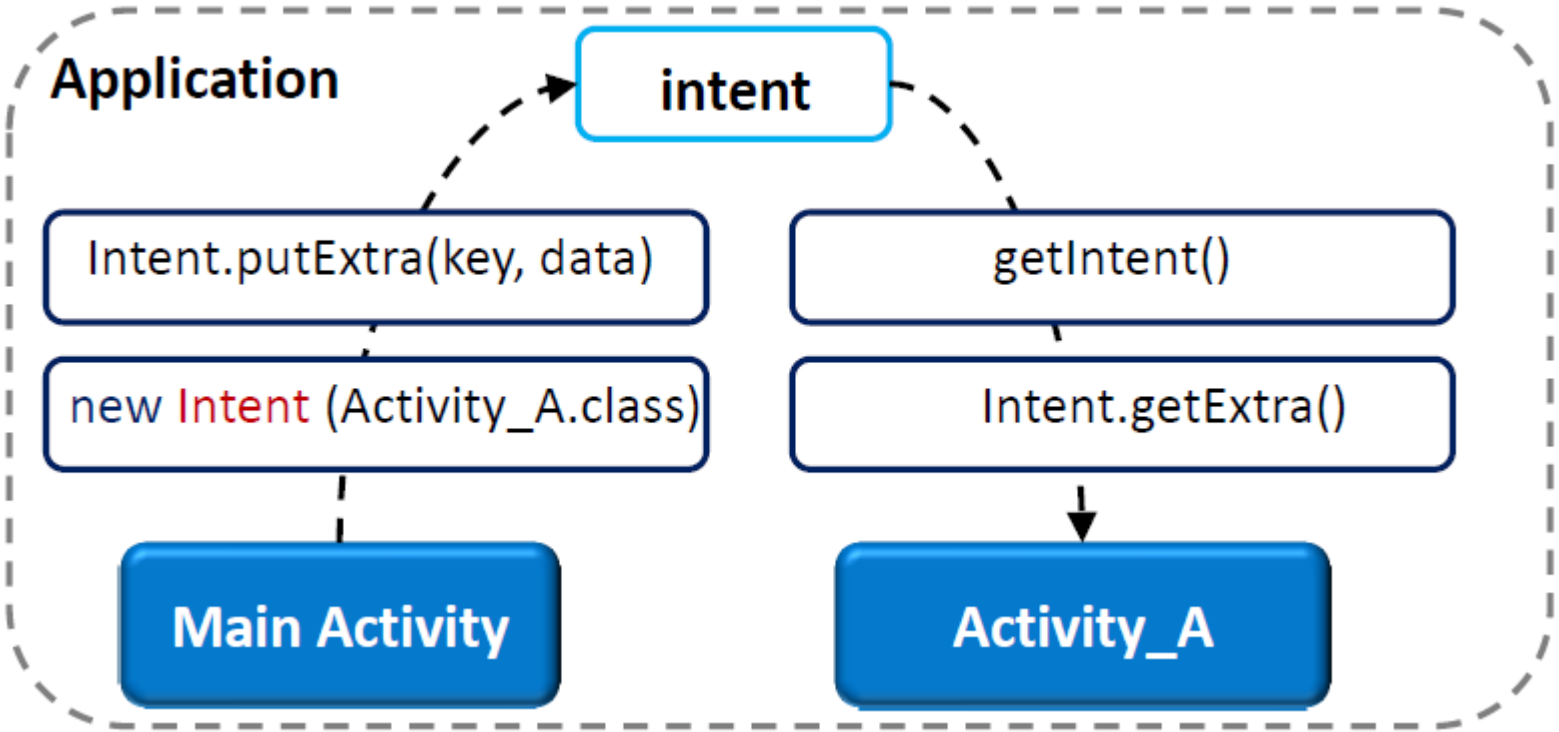


Activity A creates an Intent with an action description and passes it to startActivity().

The Android System searches all apps for an **intent filter** that matches the intent.

Match found: system starts the matching activity (Activity B) by invoking its onCreate() method and passing it the Intent.

Explicit Intent



Intent.putExtra(…) comes in many flavors, but it always has two arguments.

- key: always a String key,
- data: type will vary. It returns the Intent itself, so you can chain multiple calls if you need to.

SCENARIO 1

Start a second activity

MainActivity.kt


```
const val KEY = “myMessage" //any String that you define

fun sendMessage(view: View) {
        val intent = Intent(this, SecondActivity::class.java)
        startActivity(intent)

        }

Wee need to pass the text entered by the user to the second activity!
```

1. Create an intent

2. Launch the intent

SCENARIO 2

Pass data to second activity

MainActivity.kt


```
const val KEY = "myMessage" //any String that you define

fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editText)
        val message = editText.text.toString()
        val intent = Intent(this, SecondActivity::class.java)
        intent.putExtra(KEY, message)

        startActivity(intent)
}
```

2. Put data into the intent

3. Launch the intent

SCENARIO 2

Pass multiple data to second activity

```
const val KEY = “myMessage" //any String that you define

fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editText)
        val message = editText.text.toString()
        val intent = Intent(this, SecondActivity::class.java)
        intent.putExtra(KEY1, message1)
        intent.putExtra(KEY1, message2)
        intent.putExtra(KEY3, message3)


        startActivity(intent)

}
```

Passing data beteween activities

Extras

- arbitrary data that the calling activity can include with an intent.
- You can think of them like constructor arguments, even though you cannot use a custom constructor with an activity subclass.
- The OS forwards the intent to the recipient activity, which can then access the extras and retrieve the data,

```
Intent.putExtra(…)
```

SCENARIO 2

Receive and extract data


```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    setContentView(R.layout.activity_second)

      //To retrieve an Integer
       val received = intent.getIntExtra("myMessage")
      //To retrieve a String
       val received = intent.getStringExtra("myMessage")
```

SCENARIO 3

Get result back to first activity

SCENARIO 3

Summary Scenario 3

1. ActivityA starts intent with startActivItyForResult() function

```
startActivityForResult(intent, requestCode)
```

2. ActivityB: receive data, and send other data to ActivityA using setResult()

```
setResult(Activity.RESULT_OK, intent)
```

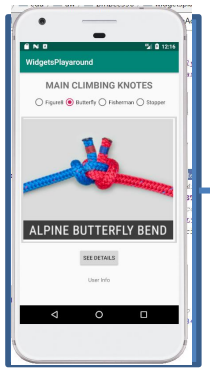
3. Activity3: receives data from ActivityB following these steps
   i. Override the function onActivityResult()
   ii. Check the requestCode
   iii. Check the request from previous activity was successful
   iv. Retrieves data from the intent

```
override fun onActivityResult(requestCode:Int, resultCode:Int, data: Intent?) {
    // Check which request we're responding to
    if (requestCode == PICK_KNOTE_REQUEST) {
        // Make sure the request was successful
        if (resultCode == Activity.RESULT_OK) {
            val data_from_ActivityB= data!!.getStringExtra("KEY_D")
}
    }
```

SCENARIO 3

Explicit Intent: get restult from activity

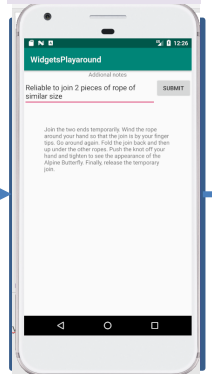
ActivityA.kt




```
intent.putExtra("KEY", message)
startActivityForResult(intent, request_code)
```

1. In response to some event: *start* ***ActivityB*** and expect some data back

ActivityB.kt



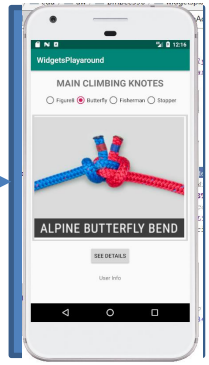

```
intent.putExtra("KEY_B", message)
setResult(Activity.RESULT_OK, intent)
finish()
```

2. Receive data from ***ActivityA***

```
intent.getIntExtra("KEY")
        Or
intent.getStringExtra("KEY")
```

3. Do something with it
4. Pass data to ***ActivityA***
5. Return to ***ActivityA.***

ActivityA.kt



Receive data from Activity B

```
fun onActivityResult(…) {
   intent.getIntExtra("KEY_B")
     ……
 }
```

Activities: Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.uw.pmpee590.exampleapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:configChanges="orientation|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SecondActivity"></activity>
    </application>

</manifest>
```

Launch main activity

How de we tell Android which activity to run/display when the App is first launched?

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="edu.uw.pmpee590.myapplication">

    <application
            android:allowBackup="true"
            android:icon="@mipmap/ic_launcher"
            android:label="My Application"
            android:roundIcon="@mipmap/ic_launcher_round"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Use the camera: Implicit Intent

1) **Implicit Intent**
   Delegating the work to another camera app on the device

   You want to take photos with minimal fuss, not reinvent the camera.
   Happily, most Android-powered devices already have at least one camera application installed.

```
val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
startActivityForResult(takePictureIntent,REQUEST_TAKE_PHOTO)
```

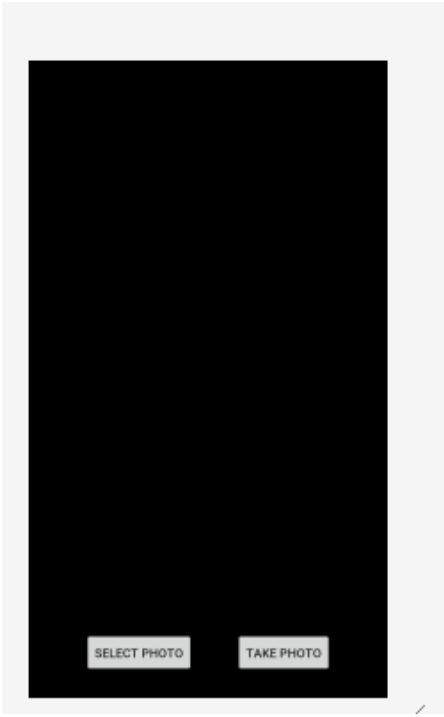
2) **Custom Camera control**
   Control the camera hardware directly using the framework APIs.
   Build a specialized camera application or something fully integrated in your app UI.

Implicit Intent: Take Pictures

1. **Ask the user to give permissions to use camera and save pictures**

in the onCreate function – **details in the takePicture App on Canvas**

```
getRuntimePermissions()
```

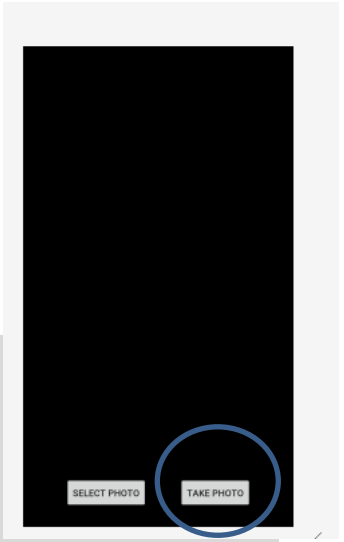


AND in the manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.uw.eep523.takepicture">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
```

Implicit Intent: Take Pictures

2.A) Create an implicit intent to launch the CAMERA and take picture




```
fun startCameraIntentForResult(view:View) {
    // Clean up last time's image
    imageUri = null
    val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)

        takePictureIntent.resolveActivity(packageManager)?.let {
         val values = ContentValues()
         values.put(MediaStore.Images.Media.TITLE, "New Picture")
         values.put(MediaStore.Images.Media.DESCRIPTION, "From Camera")
         imageUri = contentResolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI, values)
         takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, imageUri)
         startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE)
    }
}
```

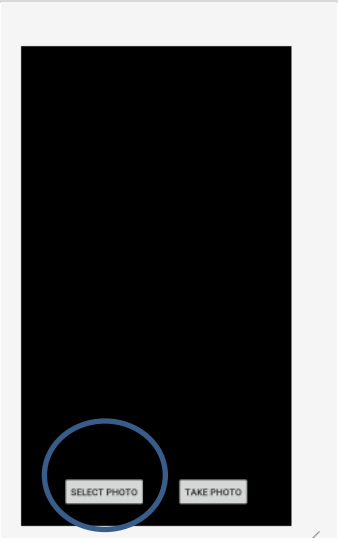
```
private const val REQUEST_IMAGE_CAPTURE = 1001
```

Implicit Intent: Take Pictures

2.B) Create an implicit intent CHOOSE picture from the phone gallery

```
fun startChooseImageIntentForResult(view:View) {
    val intent = Intent()
    intent.type = "image/*"
    intent.action = Intent.ACTION_GET_CONTENT
    startActivityForResult(Intent.createChooser(intent, "Select Picture"), REQUEST_CHOOSE_IMAGE)
}
```

```
private const val REQUEST_CHOOSE_IMAGE = 1002
```



Implicit Intent: Take Pictures

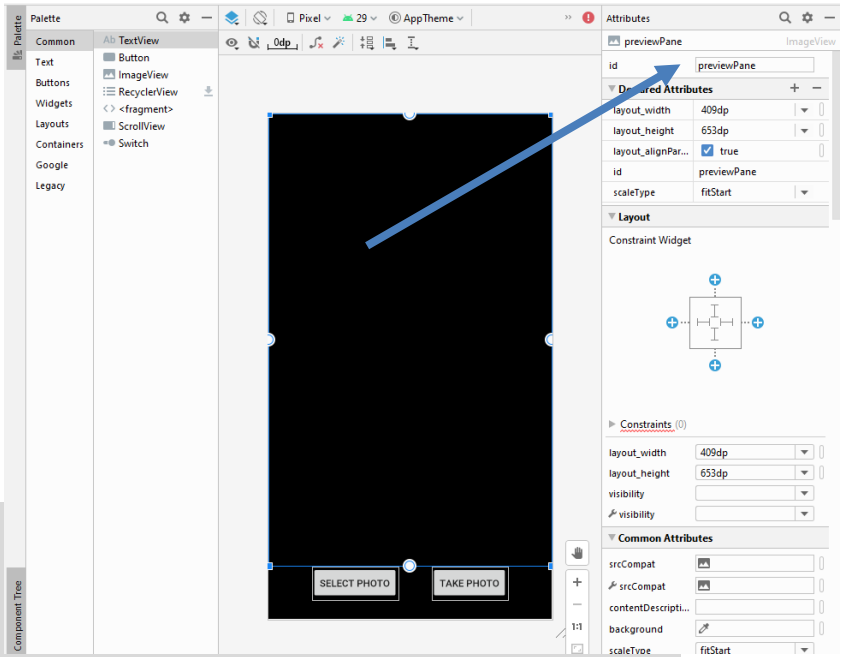
3) Receive data from Implicit Intent: New picture OR picture form gallery

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK) {
        tryReloadAndDetectInImage()
    } else if (requestCode == REQUEST_CHOOSE_IMAGE && resultCode == Activity.RESULT_OK) {
        // In this case, imageUri is returned by the chooser, save it.
        imageUri = data!!.data
        tryReloadAndDetectInImage()
    }
}
```

```
private const val REQUEST_IMAGE_CAPTURE = 1001
private const val REQUEST_CHOOSE_IMAGE = 1002
```

Implicit Intent: Take Pictures

4) Display picture on am ImageView widget




```
private fun tryReloadAndDetectInImage() {
    try {
        if (imageUri == null) {
            return
        }
        val imageBitmap = if (Build.VERSION.SDK_INT < 29) {
            MediaStore.Images.Media.getBitmap(contentResolver, imageUri)
       } else {
            val source = ImageDecoder.createSource(contentResolver, imageUri!!)
            ImageDecoder.decodeBitmap(source)
        }
        previewPane?.setImageBitmap(imageBitmap) //previewPane is the ImageView from the
                                                 layout
    } catch (e: IOException) {
        Log.e(TAG, "Error retrieving saved image")
    }
}
```

Implicit Intent: Take Pictures

What is the image URI ?

Files and Storage

Android provides several options for you to save your app data.

- **Internal file storage**: Store app-private files on the device file system.
- **External file storage**: Store files on the shared external file system. This is usually for shared user files, such as photos.
- **Shared preferences:** Store private primitive data in key-value pairs.
- **Databases**: Store structured data in a private database.

Files and Storage

Internal storage:

- It's always available.
- Files saved here are accessible by only your app.
- When the user uninstalls your app, the system removes all your app's files from internal storage.

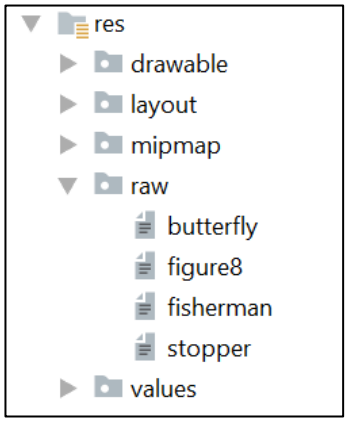
Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage:

- It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
- It's world-readable, so files saved here may be read outside of your control.
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from

Internal Storage

- Create a new resource directory **-> raw**
- Create resource file under ***raw/file_name***




```
//Read all text in the file butterfly.text

val textFiledID = resources.getIdentifier(“butterfly”, "raw", packageName)
val fileRead= resources.openRawResource(textFiledID).bufferedReader().readText()
```

External Storage

If your app needs to read/write the device's external storage, you must explicitly request **permission** to do so in your app's **AndroidManifest.xml** file.

- On install, the user will be prompted to confirm your app permissions.

```
<manifest ...>
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
...
</manifest>
```

https://developer.android.com/training/data-storage/files.html#kotlin

File and Stream Objects

File - **Objects that represent a file or directory.**

– methods: `canRead, canWrite, create, delete, exists, getName, getParent, getPath, isFile, isDirectory, lastModified, length, listFiles, mkdir, mkdirs, renameTo`

InputStream, OutputStream - **flows of data bytes from/to a source or destination**

– Could come from a file, network, database, memory, ...
– Normally not directly used; reading/writing a byte at a time from the input.
– Instead, a stream is often passed as parameter to other objects like
- methods/properties *bufferedReader* to do the actual reading / writing.

Readers and Scanners

File and **InputStream** objects are not usually used directly.
Instead you wrap them In a reader or scanner

BufferedReader – I/O object for reading a line at a time
- methods/properties: `readLine, ready, lineSequence, close`

Scanner – I/O object for reading lines or tokens at a time
- methods/properties: `readLine, hasNextDouble, hasNextInt, hasNextLine, next, nextDobule, nextIntnextLine`

Internal Storage

An activity has methods that you can call to read/write files

Method	Description
Resources.openRawResource(R.raw.id)	Read an input file from res/raw
openFileOutput(openFileOutput("name", mode)	Opens a file for writing (as an OutputStream)
openFileInput(openFileInput("name", mode)	Opens a file previously created by openFileOutput for reading (as an OutputStream)
filesDir	Returns a File for an internal directory for your app
cacherDir	Returns a File for a "temp" directory for scrap files