Android Studio

Kotlin (2011) is what Java (1995) would look like if it was developed today.

Basics of Kotlin

```
fun main() {  // entry point to program
    val firstName: String = "Rahul"
    // val is the keyword used to declare variables that are READ ONLY
    // All variables must have a type; we say that Kotlin is 'statically typed'

    val firstName = "Rahul"
    // although we don't need to specify it was we've already given it a value
    // Kotlin can do type inference

    var age = 36  // these are READ and WRITE

    println(age)
    println("My age is $age")  // string interpolation
}
```

```
// COLLECTIONS OR LISTS
val names = listOf("Ali", "Maya", "Chen")
println(names[2])

// collections are immutable by default
// they also must all have same type; can be made explicit with <String>
val names = mutableListOf<String>("Ali", "Maya", "Chen")  // to add or change
names.add("Dymtro")
```

```
// for loop to traverse
for (name in names) {
    println(name)
}

for (i in 1..5) {  // 5 included
    println(i)
}

for (i in 1 until 5) {  // 5 excluded
    println(i)
}
```

Functions

```
fun myFunction(myName: String) {  // state name of param and type
    println("Hello, $myName");
}

fun main() {
    myFunction(myName: "Shania")
}
```

Null

```
fun main() {
    val instagramBio: String? // if set, would be string else it's null
        val instagramBio: String? = "Grow and Inspire"
    val instagramBio: String? = null  // NULL
    // we say, instagramBio is a nullable value

    if (instagramBio != null) // always check before operating on a nullable valu
e
```

```
        print(instagramBio.toUpperCase())
}

// SHORTHAND
println(instagramBio?.toUpperCase())
```

ANDROID STUDIO

TO RUN APP ON YOUR PHONE
SETTINGS → ABOUT → PHONE BUILD NUMBER 7 TIMES
THEN OPENS UP DVELOPER OPTIONS
THERE ENABLE USB DEBUGGING THEN CONNECT PHONE VIA USB CABLE
In AS, click device manager, physical devices, click on your device then run app

Choose Empty Views activity NOT Empty Activity it's a Flamingo Thing

An **Activity** in android terms represents one screen. MainActivity is the business logic for that screen. activity_main.xml lets us construct UI and generates the XML file for us.

onCreate is automatically invoked when app is created → lifecycle event
Everything written inside is automatically run.

UI

Constraint layout is a way to create large and complex layouts with a flat *hierarachy* → helps reduce lag

Go into design and drag and drop elements from that screen. On the right, rename the id field to be something more meaningful, and can change text of a label etc. there to.

CONSTRAIN THEM HORIZONTALLY AND VERTICALLY

Next, we need to add constraints to position them properly at runtime. Drag the top anchor point to the top of screen and specify, for example, 32 from top and 48 from left.
For other elements drag the top anchor point to the element above it and thus we specify 32 from *that* etc.

To have it horizontally in centre of screen, drag left and right anchor points to the screen

At the top there is a tool for Align Edges of all elements

Can select 2 elements, and choose align vertical centres to align vertically

** Double Shift → type Reformat → makes code format consistent

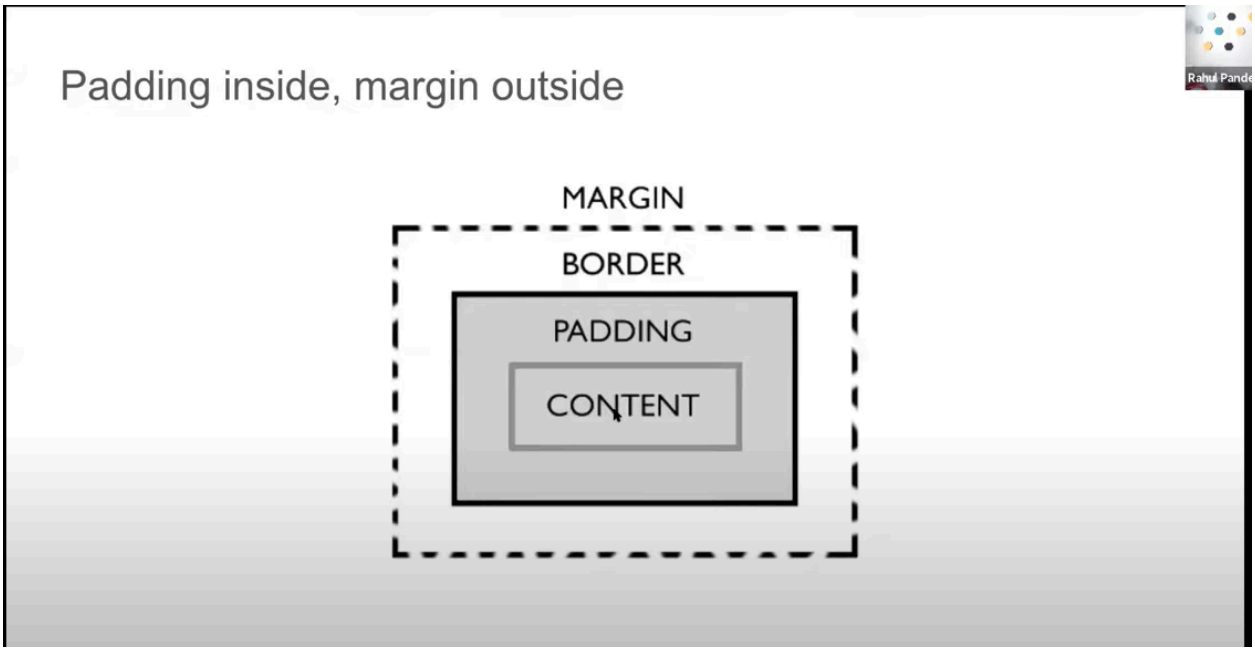
Can draw a rectangle over them to highlight all

Can type in the search bar near attributes to sometimes find what you need.

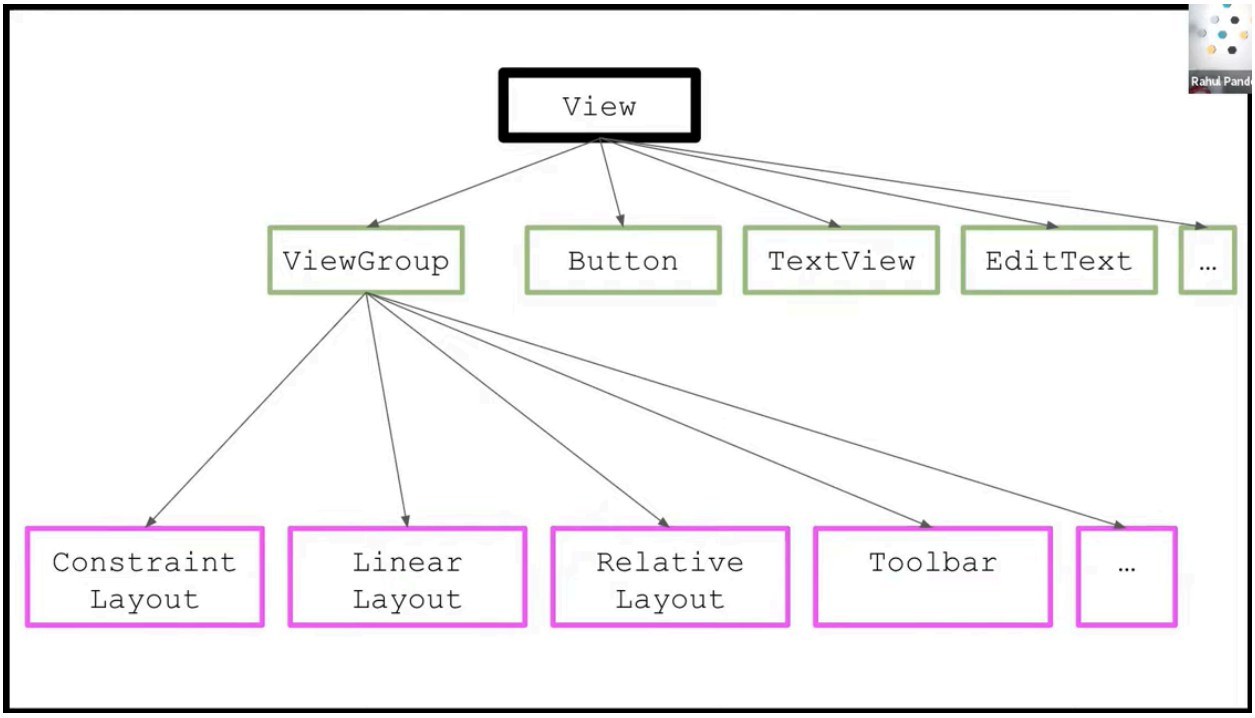
LAYOUTS
Linear layout aligns all the children either vertically or horizontally. Can give out ‘weights’ to each element → which hone takes up more space

TEXT VIEWS can make text bigger, bold etc.
Can make all caps, change font
Can also copy in emojis from google
SEEKBAR can change width in layout_width, max value that seekbar can hold
EDIT TEXT ems → how many characters wide is it, can increase textSize, can add hint → to show user what goes there.

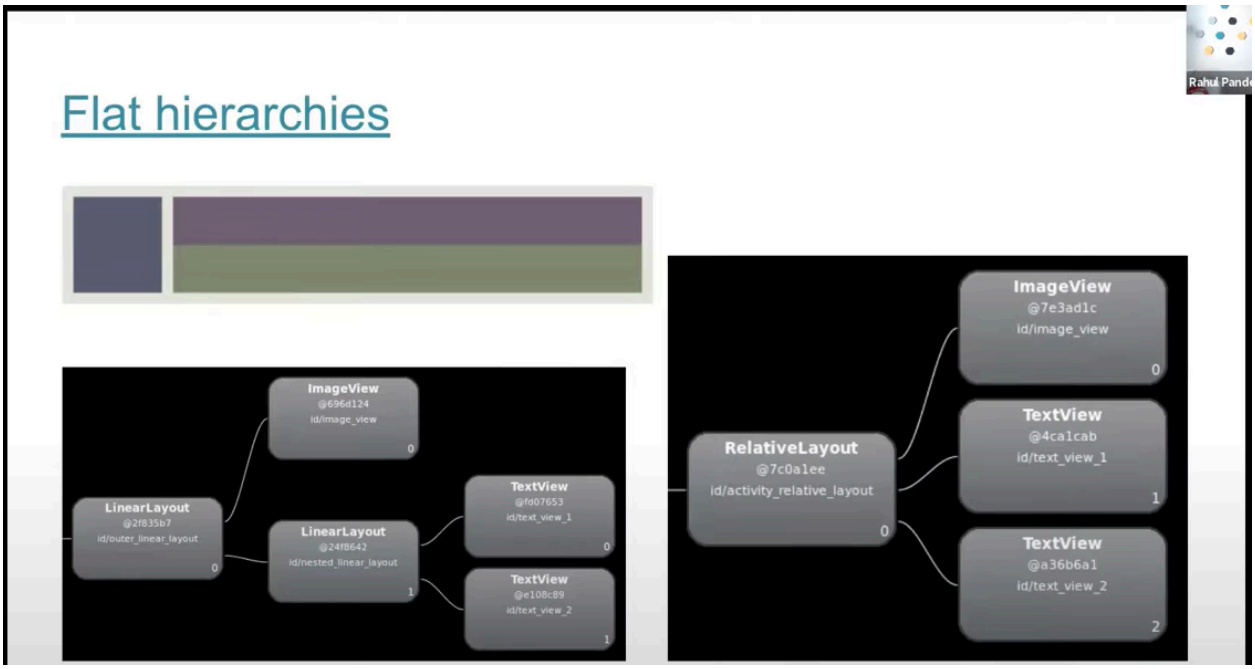
BUTTON VIEW Padding can make size of button bigger. Margin is OUTSIDE that element.



‘Tools’ text (the text with a wrench near it) only appears in the design preview, use it for the purpose of visualising
VIEW is a fundamental base class is android, anything rendered on android has to inherit from that class. One type of view is a Viewgroup → collection of views
Because of this hierarchy, XML is a good way to represent it
(XML like JSON is a way to encode data) <element> any children go here </element>
XML is a good language to define how things look and also how elements relate to each other.



Any method that can be called on a view, can be called on button also cos a button is a view and so on...
Example button has .onClickListener method which if you Ctrl+click to go tosource is defined on the VIew class!
Since backgroundView is a view, can set an on click listener on that too



Relative is depreciated because it can all be done in Constraint layout now. But reducing the depth / flattening allows Android to render faster.

BUSINESS/ APPLICATION LOGIC

Can have a submit button
OR dynamically compute as user field changes

We need reference to the views (or widgets or elements) on the screen that we need to read from or modify.

We are defining an anonymous class which implements this interface.
When we pass the object that defines what should change ,and try to define that class
We can hover over object and click Implement Members
AS helps us. Here, there are 3 methods that we have to override when using this.

'object' keyword: examples of how to create anonymous classes which are one-time use classes that are commonly used to implement interfaces

** Log Uploaded in LogCat (see bottom bar)
Choose our package name (our app) and info level logs (the i in Log.i)

AS can also help us when hovering over non defined functions

TOP OF LOG to see error during runtime: NumberFormatException. Very first blue bit after that shows where the error is. Here, cannot convert an empty string to double

Argbevaluator
a → alpha
RGB is colour
And it computes a map between integer and colour

Toast

ANDROID MANIFEST

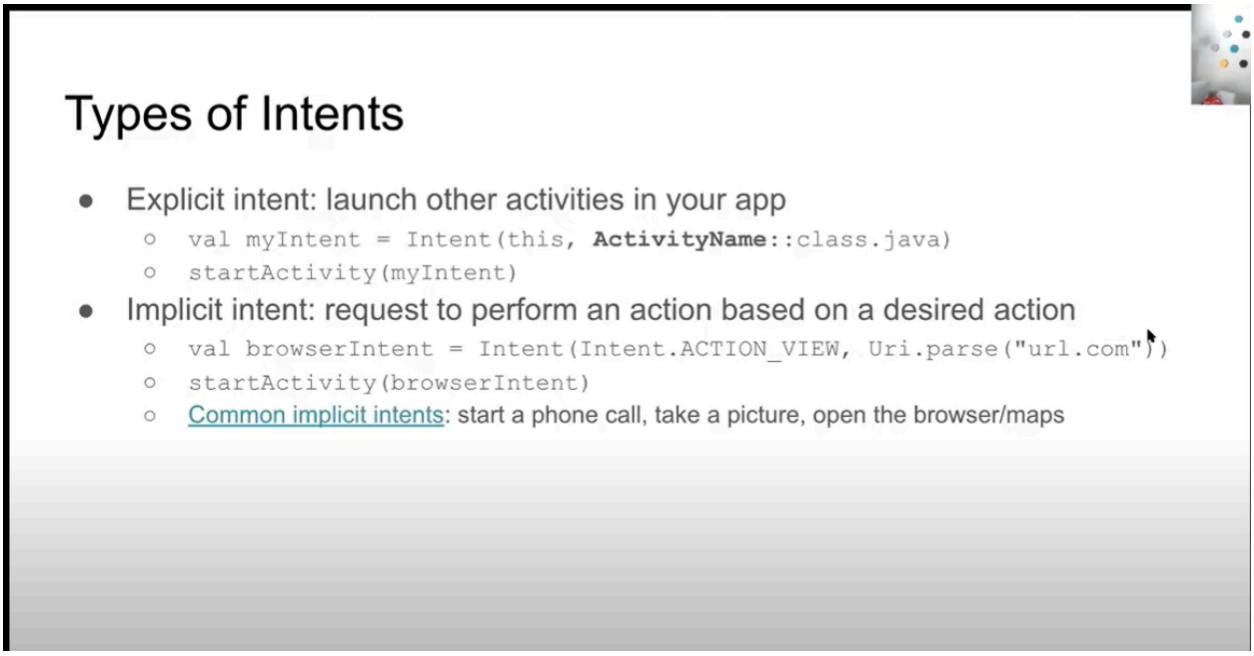
Describes different components of app
Contains permissions
But also style; Can hover over android theme: value Ctrl + Click on it
Select opened file (the little circle with 4 lines in it) to show where it lives
Colours are then defined in the colours.xml. Add them there and reference in themes

Can go to color.adobe.com

INTENTES

Can be seen as request to the Android system to open up an your own app(one activity to another), EXPLICIT
external application/ built-in service etc. IMPLICIT
If we want data back from that child activity that was launched, we call startACtivityForResult

Create a new intent object, pass in 'context' and destination of where you want to go
this keyword refers to the class which that method or line is inside
Can specify this.MainActivity if we want to refer to that and are not in the class


Types of Intents
Explicit intent: launch other activities in your app
val myIntent = Intent(this, ActivityName::class.java)
startActivity(myIntent)
Implicit intent: request to perform an action based on a desired action
val browserIntent = Intent(Intent.ACTION_VIEW, Uri.parse("url.com"))
startActivity(browserIntent)
Common implicit intents: start a phone call, take a picture, open the browser/maps