

# Dynamic Strings

29 January 2023 01:45

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_gravity="center_vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:layout_gravity="center_horizontal"
        android:textSize="30sp"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="@string/roll"/>

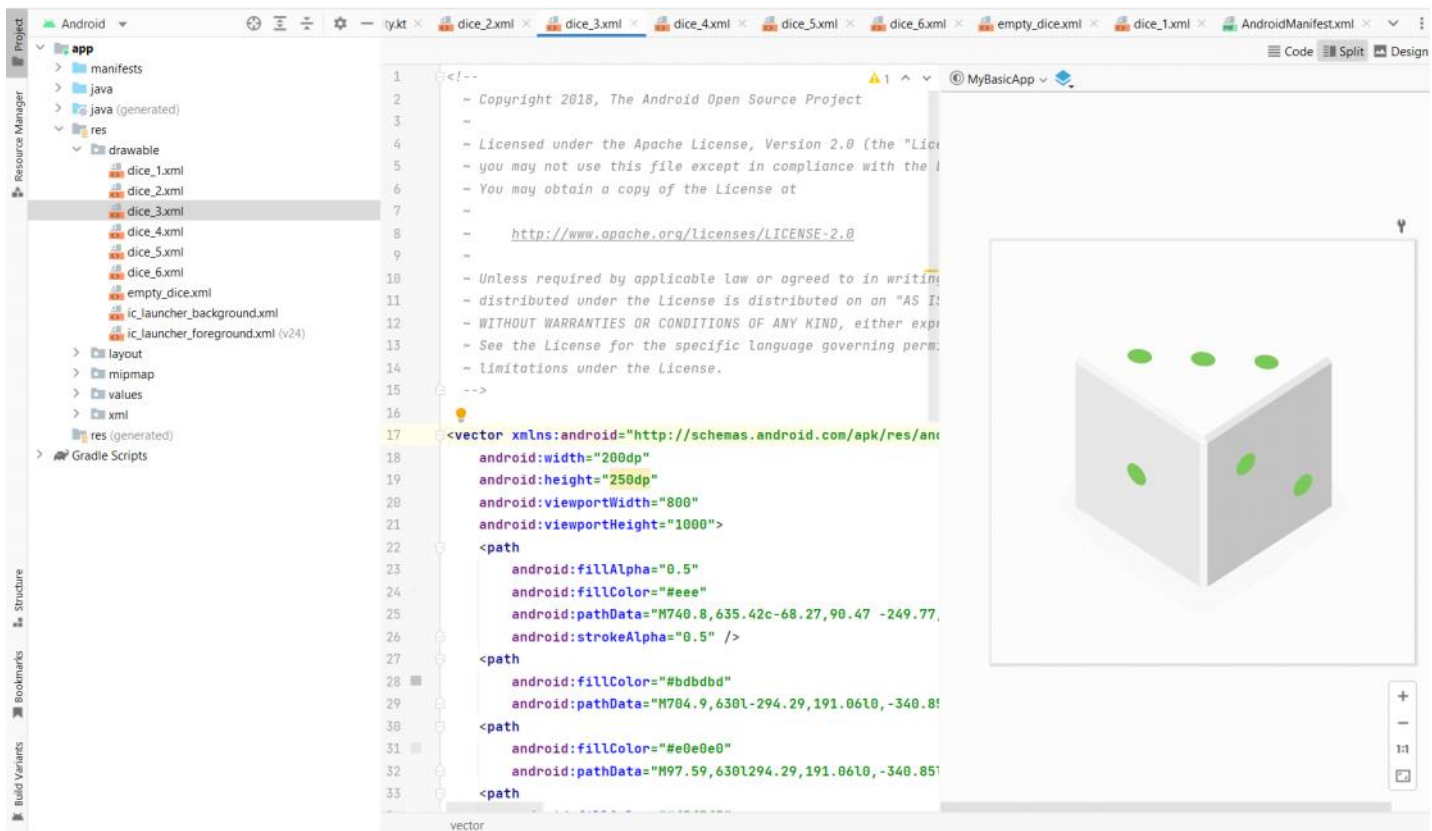
</LinearLayout>

<resources>
    <string name="app_name">DiceRoller</string>
    <string name="roll">Roll</string>
</resources>
```

- Here we can see that , we are setting text for the button dynamically , because , the value of the string and it's occurrences should be dynamic , suppose , if there is any change in the name of the view , then we hardcode it , we need to change it everywhere , so we are dynamically declaring it in strings.xml and there will be situations , where the values of the view must be changed with the change of locality , that may be change in state or change in country. So , in the following way , we refer the string of strings.xml into activity\_main.xml.
- Here in xml , the views will be coming under layout as a tree , there will be many views under a layout. For example , here TextView and Button are the child's of Linear Layout. The activity\_main.xml gives the hierarchy details to the MainActivity.
- And there will be a situation , where under one layout , there is a possibility of having multiple views , which are referred through the ID's , Main Activity refers the views through ID's and manipulate or utilize them. ID's are created when the view is created. And the information of these ID's are stored under the R class , which is nothing but the resources class. And this id is used to refer the findViewById(R.id.id\_name) , id\_name is some ascii name , but R class internally have it's address of the view object in the memory. Here we refer view object in memory , through it's id\_name , which internally refers through R class and get it's ID and give it to findViewById() and provide the instance of view object , which need to be use.

# Dice roller app and add images

04 February 2023 14:36



Here , you can see a folder called res , which consists all the resources that we can add in our app , so , if we want to add any resource in our application , we must add some xml or any type of source file such as jpg or png here and use it into the application. So currently , we are adding all the resources that our dice app needs into the res/drawable/ folder , so we can use it. And for now we are adding the xml files here in the drawable folder to use it as images. Xml file is the one , where we will be having a vector with all the details in it , and this xml will be coded in such a way to form the dice in UI format.

Images location: <https://github.com/Zain-de94/images>

## Dice Roller App

As you seen earlier , we can see we have uploaded all images that we need for the app in drawable folder of the project , to use the resources into our application , and those images are in xml format  
**MainActivity.kt**

```

1  package com.example.mybasicapp
2
3  import ...
4
5
6
7
8
9
10
11 class MainActivity : AppCompatActivity() {
12     override fun onCreate(savedInstanceState: Bundle?) {
13         super.onCreate(savedInstanceState)
14         setContentView(R.layout.activity_main)
15
16         val rollButton : Button = findViewById(R.id.roll_button)
17         rollButton.setOnClickListener { it: View!
18             rollDice()
19         }
20     }
21
22     private fun rollDice() {
23         // val randomInt = Random().nextInt(6)+1
24         val drawableResource = when(Random().nextInt( bound: 6)+1){
25             1 -> R.drawable.dice_1
26             2 -> R.drawable.dice_2
27             3 -> R.drawable.dice_3
28             4 -> R.drawable.dice_4
29             5 -> R.drawable.dice_5
30             else -> R.drawable.dice_6
31         }
32         val diceImage : ImageView = findViewById(R.id.dice_image)
33         diceImage.setImageResource(drawableResource)
34     }
35 }
36

```

Here , we can see the MainActivity class and in onCreate function we are initializing the activity\_main.xml as the main layout for the application. And in the same layout we are creating the button and the image declared in Linear layout. Here in this kt file we are giving the functionality for the button , using setOnClickListener function and if we click the button , we will call the roll Dice function which is declared in the same class , and this class is declared as the private function. And in there , we are having the when function which is used to select a value based on an input and here the value from when function will be returned into the drawableResource variable and which will be used in setting the Image in the ImageView using setImageResource function.

```

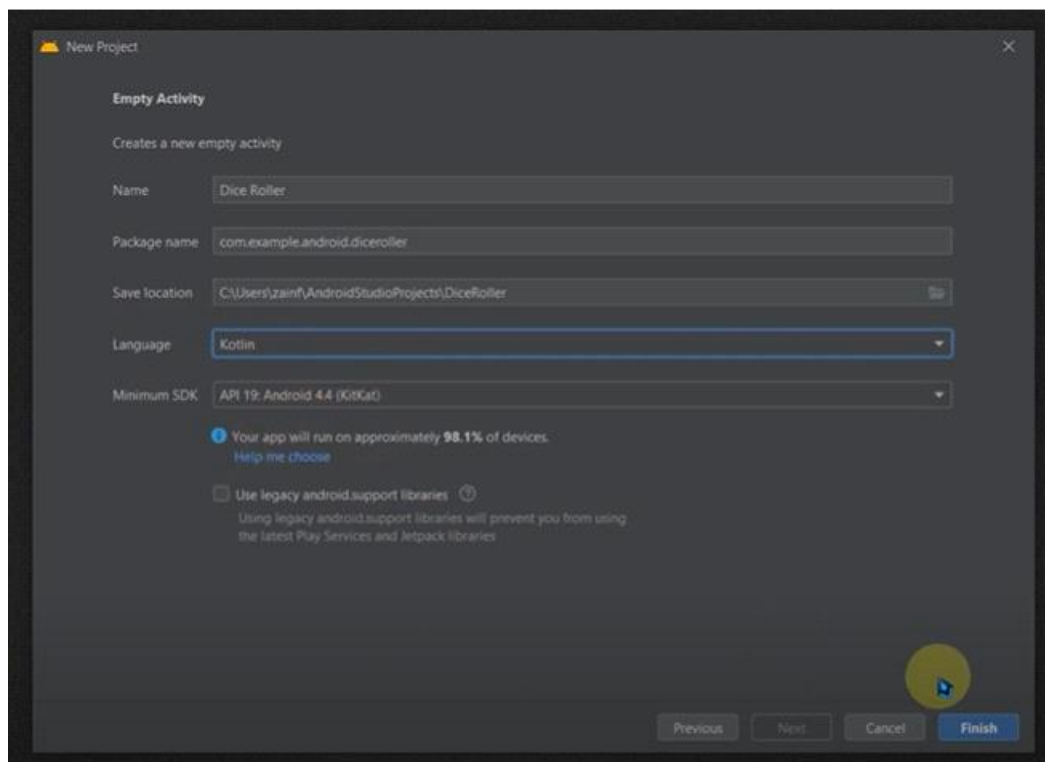
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="wrap_content"
6      android:orientation="vertical"
7      android:layout_gravity="center_vertical"
8      tools:context=".MainActivity">
9
10     <ImageView
11         android:id="@+id/dice_image"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:layout_gravity="center_horizontal"
15         android:src="@drawable/dice_1"
16     />
17
18     <Button
19         android:id="@+id/roll_button"
20         android:layout_width="wrap_content"
21         android:layout_height="wrap_content"
22         android:layout_gravity="center_horizontal"
23         android:text="Roll"/>
24
25 </LinearLayout>

```

And here , you can see a LinearLayout , in which we declared the ImageView and Button. With following attributes.

# New Project page

30 December 2022 23:17

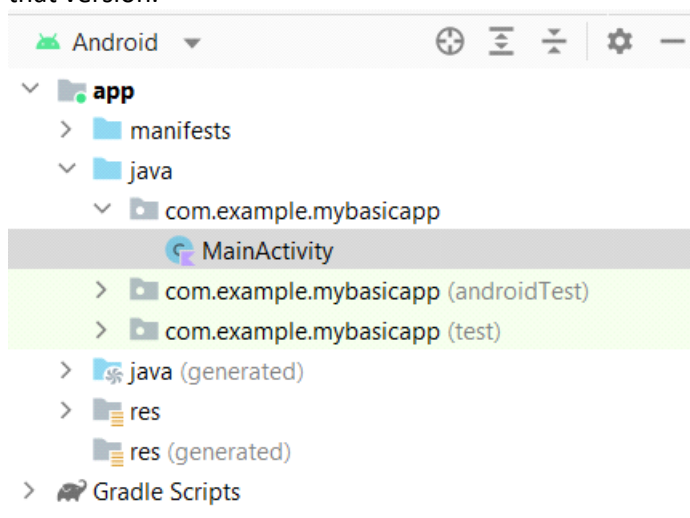


Here as we can see , when we are creating a project , we are directed to this page , and before this page , we have a page , where we can select an activity page , where we can choose the type of activity that we want. For instance , we have chosen Empty Activity.

Here we can see , we can name the project name , and we are having package name , where we have the name , com.example.android.<name-of-the-project> .

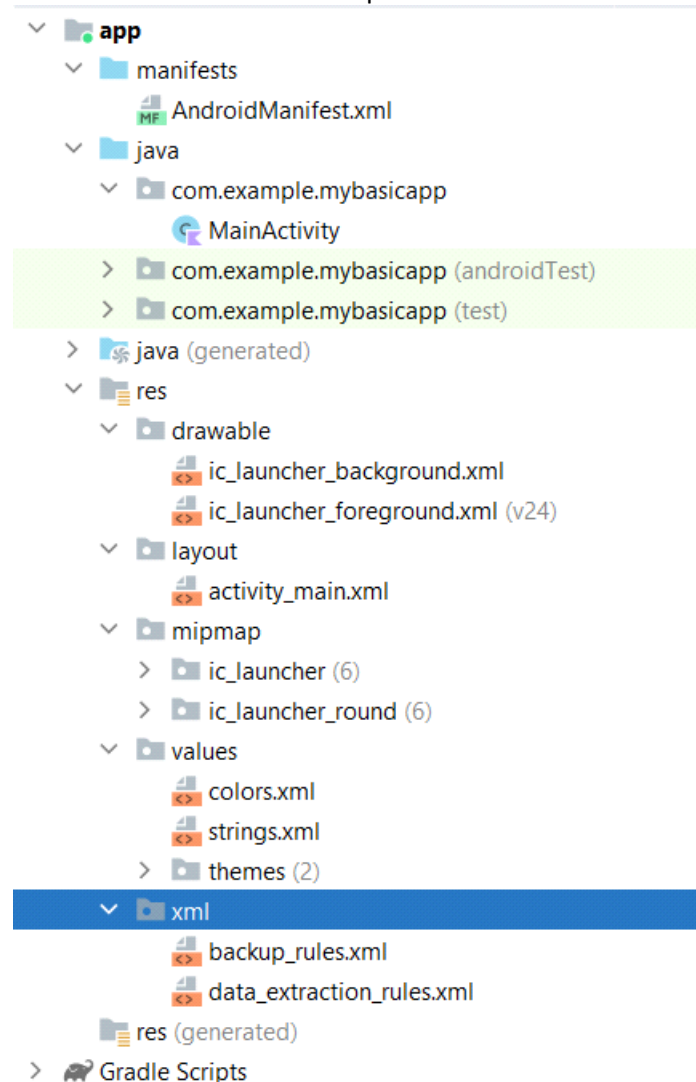
Basically this will be the domain name , where it could be <name-of-the-project>.android.example.com , so the package name will be the reverse of the domain name , and the package name denotes the structure of the folders where our project has been defined.

And then we will be having the save location and the language of the application , it can be either Java or Kotlin , we can define our minimum SDK where , we can define the latest version of android that at-least need to run the application on the mobile. Even we can see number of devices are having currently that version.



Here , we can see that , we are having a MainActivity.kt file under java /

package name / , this is the location where the main code of the app exists. Here in this main script , we will decide all the components that are necessary to run the application or any other activities that are needed to run the application or the functionality of the components are declared here , even if we are declaring any other activity , we must declare that here first. In short , any android application execution starts from this script.



Here , we can see a file AndroidManifest.xml where it contains the essential app details , so the OS can launch our app , it contains the workflow of our app , and details such as app icon , name and font style and everything that is needed for our app , it doesn't consists the resources , it just maps the resources to the necessary output that need to be displayed to the user.

And we can see a file , called MainActivity.kt , where we can find the kotlin file for the core logic of the app.

And we can see a folder called 'res' , where we can see four folders ( drawable , layout , mipmap , values , xml ). Drawable contains the background and foreground details , layout contains the structure of the layout that app is having , and mipmap contains the app icon and app icon background details. Values folder contains the strings that are needed in the application components. Here we map the string values with the keys to use it in the Application. And internally to that folder we are having a folder called themes.

Finally resources folder for static content such as images and strings.

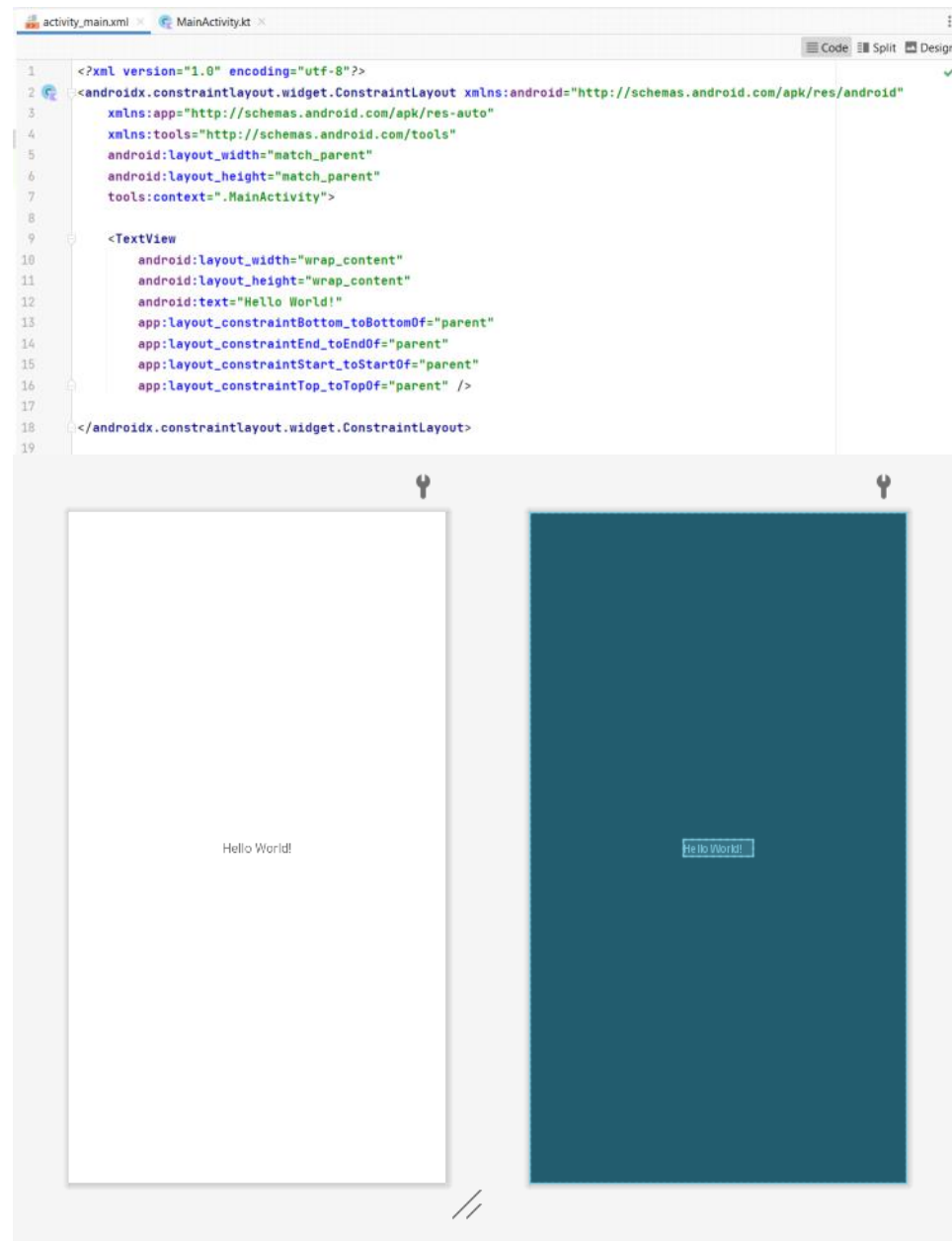
And finally , the Gradle Scripts for building and running our app , and it is used to pack the application and resources of the app and package it as the apk .

All the functionality declarations are been shown and declared under MainActivity.kt , here we will have a MainActivity class , which states all the functionality or declaration of the dependent pages for this MainActivity class.

And in res folder , we will have a drawable folder , where we store the

media details for example , in drawable we store the app icon foreground and the background. And next we are having a file called activity\_main.xml , where this shows the layout of the app and this defines the look of the app , here all the UI elements will be declared.

### activity\_main.xml ( code view & design view )



In mipmap , we store so many types of Icons for the app , and in values directory , we do key – value mapping , where we map the colors with their value codes and strings with their values , so all these we can use it under AndroidManifest.xml , this is the main app file which triggers the main activity and call other pages of the application and even use these resources folder , to do proper placement and mapping them into the application.

So all these files and resources are bound by the AndroidManifest.xml and the app will work well on the computer in the emulator as a program , but before it is building into the app , Gradle bind all the resources , code and logic into the apk and create an application to launch and use. The file which triggers the Gradle is 'build.gradle'.



## In Summary

your Android project contains

- Kotlin files for the core logic of the app
- A resources folder for static content such as images and strings
- An Android Manifest file that defines essential app details so the OS can launch your app
- Gradle scripts, for building and running your app

```
<activity
    android:name=".MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

    <meta-data
        android:name="android.app.lib_name"
        android:value="" />
</activity>
```

- This is the snippet of AndroidManifest.xml , where we can see we are defining the activity and that activity is declared as the main activity which launches first while we start the app.
- For every programming language , we will be having main class , where the program executes from there , so in android studio , we don't have such main class , so we need to declare one , and that part was done by the AndroidManifest.xml , here we can see we are naming the activity as MainActivity which links it's MainActivity.kt file and we are creating an intent , where that intent is the main intent , which launches while the app open.
- And we are having the 'activity\_main.xml' , which is the layout for the app , the layout can be designed by the UI objects or direct xml code , xml have the xml view tags which connect to the MainActivity class and connect to the objects of MainActivity through the references.





Here , we are going to see , how to connect the activity\_xml (layout) to the MainActivity (logic) , In the activity\_main.xml side , we are going to create a layout , for our case it is the linear layout and later we are going to inflate it , by adding the components , and these UI elements will be referred into the Kotlin code through the references and utilize the by creating objects for them. In this way , the UI layout code is mixing with the core code logic.

```
package com.example.mybasicapp

import ...

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

Here in the code , where we can see MainActivity is the one which is predefined class name , and we are inheriting the AppCompatActivity() method, because to get new functionality and even get some libraries which are needed to support the oldest versions of android.

And we can see here the onCreate() function , this method is called when an activity is first created in Android studio , it is used to initialize the activity and is where most of the activity's code should go. It is called only once throughout the lifecycle of an activity and is passed a Bundle object which can be used to restore any saved state from a previous instance of the activity. The onCreate() method is part of the android activity lifecycle , and it is where the activity sets up the initial layout of the screen , creates any necessary objects and sets up any event handlers or listeners. And internally in this onCreate function , we are inheriting any previous savedInstanceState and setting the content view points to activity\_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Layout of the app will be defined here , and so many views can be defined , these views are the UI elements for the apps.

### Examples of Views:

## View Examples

---

- TextView
- Button
- ImageView
- CheckBox
- RadioButton
- **EditText**
- ProgressBar

```
package com.example.mybasicapp

import ...

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

Here , this is the code we can see this code in MainActivity.kt , where in the MainActivity class , we can see onCreate function with setContentView. It means , we are setting a ContentView while we are creating the application and we are naming the content view to be set as activity\_main.xml here , so now the MainActivity class have set it's content view to this activity\_main.xml and from now on , it can manipulate it's content views , by getting the view through the reference id. The layout resource is typically defined in an XML file which describes the structure and layout of the user interface for the activity.

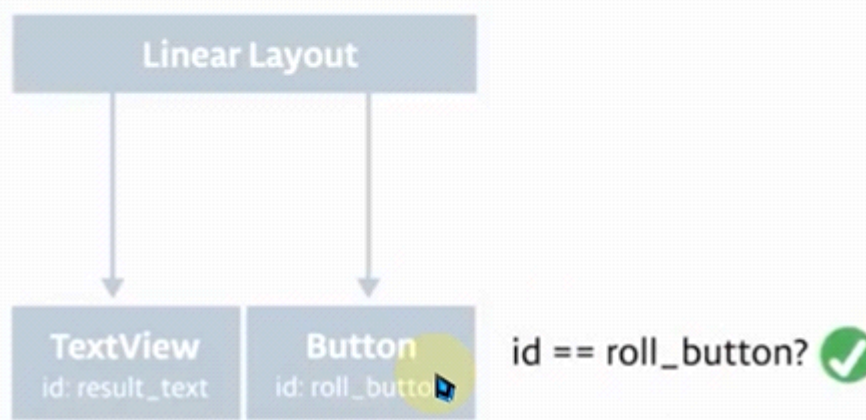
When the **setContentView()** method is called, it inflates the layout resource and attaches it to the activity's view hierarchy, so that it is

displayed on the screen.

If we want to use a particular xml layout file as a layout for our main activity , we would call it as setContentView(R.layout.activity\_main) in the onCreate() method of our main activity class.

And we also can see , here we are referring activity\_main.xml through R.layout , it means , R mean resources (res) and layout is sub folder under res folder and under that the last activity\_main is a file , which denotes the file that setContentView must refers to. Here MainActivity.kt creates an UI layer on top of it using setContentView method , here it refers to activity\_main.xml .

## View Objects in Memory



Here , we need to refer to the view object through the id , so , when we pass an id through findViewById , then internally the resources 'R' checks all the resources in hierarchial way from top to bottom , and check the equivalent id , if it matches it utilize that particular ID and if it didn't match , we go to next view in the hierarchy.

# Modify the text of the button

29 January 2023 20:40

```
package com.example.mybasicapp

import ...

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

        val rollButton : Button = findViewById(R.id.roll_button)
        rollButton.text = "Lets Roll"
    }
}
```

Here we are referring the button through it's id and creating the object locally and manipulating the property of the button. Here we are changing the text of the button , dynamically through the kotlin code.

# Basic button functionality

29 January 2023 20:48

```
package com.example.mybasicapp

import ...

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

        val rollButton : Button = findViewById(R.id.roll_button)
        rollButton.setOnClickListener { it: View!
            Toast.makeText( context: this, text: "Button Clicked", Toast.LENGTH_SHORT).show()
        }
    }
}
```

Here , we are referring the view through the ID and using `findViewById` in kotlin code , then we are referring the view in kotlin through the variable called `rollButton` of type `Button` , and this button have a function **`setOnClickListener()`** which creates a on click listener for the button , so whenever we clickOn the Button , the listener activates and it goes inside the function , in here , we have declared a `Toast` , which have a method called `makeText` , where we are setting the context of the toast as this `MainActivity` , as we are under the `MainActivity` , we are self-referring the context using `this`. And naming the button as 'Button Clicked' , And giving the duration for the toast ( `Toast.LENGTH_SHORT` ) , the toast will be created , when we declare it through `Toast.makeText` , but it doesn't shows up until we call `show` method , which is under `makeText`.

# Project 1

29 January 2023 01:27

```
<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="1"
        android:layout_gravity="center_horizontal"
        android:textSize="30sp"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Roll"/>

</LinearLayout>
```

- Here we can see , we have replaced the constraint layout with the linear layout , and if we want to change the properties of the layout or the views or any objects , we can define that in the tags by declaring properties.
- For example , layout\_width and layout\_height is used to change the height and width of the layout , wrap content mean , the objects use the space only how much it is required , but match parent , will use all the space available on the screen , even it is filled by the view or object or not.
- And we are having another attribute android:text which defines the text and textSize used to define the text size of the text , layout\_gravity is used to place the object on the layout , orientation attribute of the layout , is used to denote the position of overflow of views or objects one after one , if the orientation is horizontal , the views will accumulate to the right and if it is vertical , the objects doesn't overlap to the right , new objects add to the down.
- Here we can see , we have declared the layout\_width and the layout\_height as match parent it means , we are saying linear layout to occupy all the space on the screen , so if we want the layout to be consumed only upto it is required , then we must use wrap\_content , then the layout occupies only upto how much the views required.
- So , to make the layout to be center , we need three attributes
  - Layout\_width = match\_parent
  - Layout\_height = wrap\_content
  - Layout\_gravity = center\_vertical

# Efficient variable declaration

04 February 2023 16:01

```
package com.example.mybasicapp

import ...

class MainActivity : AppCompatActivity() {
    lateinit var diceImage: ImageView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val rollButton : Button = findViewById(R.id.roll_button)
        rollButton.setOnClickListener { it: View!
            rollDice()
        }

        diceImage = findViewById(R.id.dice_image)
    }

    private fun rollDice() {
        // val randomInt = Random().nextInt(6)+1
        val drawableResource = when(Random().nextInt( bound: 6)+1){
            1 -> R.drawable.dice_1
            2 -> R.drawable.dice_2
            3 -> R.drawable.dice_3
            4 -> R.drawable.dice_4
            5 -> R.drawable.dice_5
            else -> R.drawable.dice_6
        }
        diceImage.setImageResource(drawableResource)
    }
}
```

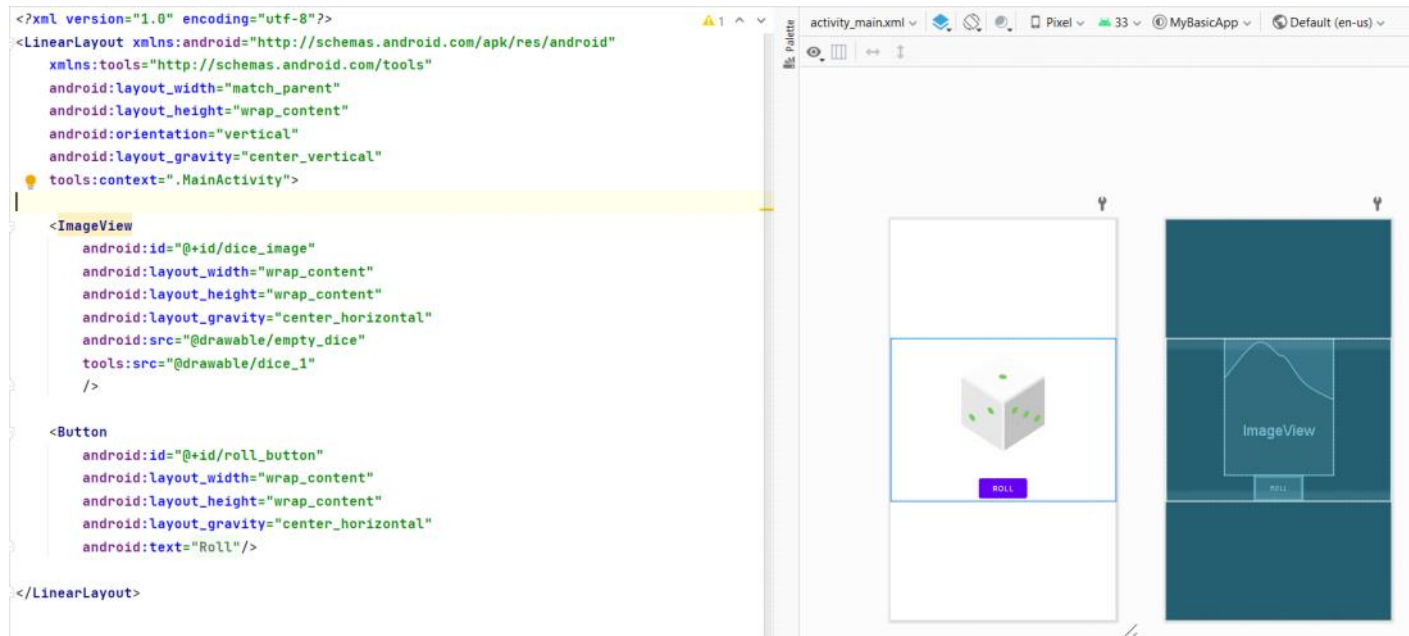
Previously, we are declaring the `diceImage` in the `rollDice` function, it means whenever the button is triggered the `rollButton` triggers the `rollDice` function and whenever the `rollDice` function triggers the `ImageView` will be re-initialized. So, it is not efficient to repeatedly initialize the `imageView` because it will lead to the memory inconsistent issues. So, to prevent this repeated initialization, we are declaring the `imageView` in the class level, before `onCreate` function, we are declaring the `ImageView` with the declaring by `lateinit`, so, it tells that we will initialize the variable late to the compiler, so the application assumes that this variable to declare this variable later in the application, instead of assuming the value `null`, and again reinitializing it with some other variable. So as all the views initialize after the `setContentView()`, we will declare the `diceImage` here with specific `ImageView` Id, and we do that in the `onCreate` function rather than the child functions of it, to prevent re-initializations.

"Don't declare the variable through `null`, you can do it through `lateinit` (late initialize), if you want to declare the variable name now and the value later."



# Namespaces

05 February 2023 13:00



Here , we can see two src's in an imageView , where one is from android namespace and other is from the tools namespace , where the tools namespace src works as the placeholder in the android studio , but actually inside the application , we are able to see the android namespace's src as the output.

# Gradle

05 February 2023 13:06

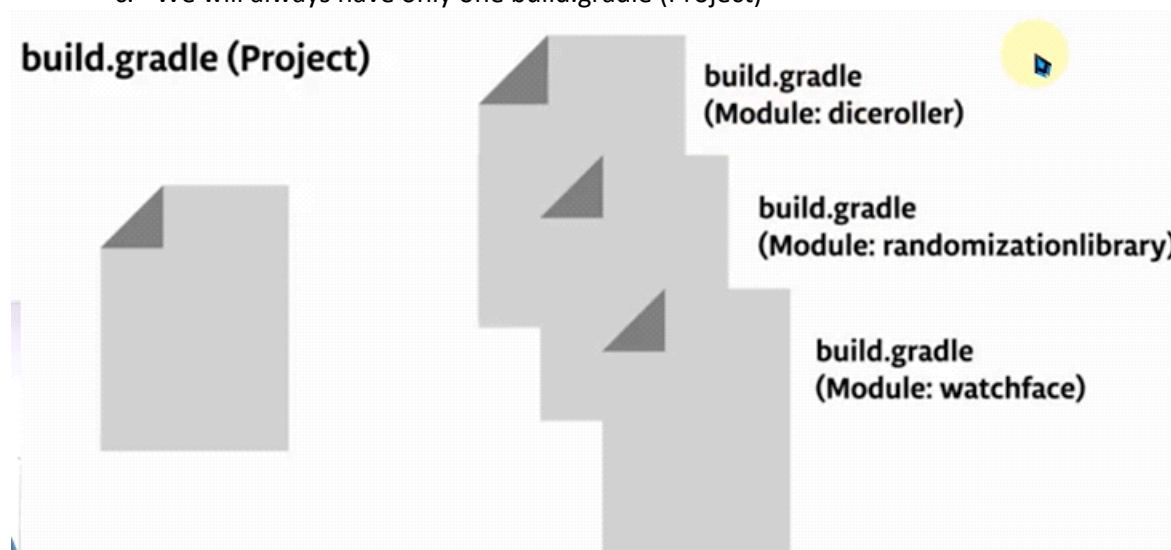
Gradle is typically an Android Build Tool , which defines what devices are supported to run our app , and compile the code to generate the executable code , and manages all the dependencies for the app and compiles everything in a single APK and this Gradle provides the app signing for Google play , so , if we have the unique sign-in details , we can login and upload our apps and no one can duplicate our apps and we can perform the automated Tests for our application.

In emulator , we will be having the run button , where whenever we click the run button the our app starts building process. During that build all our resources , compiled code and Android Manifest all other will be compiled into the single resource file. The compiled code will be converted into APK by the Gradle . APK is the android application package , this is the executable format for distributing Android applications.

APK's are simply an archive file , where it contains multiple files , plus some metadata about them. It is the file format that android uses to distribute and install apps. As a result , an APK contains all the elements that an app needs to install correctly on our device.

We are having the Gradle to convert all the resources of the application into the APK with the help of Gradle , we are having two types of Gradle files , they are

- 1) build.gradle (Project)
  - a. This contains all the project related information , and the changes made in the project . It is main centric on the project but not on the app functionality. It basically record changes of the application , so whenever we rebuild the changes , it only compiles the necessary modules to run instead of the whole project.
- 2) build.gradle (Module:app)
  - a. Module is a folder with source files and resources for a discrete piece of functionality in our app.
  - b. For now in our app , we are having only one module , so we will be having only one build.gradle (Module:app) but if we are having multiple modules then we will have many
  - c. We will always have only one build.gradle (Project)



Here in this example , we are having our already build diceroller module's build.gradle and again in our application , if we want to have the dice randomization very well , we may add some additional library from external , and for that we will have some additional module and sometimes we want to create our app's for TV's and watch faces , so even for that , we need separate modules. So , for every module , we will have one build.gradle and for the project we can have multiple build.gradle(Module) but only one build.gradle(Project).

In build.gradle we can find the repository declaration , it means the repository is a remote server where external code is downloaded from. We may download all the libraries that are needed to run the application through these repositories. For ex: google provides all the libraries that are basically needed for the application. And Jcenter() provides the plugins which are necessary , for example we need to run Kotlin , so we need libraries to run the Kotlin app and these repositories provide those for us.

Whatever files , libraries and dependencies that we install in kotlin project level , it will be providing those to all the sub projects and modules in that project level.

```
// Top-level build file where you can add configuration options common to all sub-projects/modules

plugins {
    id 'com.android.application' version '7.3.1' apply false
    id 'com.android.library' version '7.3.1' apply false
    id 'org.jetbrains.kotlin.android' version '1.7.20' apply false
}
```

For example , these three are some plugins that are installed in project level , it shows that android application library and android libraries and Kotlin android libraries are required to run the project and all the modules under it , so we installed them through the project level.

```

plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}

android {
    namespace 'com.example.mybasicapp'
    compileSdk 32

    defaultConfig {
        applicationId "com.example.mybasicapp"
        minSdk 24
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
        }
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }

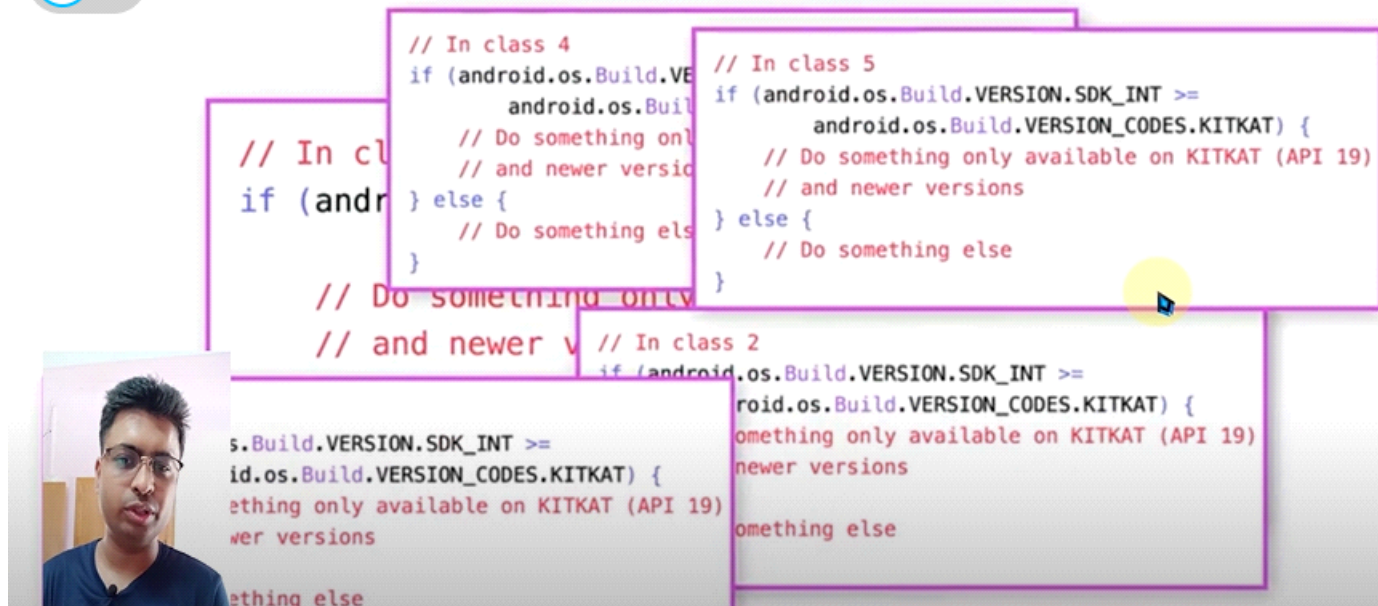
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.5.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}

```

Here , we are having the build.gradle file for the application where we declare plugins , minimum and maximum SDK required to run the application. And we will also have the application ID for the application , it looks like a website name in reverse manner , it is used as unique ID for the play store

or the app store to prevent the duplication or corruption of the application by some other person. Compiler declarations and some env variables as required and dependencies declaration for the application.



We can also declare multiple classes in following manner for different different SDK versions or SDK classes based on range of SDK version , but it is the overhead for maintaining the application.

We will be having two type of namespaces to import libraries and dependencies , one is android namespace and the other is androidx namespace , where android namespace comes with all the latest libraries and dependencies and all will be part of the phone OS already , but Google introduced the androidx to support backward compatible libraries , so most of the versions , the latest and the old could support those libraries in the application if they added these libraries as dependencies.

Android Lib (latest) + Old Android Lib = AndroidX Lib

```

package com.example.mybasicapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button
import android.widget.ImageView
import android.widget.TextView
import java.util.Random

class MainActivity : AppCompatActivity() {
    lateinit var diceImage: ImageView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val rollButton : Button = findViewById(R.id.roll_button)
        rollButton.setOnClickListener { it: View!
            rollDice()
        }

        diceImage = findViewById(R.id.dice_image)
    }
}

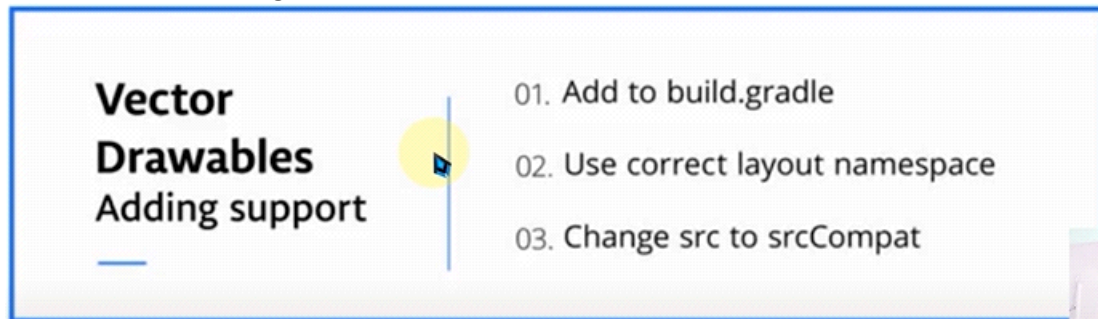
```

Here , we can see the AppCompatActivity() function , this functionality provides the backward compatibility for our apps in older versions of android. And if we see in the imports this was retrieved from androidx.appcompat.app library , which is part of androidx , as we know androidx is known for it's libraries which provide backward android version compatibility.

# XML images compatibility

05 February 2023 16:53

As we know on our app , we have developed the application based on the images of xml format , and the main benefit of these xml images are they are auto scaled , but the issue comes here is the minimum SDK version needed for using xml images is 21 , but here in this project we use minSDK as 19 , so we need to add the androidx instead of android and we need to some changes for these vector drawable through androidx



We can see the support that is given for this particular app through the build.gradle file of the application.

```
android {  
    namespace 'com.example.mybasicapp'  
    compileSdk 32  
  
    defaultConfig {  
        applicationId "com.example.mybasicapp"  
        minSdk 24  
        targetSdk 32  
        versionCode 1  
        versionName "1.0"  
  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
    }  
}
```



# 11. Vector Drawable Support

11 February 2023 11:58

In general , we are having the vector images which we earlier in our application as the dice images , the main benefit of this vector images is it's scalability and adaptability in its size according to the host device screen , but the main drawback for this is vector images are only supported for the android devices with version greater than 21 , so the devices which are less than this version will not work as intended and vector images won't work. And to handle this androidx is providing the library which is backward compatible up-to android version 7.

Steps to add vector drawable (adding support for vector drawable for the old android versions)

- 1) Add to build.gradle

```
android {  
    namespace 'com.example.mybasicapp'  
    compileSdk 32  
  
    defaultConfig {  
        applicationId "com.example.mybasicapp"  
        minSdk 19  
        targetSdk 32  
        versionCode 1  
        versionName "1.0"  
  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
  
        vectorDrawables.useSupportLibrary = true  
    }  
}
```

- a. We need to add the vectorDrawables support library into the gradle , and make the flag true.
- c. vectorDrawables.useSupportLibrary = true
- 2) Use correct layout namespace (activity\_main.xml)
  - a. xmlns:app="http://schemas.android.com/apk/res-auto"
- 3) Change src to srcCompat
  - a. app:srcCompat=""

```
<LinearLayout 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="wrap_content"  
    android:orientation="vertical"  
    android:layout_gravity="center_vertical"  
    tools:context=".MainActivity">
```

```
<ImageView  
    android:id="@+id/dice_image"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_horizontal"  
    app:srcCompat="@drawable/empty_dice"  
    tools:src="@drawable/dice_1"
```

3

```
android:layout_gravity="center_horizontal"  
app:srcCompat="@drawable/empty_dice"  
tools:src="@drawable/dice_1"  
</>
```

- This makes us to use the vector images in the old android versions , the main benefit of vector images is it helps us to reduce the size consumption of the apk , we couldn't store every image as jpg and png , if we do so , our memory size consumption of the application will be increase.

# 12. Layout Editor

11 February 2023 12:31

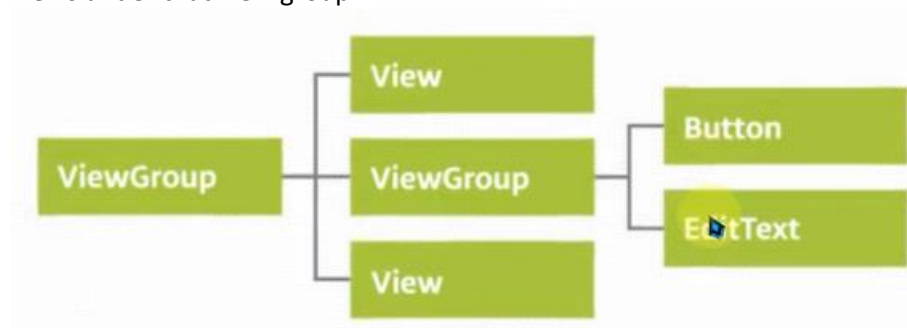
In android applications , everything is about the views , there are many type of views some of them are ImageView , TextView and ViewGroup , where ImageView cannot be further bifurcated , but viewgroup can bifurcate into many layouts they are linear layout , frame layout , constraint layout and many more , these layouts are the one , which holds all the views it can be either text or image view or any other view into one page. And TextView can further bifurcate into button and editText.

The text size dependent on the density independent Pixel (dp). This dp size system is used to give the uniform size across the systems. For example ,

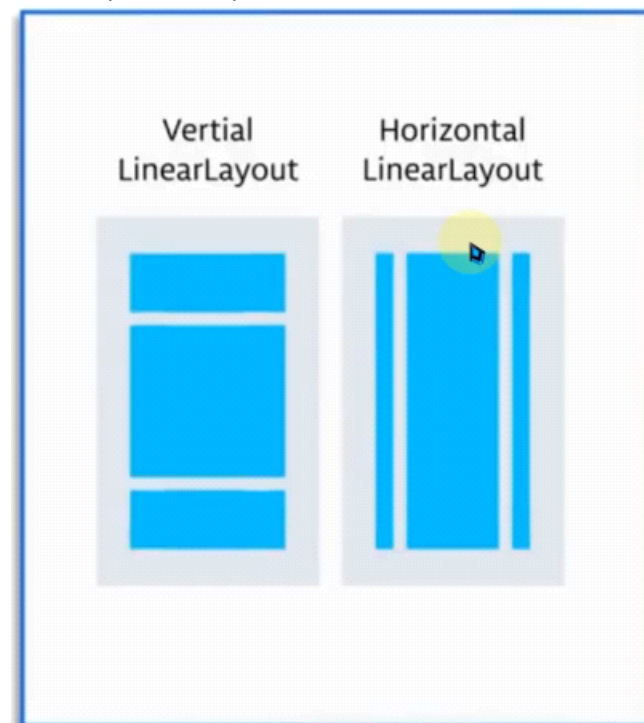
On 160 dpi screen => 1 dp == 1 pixel

On 480 dpi screen => 1 dp == 3 pixels

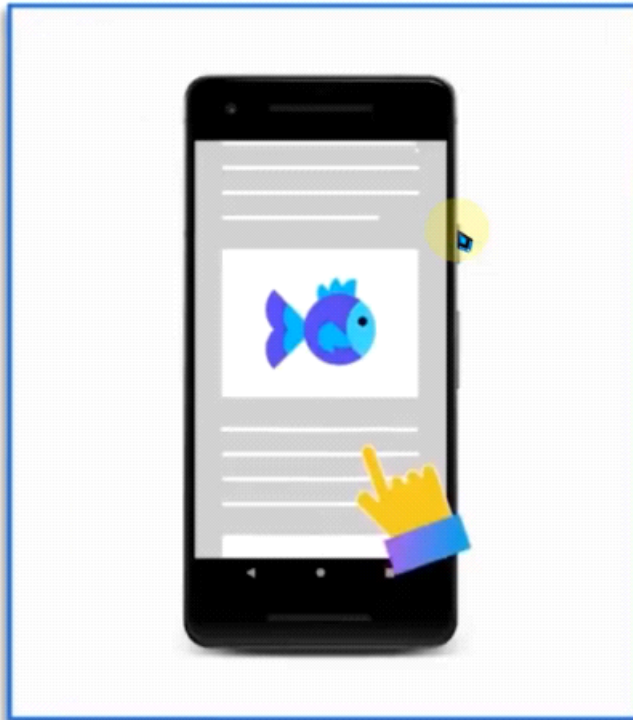
Different type of views combine and form the view group , it consist of many views on the screen and we can create another view group inside already existing view group and create views inside it. It is just having a parent with many views and having a child view group which is also having some views under that view group.



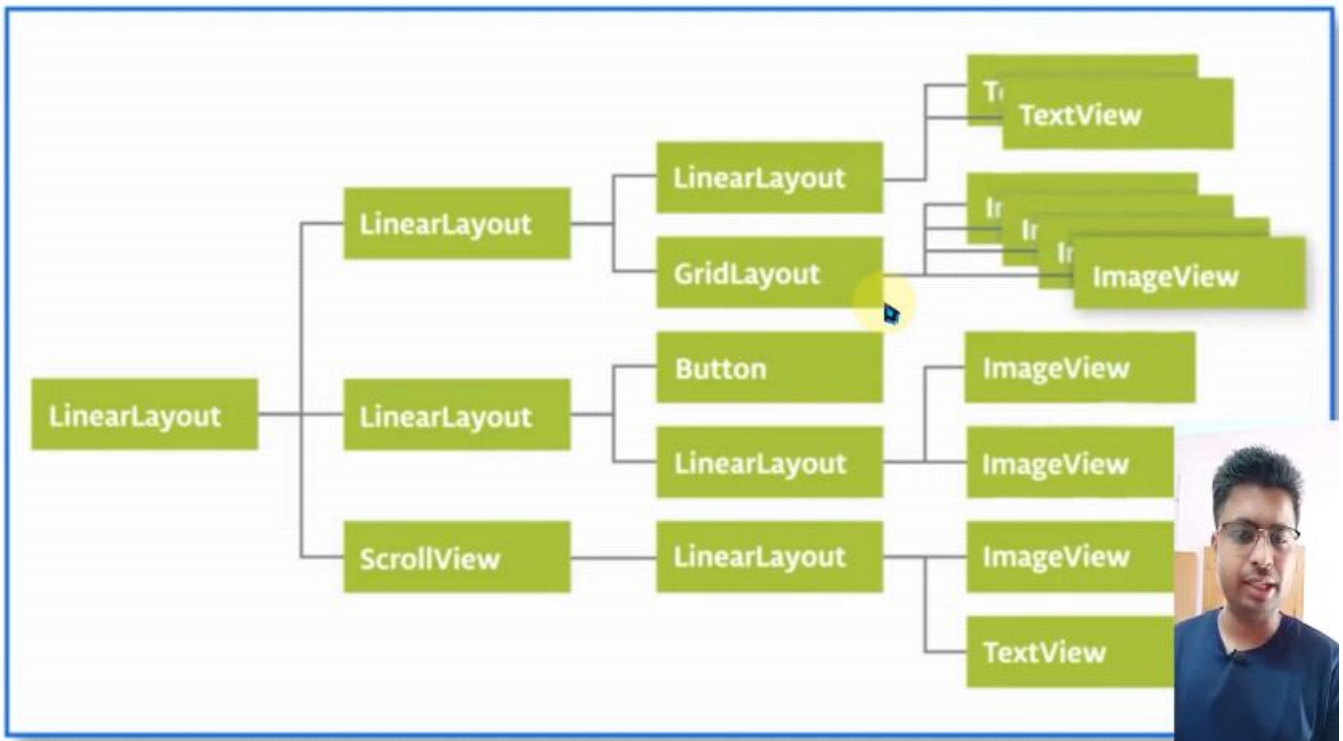
Some layout examples,



Scroll view



This is the scenario , where we are having multiple nested LinearLayout and the main issue will come when the logic combined with the UI. Because , as we know the binding in Linear Layout happens through the method similar to tree search , so how much hierarchy we made , that much time it is going to take to search the resource in our application. So , this Linear Layout will take so much memory in searching the resources to display on to the app. This increases latency of our application. Linear Layout uses the hierarchical resource binding.



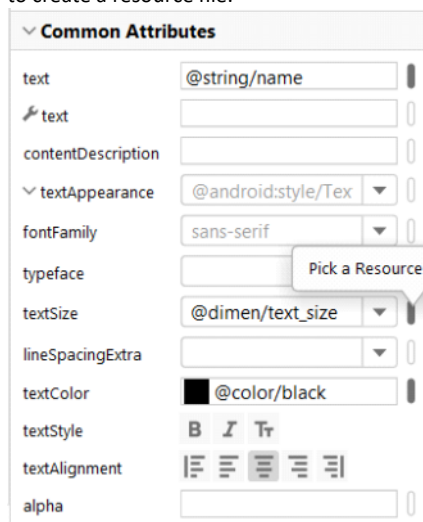
To prevent this type of difficulty , we will be using the Constraint Layout , which supports the flat data binding or also called flat layout. So , it in turn decreases the latency of the application.

## 13. Setting up styling for views

11 February 2023 15:32



This is the basic linear layout, basically as we know linear layout is not that efficient when compared to constraint layout, We can create a UI either by writing the code or we can do it through drag and drop with some manual settings performed on properties of the UI. Here we can see the android (text, textcolor, textsize, color) all four are having the values but not as direct values, but as indirect value redirection through xml file which is present on the resource, if you check (res/values/) we can find some xml files, named as "color", "dimen", "string", among these color and string are already created earlier, but now dimen is the one we created new file. We can do that through small vertical rectangle box which we can see beside the attribute that we want to create a resource file.



And we also need to know about padding and margin:



Padding is the separation space required, between the border and the content that we want to set padding to, and margin is the one which is the space between the screen to the border. In general, we give margin more than that of padding.

Extracting style out of any view.

- Go to component tree, after styling any attribute completely, and then just right click the view that we want to replicate the styles or if we want to short the xml text of the view on activity\_main.xml, we need to right click on the view of the component tree, and go to refactor option and go to 'extract style' option, and where, we can filter out what attributes that we want to combine and make a file and click Ok, then we will get the file similar to this, with all the styles that we selected in one file.

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

    <style name="style_textview">
        <item name="android:layout_marginTop">@dimen/layout_margin</item>
        <item name="android:fontFamily">@font/roboto</item>
        <item name="android:paddingTop">@dimen/small_padding</item>
        <item name="android:textColor">@color/black</item>
        <item name="android:textSize">@dimen/text_size</item>
    </style>
</resources>
```

- This particular file will be present under values folder of resources.
- And combinedly the name of this file which have all styles is styles.xml and this style file will referred in activity\_main.xml for a textView. Similar to this,

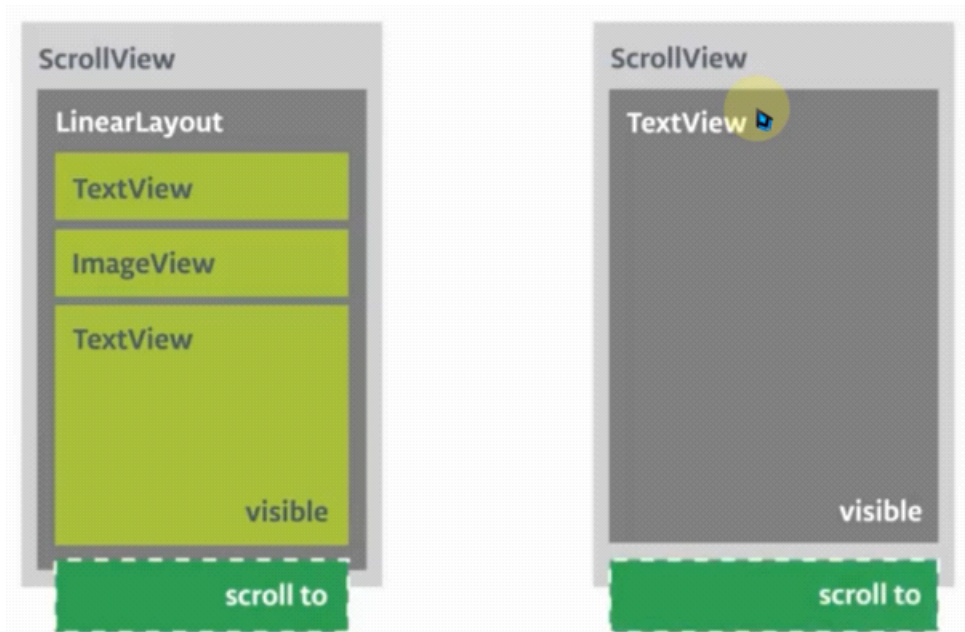
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
```

```
    <TextView
        android:id="@+id/textView2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/name"
        android:textAlignment="center"
        style="@style/style_textview" />
</LinearLayout>
```

- Here as we mentioned the style file in style attribute, we will get all the properties of the style file for this textView, this decreases the clumsy nature for the objects to represent in activity\_main.xml and we can also use the same file for many supported views, so reusability also achieved.

## 14. Scroll View

11 February 2023 17:24



We can implement scroll view in two ways , one is having a scroll view in outer level and having Linear Layout in the next level inside the scroll view , and this linear layout contains multiple types of views inside it , and here we can scroll all the views which are declared inside the linear layout , and even we can create multiple Linear layout's in once scroll view even. Here we can see the nested views.

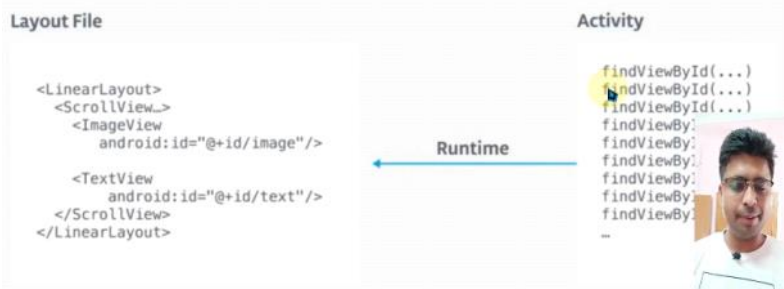
But when coming to the another scroll view creation method , we can have the single simple view , for example TextView and which we can scroll infinitely until we declared , here we are having only one layer of children , but in the first case , we can have multiple layers of children of views.



## 15. Issue without data binding (Binding Views)

11 February 2023 20:16

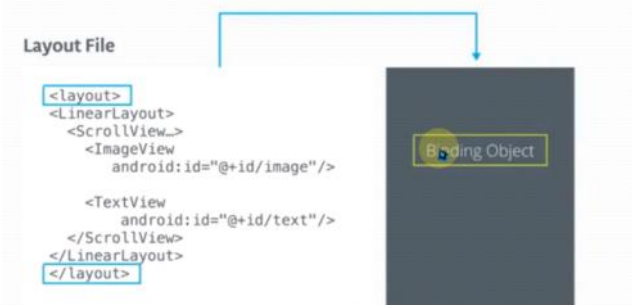
### Before Data Binding



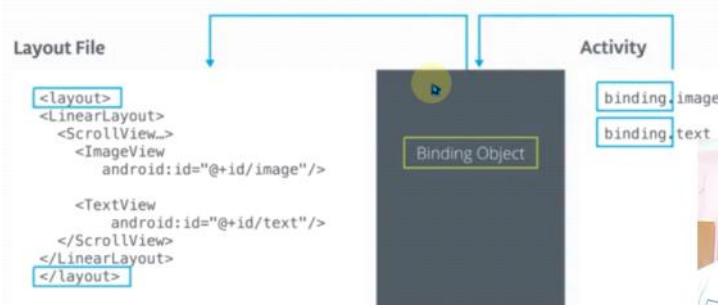
In general, without data binding, whenever we use `findViewById()` in our android application, then the application traverse the whole view tree, to get the object of the particular ID, and if this process happens at compile time, it won't be that problem, but this is happening during run time, so it increases the latency of the application. So, to solve this, there is a concept introduced data binding during compile time.

So, now we are going to create another tag to bind the layout of the application called `<layout>` and bind it to the binding object of binding class.

### Before Data Binding



And for this object, we will create an instance and use all the views directly using the object.



Now, we are just having one binding object to map the connection between the layout file and the main activity (Activity), we no need to execute find view by id, repeatedly, we can use views directly by creating the instance for the binding object, and this is similar to using the variables of the classes as those variables are already bound to class and those location of memory is certain and fixed, here that class variables are views. The mapping was done during the compile time, so it can use the variables directly using run time without latency.

To activate the binding on the application, we must do some changes in the properties.

- 1) Build.gradle app file ( We need to add data binding enabled to true )

```

android {
    namespace 'com.example.aboutme'
    compileSdk 33

    defaultConfig {
        applicationId "com.example.aboutme"
        minSdk 24
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    dataBinding{
        enabled = true
    }
}

```

- 2) Activity\_main.xml ( we need to hover the activity\_main.xml with the layout tag , so we can bind the layout that we are having earlier , it could be any view , this separate layout tag is to bind the layout view )

```

<?xml version="1.0" encoding="utf-8"?>
<layout 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">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:paddingStart="@dimen/padding"
        android:paddingEnd="@dimen/padding"
        tools:context=".MainActivity">

        <TextView
            android:id="@+id/textView"
            style="@style/style_textview"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/name"
            android:textAlignment="center"
            android:textColor="@color/black" />

        <EditText
            android:id="@+id/nickname_edit"
            style="@style/NameStyle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:ems="10"
            android:hint="@string/What_is_your_nick_name"
            android:inputType="textPersonName" />
    </LinearLayout>
</layout>

```

- b. Remember to shift the namespaces of the view layout to the outer binding layout  
3) MainActivity.kt

```

package com.example.aboutme

import ...

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // setContentView(R.layout.activity_main)
        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)
        // findViewById<Button>(R.id.button).setOnClickListener {
        //     addNickName(it)
        // }
        binding.doneButton.setOnClickListener { it: View?
            addNickName(it)
        }
    }
}

```

a.

```

private fun addNickName(view: View) {
    //Version 1
    //The following variable to the textbox
    val editText = findViewById<EditText>(R.id.nickname_edit)
    //The following variable refers to the text that we want
    //to represent same as the text in the text box
    val nickNameTextView = findViewById<TextView>(R.id.nickname_text)

    //Version 1
    //Here we are getting the text from the textbox (edit text) to textview (text)
    nickNameTextView.text = editText.text
    //We are making textbox to get gone after getting the text from EditText
    editText.visibility = View.GONE
    //We are making button to get gone after getting the text from EditText
    view.visibility = View.GONE
    //We are making textview (text) to get visible after getting the text from from the textbox (EditText)
    nickNameTextView.visibility = View.VISIBLE

    //Version 2
    //User control+b by selecting 'binding.nicknameText' then we will get to the corresponding view it is using
    //binding ignores underscores and letter cases , so our id is nickname_text but it recognized nicknameText in binding process below
    binding.nicknameText.text = binding.nicknameEdit.text
    binding.nicknameEdit.visibility = View.GONE
    binding.doneButton.visibility = View.GONE
    binding.nicknameText.visibility = View.VISIBLE

    //Version 3
    binding.apply { this: ActivityMainBinding
        nicknameText.text = nicknameEdit.text
        invalidateAll()
        nicknameEdit.visibility = View.GONE
        doneButton.visibility = View.GONE
        nicknameText.visibility = View.VISIBLE
    }

    val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
    imm.hideSoftInputFromWindow(view.windowToken, flags: 0)
}
}

```

b.

- c. Here we can see that , we are creating an instance for binding object , which is of type activity main binding and in the onCreate function , we are binding this MainActivity kotlin file with the activity\_main.xml to make easy for activity\_main.xml view objects easily accessible to the MainActivity kotlin file. And next we can see using "binding." we can access any view object under activity\_main.xml, the syntax would be
- d. binding.<view-object-id>.method
- e. And we can see in version 2 , we can call all objects individually with the binding object or else , we can call every object from inside the binding.apply method , if we use binding.apply , the binding object will automatically applied to every object declared under the apply function.

## 16. Binding data

12 February 2023 00:20



Earlier we tried to bind the views , activity\_main.xml with the mainactivity , but now we want to bind the data itself , for example earlier , when we binded the views , we tried to access the text of any view then we tried to access it through the view , but not direct way , but now when we do binding through the data , we can directly bind the data , through the data classes.

As we can see the image , we are creating a new data class and where we are declaring a object class MyName and the variable names under it are name , nickname. So , we are going to use these variables in our application through the data binding.

As seen in the image all our data class and activity\_main.xml and MainActivity.kt are binded by the ActivityMainBinding. So using this link we want to establish data binding. So , now as seen in the image , we are declaring the value of the textView , through the data binding , but not by the strings.xml or hard coding. Here we are declaring a new data object inside the activity main xml file and we are importing the variable with it's value from the particular package. See the data and variable tag in the image above. And we use that variable inside the textView through annotation. Here myName object can access two variables as per declared in separate kt file. And use them in the main activity kt code.

activity\_main.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<layout 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">  
  <data>  
    <variable  
      name="myName"  
      type="com.example.aboutme.MyName" />  
  </data>  
  <LinearLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="vertical"  
    android:paddingStart="@dimen/padding"  
    android:paddingEnd="@dimen/padding"  
    tools:context=".MainActivity">  
  
    <TextView  
      android:id="@+id/textView"  
      style="@style/style_textview"  
      android:layout_width="match_parent"  
      android:layout_height="wrap_content"  
      android:text="@={myName.name}"  
      android:textAlignment="center"  
      android:textColor="@color/black" />
```

```

<EditText
    android:id="@+id/nickname_edit"
    style="@style/NameStyle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="@string/What_is_your_nick_name"
    android:inputType="textPersonName" />

<Button
    android:id="@+id/doneButton"
    style="@style/Widget.AppCompat.Button.Colored"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="@dimen/layout_margin"
    android:fontFamily="@font/roboto"
    android:text="@string/done" />

<TextView
    android:id="@+id/nickname_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAlignment="center"
    android:visibility="gone"
    style="@style/NameStyle"
    android:text="@={myName.nickname}"/>

<ImageView
    android:id="@+id/bigstar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@color/black"
    android:contentDescription="@string/star_desc"
    app:srcCompat="@android:drawable/btn_star_big_on" />

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/bio"
        style="@style/NameStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingStart="@dimen/padding"
        android:paddingEnd="@dimen/padding"
        android:text="@string/bio" />

</ScrollView>
</LinearLayout>
</layout>

```

MainActivity.kt



```

package com.example.aboutme

import android.content.Context
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.view.inputmethod.InputMethodManager
import androidx.databinding.DataBindingUtil
import com.example.aboutme.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private val myName: MyName = MyName( name: "Jeevan", nickname: "Kanna")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        // setContentView(R.layout.activity_main)
        binding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main)
        binding.myName = myName
        // findViewById<Button>(R.id.button).setOnClickListener {
        //     addNickName(it)
        // }
        binding.doneButton.setOnClickListener { it: View!
            addNickName(it)
        }
    }

    private fun addNickName(view: View) {
        //Version 1
        // The following variable to the textbox
        val editText = findViewById<EditText>(R.id.nickname_edit)
        // The following variable refers to the text that we want
        // to represent same as the text in the text box
        val nickNameTextView = findViewById<TextView>(R.id.nickname_text)

        // Version 1
        // Here we are getting the text from the textbox (edit text) to textview (text)
        nickNameTextView.text = editText.text
        // We are making textbox to get gone after getting the text from EditText
        editText.visibility = View.GONE
        // We are making button to get gone after getting the text from EditText
        view.visibility = View.GONE
        // We are making textview (text) to get visible after getting the text from from the textbox (EditText)
        nickNameTextView.visibility = View.VISIBLE

        //Version 2
        // User control+b by selecting 'binding.nicknameText' then we will get to the corresponding view it is using
        // binding ignores underscores and letter cases , so our id is nickname_text but it recognized nicknameText in binding process below
        binding.nicknameText.text = binding.nicknameEdit.text
        binding.nicknameEdit.visibility = View.GONE
        binding.doneButton.visibility = View.GONE
        binding.nicknameText.visibility = View.VISIBLE

        //Version 3
        binding.apply { this: ActivityMainBinding
            myName?.nickname = nicknameEdit.text.toString()
            nicknameText.text = nicknameEdit.text
            invalidateAll()
            nicknameEdit.visibility = View.GONE
            doneButton.visibility = View.GONE
            nicknameText.visibility = View.VISIBLE
        }

        val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
        imm.hideSoftInputFromWindow(view.windowToken, flags: 0)
    }
}
}

```

MyName.kt

```
package com.example.aboutme
```



```
data class MyName (var name:String = "",var nickname:String = "")
```



## 17. Constraint layout

12 February 2023 18:50

Constraint layout example:

```
<TextView
    android:id="@+id/infoText"
    android:layout_width="8dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="@dimen/margin_wide"
    android:layout_marginTop="@dimen/margin_half"
    android:layout_marginEnd="@dimen/margin_wide"
    android:text="@string/tag_the_boxes_and_buttons"

    android:textAppearance="@android:style/TextAppearance.DeviceDefault.Medium"
    app:layout_constraintBottomToBottomOf="parent"
    app:layout_constraintEndToEndOf="parent"
    app:layout_constraintHorizontalBias="1.0"
    app:layout_constraintStartToEndOf="parent"
    app:layout_constraintTopToBottomOf="parent"
    app:layout_constraintVerticalBias="0.5" />
```

### Constraint

A connection or alignment to another UI element, to the parent layout, or to an invisible guideline.

In constraint layout, every UI element or view is dependent on the another UI element or view or the parent layout itself.

With help of constraint layout, we can position our UI components in any sort of order whether it may be horizontal or vertical. But in case of Linear layout, we can only arrange our UI components either in horizontal or in vertical manner. This is the main benefit we can get using constraint layout.

### Advantages of Constraint Layout

- You can make it responsive to screens and resolutions
- Usually flatter view hierarchy
- Optimized for laying out its views
- Free-form - place views anywhere, and the editor helps add constraints

The main advantage of constraint layout is to we can place the views relatively anywhere on the screen either vertical or horizontal and across all through resolutions because the constraint layout is similar to relative layout, the views will be set based on the space that is available.

In linear layout, we will be having the hierarchical view placements, it means when we want to use any view under linear layout, we need to follow and iterate through the tree structure and find it, this will be difficult when it is a complicated application, this will be solved when we use the binding concept for linear layout to make the hierarchy flat.

And when coming to the constraint layout, we will have flatter view hierarchy as default. We can place the views where ever we need, there is not perfect orientation or placement order for this layout it can be vertical order or the horizontal. And we no need to be concern about remembering the xml format to write the code for constraint layout, we will be having the editor from where we can add the constraints for the layout and the views or UI elements in it.

#### Types of constraints:

**match\_constraint**: It basically place the elements of the UI based on the parent layout, so this type of constraints are basically useful when we want to make an app which need to work on all the resolutions.

**wrap\_content**: This occupies the dp space that we asked to take, but it also wraps up the content, it means that it calculate space including the content.

**fixed\_constraints**: It is basically giving a strict constraint to occupy the space for the UI elements.



No X Y  
coordinates or  
constraints in  
Design

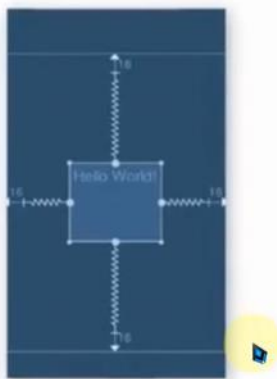


View appears  
at 0,0 when  
app is run

If we didn't provide any constraints for the view , then the view appears at (0,0) when the app is run.



Attributes	
Layout_Margin	[?, 16dp, 16dp, 16dp, ?]
all	
end	@dimen/margin_wide
start	@dimen/margin_wide
top	@dimen/margin_wide
bottom	



Adding constraints to all all sides  
by default creates a 50% bias  
and centers the view



0 horizontal bias  
0 vertical bias



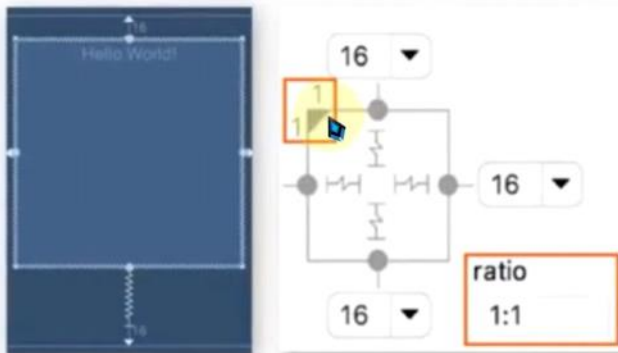
100 horizontal  
100 vertical bias



20 horizontal  
80 vertical bias



## Ratio Square




After giving the constraints , we can even give the box ratio , to get the desired shape no matter what constraints around it , so this helps us to have multiple such views in a screen , with the same shape and no interdependency in views , every shape be the same and the views will be auto adjusted accordingly as the size given in DP.

## 18. Ratio setting for a button

12 February 2023 22:22

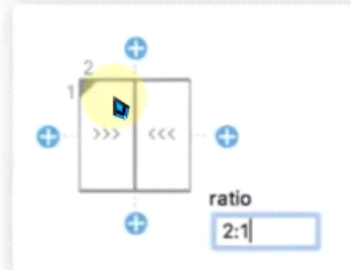
### Ratio Setting



layout\_width: wrap\_content  
layout\_height: 0

```
< Button  
    android:layout_width="wrap_content"  
    android:layout_height="0dp"  
    app:layout_constraintDimensionRatio="2:1"  
    [... other properties...] />
```

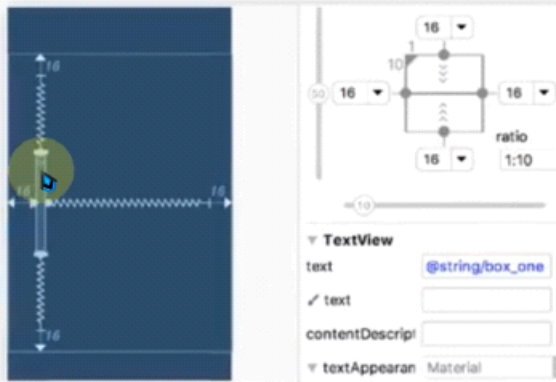
### Ratio Setting



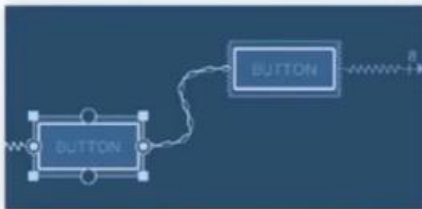
ratio: 2:1

```
< Button  
    android:layout_width="wrap_content"  
    android:layout_height="0dp"  
    app:layout_constraintDimensionRatio="2:1"  
    [... other properties...] />
```

# Ratio in Layout Editor



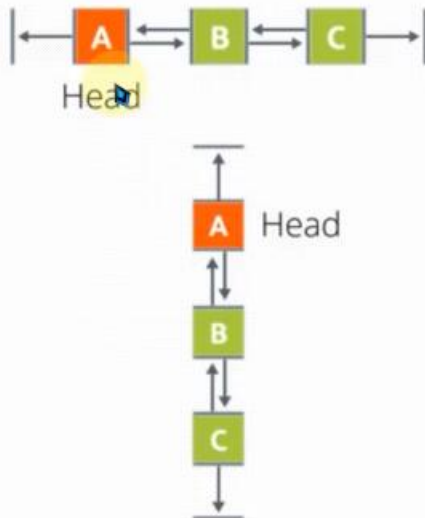
## Chains



```
<Button
    android:id="@+id/button"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/button2"
    ... />

<Button
    android:id="@+id/button2"
    app:layout_constraintEnd_toStartOf="@+id/button"
    app:layout_constraintStart_toStartOf="parent"
    ... />
```

## Chains



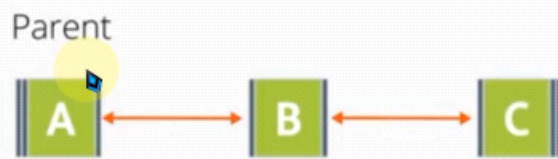
## Spread Chain

Parent



**Spread chain:** Margin starts from the parent level

## Spread Inside Chain



**Spread Inside chain:** Parent doesn't have any margin and the intermediary views will be using all the space available.

## Weighted Chain



**Weighted chain:** we will be giving weight to occupy the space for all the nodes which exists intermediary , except the last one , the last one will use the rest of the space.

## Packed Chain

Parent



**Packed chain:** The blocks will be margined from the parent ( equal margin from both sides ) and all the node views will be packed and be together as one.

## Packed Chain with Bias

Parent



**Packed chain with bias:** similar to the packed chain , but there is a bias in the margin from the two sides of the parent.