Used to resize the UI element when rotating

override fun onConfigurationChanged(newConfig: Configuration) {  
 super.onConfigurationChanged(newConfig)  
 if (newConfig.orientation === Configuration.*ORIENTATION\_LANDSCAPE*) {  
 display.setPadding(0,0,24,0)  
 display.*textSize* = 24f  
 } else if (newConfig.orientation===Configuration.*ORIENTATION\_PORTRAIT*){  
 display.setPadding(0,24,24,0)  
 display.*textSize* = 32f  
 }  
}

using the binding method that we learned in recyclerView we can use it in the main to short our fidViewById

override fun onCreate(savedInstanceState:Bundle?) {

super.onCreate(savedInstanceState)

**val binding = ActivityAwesomeBinding.inflate(layoutInflater) 🡺 binding line**

binding.title.text = "Hello"

binding.subtext.text = "Concise, safe code"

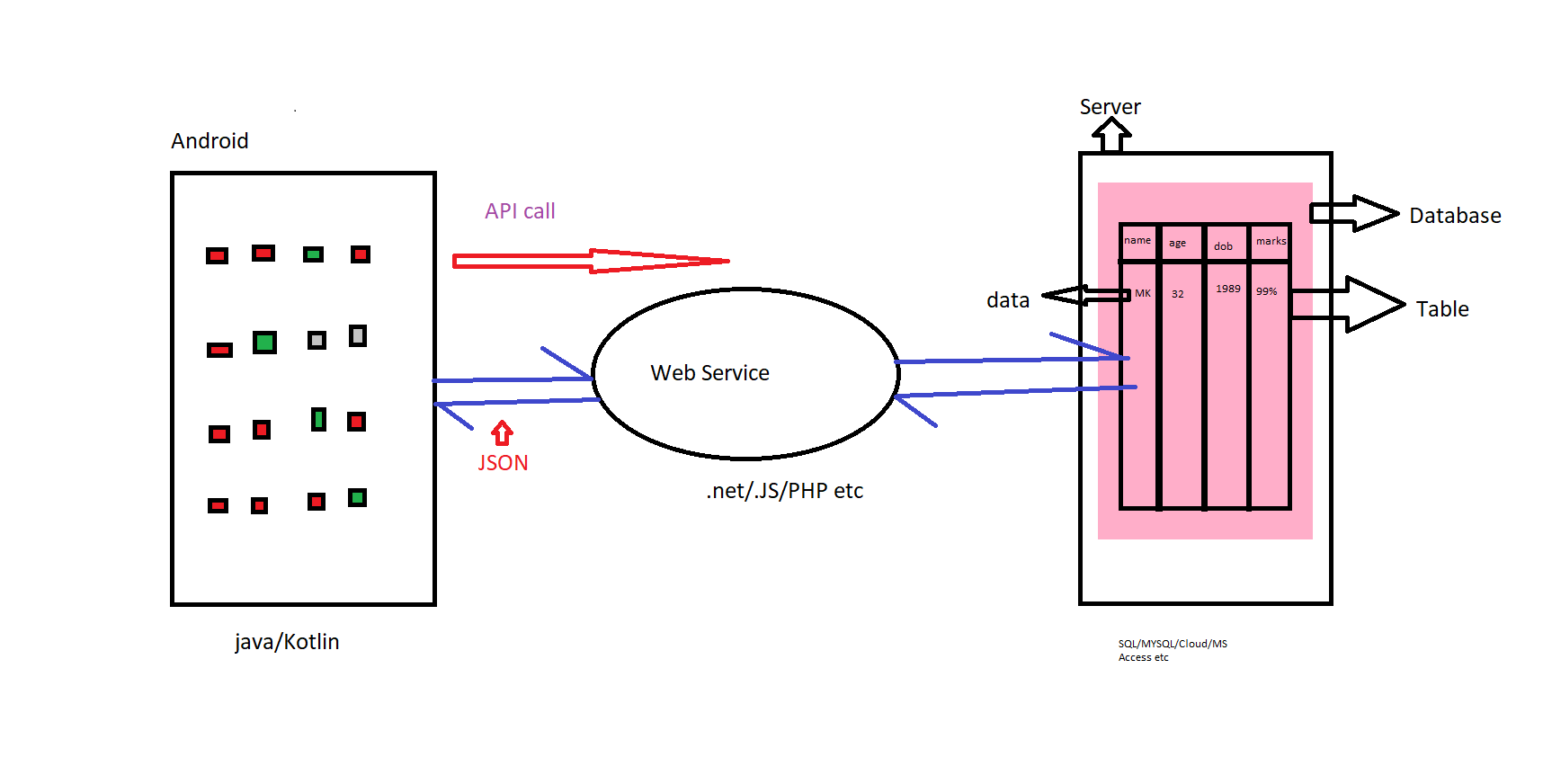
binding.button.setOnClickListener { /\* ... \*/ }

setContentView(binding.root) 🡺 needed to make it the active view on the screen

}

Int build.gradle file

android {  
    ...  
    buildFeatures {  
        viewBinding = true  
    }  
}



We refer to Objects as JsonObject

Arrays are called JsonArrays

JsonObjects are denoted with Curly braces {}

JsonArrays are denoted with Curly square brackets []

**Example JSON format data:**

{“Name”:”CodingDojo”} simple JSON example with value as single string

{ “Age”  : 26 } simple JSON example with value as single number

{“Details” : {“Name”:”CodingDojo”}  } simple JSON example with value as a single JsonObject

{“Users”: [

              {“Name”:”CodingDojo”} ,

              {“Name”:”CodingDojo”} ,

              {“Name”:”CodingDojo”}

              ]

} simple JSON example with value as JsonArray

In order to work with Retrofit, let’s divide the steps in using Retrofit HttpClient.

* Add the dependencies in the dependency section
* Understand your result and based on that create a Model class, to hold data
* Declare and initialize the Retrofit instance
* Start getting data from Retrofit!
* Use the data in your Android app.

**Adding Dependencies:**

In the build.gradle file (Module level) you will have to add the following decencies.

implementation **'com.google.code.gson:gson:2.8.8'**

implementation **'com.squareup.retrofit2:retrofit:2.9.0'**

implementation **'com.squareup.retrofit2:converter-gson:2.9.0'**

implementation **'com.squareup.okhttp3:logging-interceptor:4.2.2'**

Add internet permission in the AndroidManifest file above the <application> tag

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

Getting all the data in the database will freeze the whole application until it finishes loading the data, such a situation might lead to a crash in the application.

To avoid this issue, Coroutines were introduced in Kotlin to produce a new way of writing code that can run in parallel to others. Coroutines can also be used to run heavy code in a way that doesn’t lock the main thread.

A Coroutine runs jobs in the background thread in an efficient way that will solve the time-consuming operations problem in the main thread

Copy the Dependency and paste it in the “build.gradle”:

dependencies {

implementation("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9")

}

To run time-consuming code in the background using Coroutines we need to define the context “Dispatchers” of the background work which can be (IO, Main, Default).

After defining the context, we can start our coroutines by creating its builder (Launch):

CoroutineScope(IO).launch {

}

We also have globalScope which will keep running as long as your application runs, And lifecycleScope which is bound to the Activity or Fragment lifecycle.

The time-consuming code will be written inside the two curly brackets.

Next, we will create a function which will be used and controlled by coroutines:

private suspend fun getResult(){

}

The suspend modifier is added to the function to define that it can be stopped/resumed by the Coroutines and it can only get called inside the Coroutines scope.

If we need to access views from within Coroutines then we need to change the context of the job to be in the main thread by using withContext(Main).

withContext(Main){

}

**Lambda** expressions are anonymous functions that can be passed as parameters, stored in variables, or returned as values.

data class HighScore(val name: String, val score: Int)

fun main(){

// Lambda

val greeting = { println("Hello") }

greeting()

// Function

fun greetingFun(){

println("Hello")

}

greetingFun()

// Lambda

val personalGreeting = { name:String -> println("Hello $name") }

personalGreeting("Sam")

// Function

fun personalGreetingFun(name: String){

println("Hello $name")

}

personalGreetingFun("Sam")

// Lambda

val returnSum = { a: Int, b: Int -> a + b }

println(returnSum(4, 5))

// Function

fun returnSumFun(a: Int, b: Int): Int{

return a + b

}

println(returnSumFun(4, 5))

val highScores = listOf(

HighScore("Bob", 5000),

HighScore("Sara", 300),

HighScore("Jane", 7200),

HighScore("adfasfds", 900),

HighScore("Fred", 2300)

)

// Lambda

val highestScore = highScores.maxByOrNull { it.score } // it replaces highScores -> highScores.score

println("${highestScore!!.name} wins!")

// Function

fun highestScoreFun(): HighScore{

var hs = 0

var winner = highScores[0]

for(i in highScores){

if(i.score > hs){

hs = i.score

winner = i

}

}

return winner

}

println("${highestScoreFun().name} wins!")

}

You could also PostMan to test your API

[kpere@codingdojo.com](mailto:kpere@codingdojo.com)

[apiric@codingdojo.com](mailto:apiric@codingdojo.com)

[ohedaia@codingdojo.com](mailto:ohedaia@codingdojo.com)

This an Android Application, It will check the weather in the USA using zip code