**Coroutines**

**(Coroutine is the solution of asynchronous programming on android)**

* Before Learning about Coroutine we need to understand the basic work of application .

**Timeline

Description automatically generated**

**NOTE:**

* when ever application was started “Main Thread ” will start execution first.
* In “Main Thread ” we can perform the some of the tasks like button clicking ,small

Mathematical operations(+,-,\*,/,%,&&,|,||)and also fragment interactions .These kind of operations we can do it on the “Main Thread”.

* When come’s to the long term operations like “Network operations”,”File Commu..”,

“Database Executions” and “Server Interactions(like API’s)” we can’t perform on the

“Main Thread” .

* If we perform long term tasks on “Main Thread ” like more than “5 second” the app will be unresponcive.
* We will get the pop has below…

A picture containing text, indoor, computer

Description automatically generated

* To avoid Those conflicts we need to use Background Threads (or)Working Threads.

Timeline

Description automatically generated with medium confidence

**NOTE:**

* Here we are creating for each individual operation assigning single Thread.
* This is also not a good idea to assign each individual thing to single thread it will leads to the memory out of bounds .

Graphical user interface

Description automatically generated with medium confidence

* To over come this we are using the Only Single Thread called “BackGround” Thread.

Timeline

Description automatically generated

**NOTE:**

* Here we are using single Background thread to perform multiple tasks this is known as Coroutine in kotlin.

**What is Coroutines?**

**(Official Document Points)**

**(Coroutine is the solution of asynchronous programming on android)**

* On Android, coroutines help to manage long-running tasks that might otherwise block the “main thread” and cause your app to become unresponsive.
* Over 50% of professional developers who use coroutines have reported seeing increased productivity.
* A coroutine is a concurrency design pattern that you can use on Android to simplify code that executes asynchronously.
* [Coroutines](https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html) were added to Kotlin in version 1.3 and are based on established concepts from other languages.

**How to Define Coroutine?**

* Coroutine’s are not threads.
* We can communicate with one coroutine to another coroutine.
* We can create “N” number of coroutines in applications.

**A picture containing shape

Description automatically generated**

**How to use Coroutines?**

**Steps:**

* Create a new kotlin project.
* Add the dependency .

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

* Once u added it will integrated into your application.
* How to check whether.

Example: (With thread{})

**fun** main() {  
 *println*(**"Main Program Starts :${**Thread.currentThread().*name***}"**)  
 *//some code...  
 println*(**"Main program Starts:${**Thread.currentThread().*name***}"**)  
}

output:

Main Program Starts :main

Main program Starts:main

Example: (With thread{})

* This Example we are creating background thread.

**fun** main() {  
 *println*(**"Main Program Starts :${**Thread.currentThread().*name***}"**)  
  
 *thread* **{** *//creates a background thread ( worker thread)  
 println*(**"Fake work starts:${**Thread.currentThread().*name***}"**)

Thread.sleep(2000)

*//pretend doing some work like uploading a file.*

*println*(**"Fake work finished:${**Thread.currentThread().*name***}"**)  
 **}** *println*(**"Main program Starts:${**Thread.currentThread().*name***}"**)  
}

**NOTE:**

* In the above example we are creating worker thread using thread{}.
* Inside that we are using Thread.sleep(2000)=2 seconds where we can

Do the operations like file uploading,data base operations stuff.

* Here we need to observe the output .
* Both “MainThread” and “Thread-0(worker thread)” will work parallel to complete the execution “Main Thread ” will not wait for “Woker Thread ”to finish the task .it is has executed it’s part.
* But after executing also it is not finished the application it has waited for the

“Worker Thread ” to complete.

**How to implement Coroutine in the Kotlin:**

**Graphical user interface, text, application, email

Description automatically generated**

Note:

* Here we are using the “GobalScope.launch “ to create the Coroutine in background and inside background thread generally “MainThread” will

Not wait for the “Worker Thread” to finish so that’s why we are using

The “Thread.sleep(2000)=2seconds”.

* Here we need to import these two packages .

**import** kotlinx.coroutines.GlobalScope  
**import** kotlinx.coroutines.launch

**Output:**

Main Program Starts :main

Fake work starts:DefaultDispatcher-worker-1

Fake work finished:DefaultDispatcher-worker-1

Main program Starts:main

**delay()**

* **Delay()** function will be acts has **Thread.sleep()** but main difference is delay() will be used in the coroutine or in suspend() but not in other places like (MainThread).
* Delay() is having great feature has it will not pause the entire thread it will just pause the Coroutine.

Graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**What is suspend modifier?**

**What is suspending function()?**

Graphical user interface, text, application, email

Description automatically generated

|  |
| --- |
| * When we see the code of delay() in official documentation it is like this.   public suspend fun delay(  timeMillis: Long  ): Unit   * So this is the reason we are not able to make use it in the “MainThread” to overcome this we need to use the runBlocking(){}. |

Graphical user interface, text, application

Description automatically generated

**Note:**

|  |
| --- |
| * Here the we are creating coroutine’s in two places which is “GlobalScope.launch{}”🡪where coroutine is pause but “Thread is not blocked”. * But another place is ”runBlocking{}”🡪Here we are creating the coroutine+and also blocking the thread for to finish the previous coroutine. |

**Another-way of Doing: (Small Changes in code):**

**Graphical user interface, text, application, email

Description automatically generated**

**Another-way of Doing: (Small Changes in code):**

* This example will show how to use the rawBlocking{} has function expression.

**Graphical user interface, text, application, email

Description automatically generated**

* Here also we will get the same output.

**Suspend function creation:**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

Still lot more to explore!!!

Get a detailed understanding of using Coroutine builders which are nothing but the

higher-order functions used to create coroutines. - Explore launch, async, and runBlocking coroutine builders. –

Use of GlobalScope companion object which is used as GlobalScope.launch and GlobalScope.async function. –

Understand why using GlobalScope is highly discouraged? –

Difference between GlobalScope.launch() and launch().

Also between GlobalScope.async() and async(). –

Explore what exactly is the use of runBlocking builder. –

Write JUnit test cases for suspending function (suspend modifier) using runBlocking coroutine builder