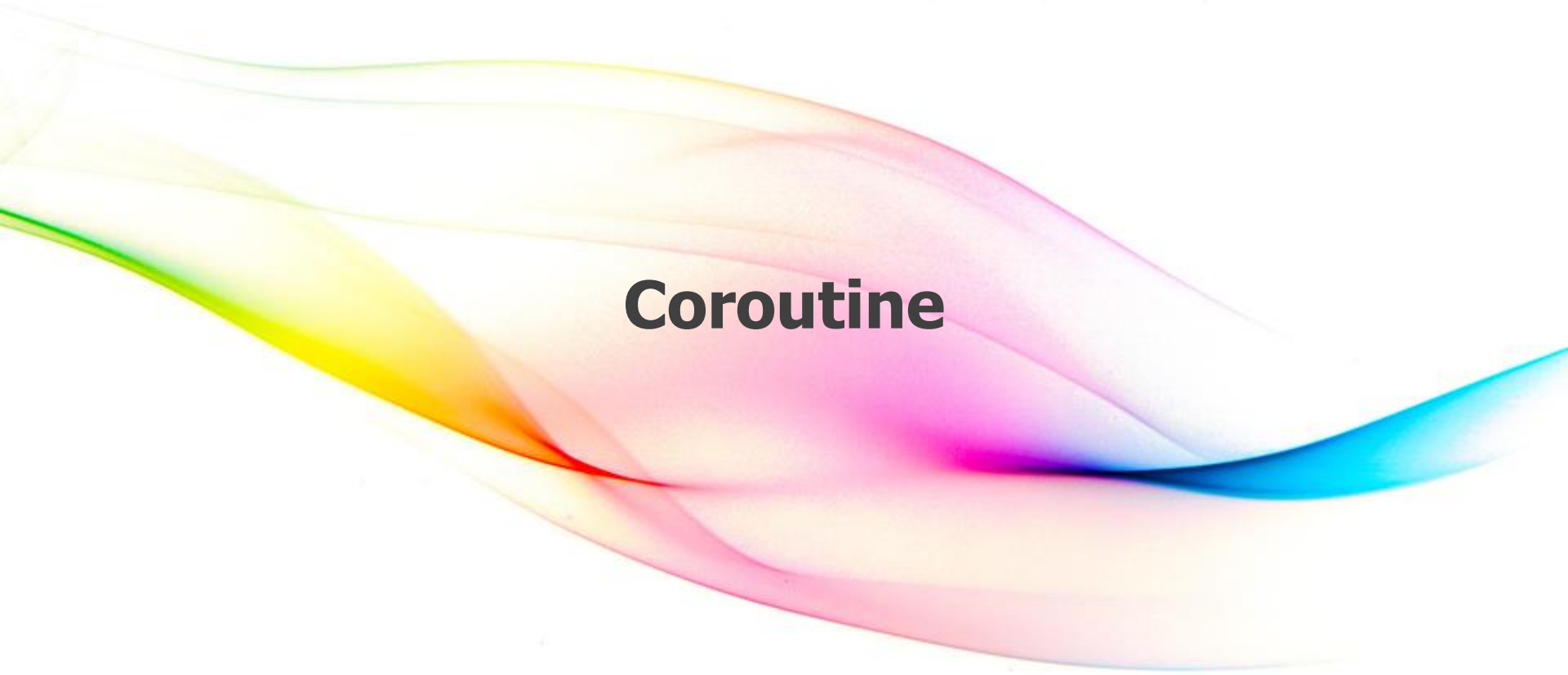


Coroutine

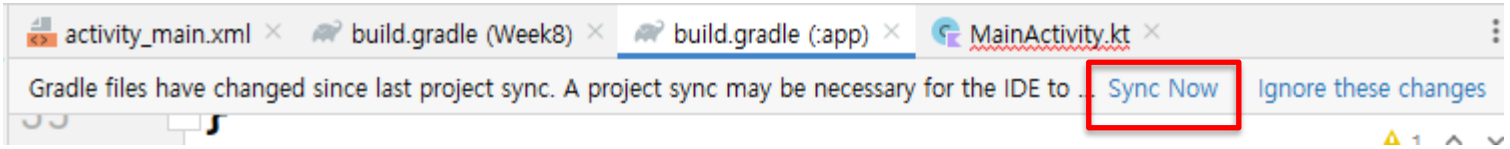
Mobile App Programming

Today's Contents

- Review
  - Gradle
- Multi-Threading
- Coroutine
- Lab practice

Gradle

- You can add dependency, or add some build configs
  - Mostly on build.gradle(Module: app)
- You need to "Sync Now" after changing gradle file



- To be handled & used later..

Concurrent and Parallelism

- **Concurrency** (동시성)
  - Concurrency relates to an application that is processing **more than one task at the same time**. Concurrency is an approach that is used for decreasing the response time of the system by using the **single processing unit.**

- **Parallelism** (병렬성)
  - Parallelism is related to an application where tasks are **divided into smaller sub-tasks that are processed seemingly simultaneously or parallel**. It is used to increase the throughput and computational speed of the system by using **multiple processors.**

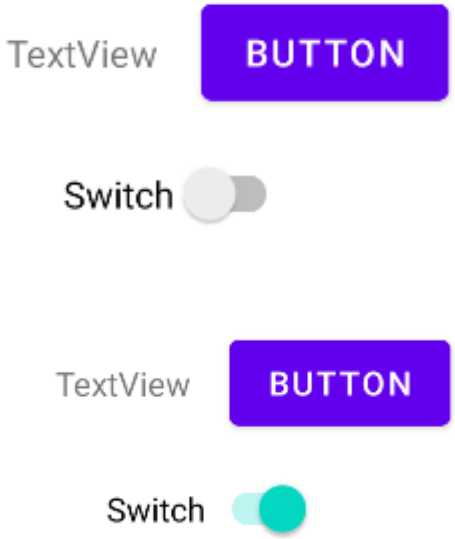
Concurrent Programming

- Nowadays, most application(not only mobile) are running many components at the same time.
  - But in this case, you need to take a deep consider into the critical section.
  - Details like **deadlock** and so on is not for our lecture
  - **In Android**, only **ONE** thread named **UI(or main) thread** can modify the user interface.
- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - If you do another task in other thread, it cannot modify UI.

https://gist.github.com/devquint/fe73034cb58b8f2ae770826beb192fe1

Exercise 1: Pause UI Thread

- UI thread matters!
  - If you do long task in UI thread, **UI will not work** for that time.
  - If you do another task in other thread, it cannot modify UI.
- Take a look at the given template UI.
  - Press a button
    - Execute a long task `Thread.sleep(5000)`
    - Try to modify left textview
  - Switch
    - Click while doing the long task
    - If UI stopped, it does not work



https://gist.github.com/devquint/fe73034cb58b8f2ae770826beb192fe1

Exercise 1: Pause UI Thread

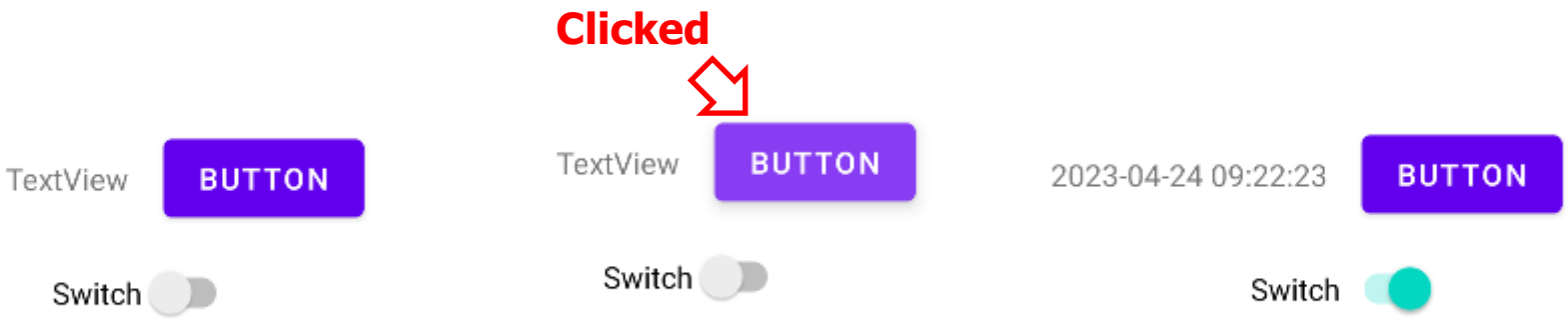
- UI thread matters!
  - If you do long task in UI thread, **UI will not work** for that time.
  - If you do another task in other thread, it cannot modify UI.
- **Thread.sleep(5000)** will stop thread for 5000 millisecond.
- Function getCurrentTimeString() will return time in String.

```
val button = findViewById<Button>(R.id.button)
val textView = findViewById<TextView>(R.id.textView)
button.setOnClickListener { it: View!
    textView.text = getCurrentTimeString() // button pressed
    Thread.sleep( millis: 5000)
    textView.text = getCurrentTimeString() // after 5000 ms
}
```

https://gist.github.com/devquint/fe73034cb58b8f2ae770826beb192fe1

Exercise 1: Pause UI Thread

- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - If you do another task in other thread, **it cannot modify UI.**



Exercise 1: Pause UI Thread

- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - If you do another task in other thread, **it cannot modify UI.**

- UI thread will do other task for 5 seconds (stopped).
  - While doing other task, UI thread cannot handle switch onClick function.
  - It will not reply for 5 seconds.
- **ANR(Application Not Responding)** could be happened.
  - You know, Not Responding(응답 없음) in other OS.

https://gist.github.com/devquint/4a56424997dc0106d8b7a4cbfa14c438

Exercise 2: Modify on non-UI Thread

- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - **If you do another task in other thread, it cannot modify UI.**
- Then, make Runnable to do on another thread.

```
button.setOnClickListener { it: View!
    textView.text = getCurrentTimeString() // button pressed
    val runnable = Runnable{
        Thread.sleep( millis: 5000)
        textView.text = getCurrentTimeString() // after 5000 ms
    }
    Thread(runnable).start() // start new thread
}
```

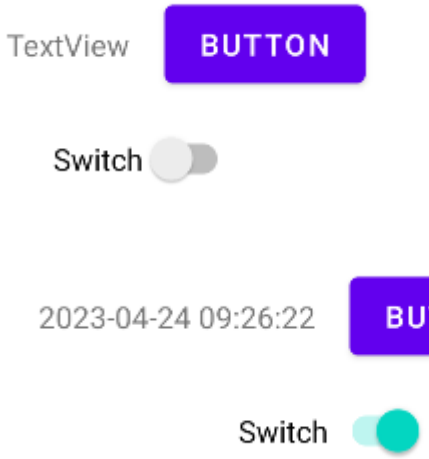
Makes a new runnable and executes it!

https://gist.github.com/devquint/4a56424997dc0106d8b7a4cbfa14c438

Exercise 2: Modify on non-UI Thread

- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - If you do another task in other thread, it cannot modify UI.

Only the original thread that created a view hierarchy can touch its views. (UI Thread)




```
FATAL EXCEPTION: Thread-2
Process: edu.skku.cs.week9, PID: 1185
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
    at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:8798)
    at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1606)
    at android.view.View.requestLayout(View.java:25390)
    at android.view.View.requestLayout(View.java:25390)
    at android.view.View.requestLayout(View.java:25390)
    at android.view.View.requestLayout(View.java:25390)
                .View.requestLayout(View.java:25390)
                .View.requestLayout(View.java:25390)
                straintlayout.widget.ConstraintLayout.requestLayout(ConstraintLayout.java:3605)
                .View.requestLayout(View.java:25390)
                et.TextView.checkForRelayout(TextView.java:9719)
                et.TextView.setText(TextView.java:6311)
                et.TextView.setText(TextView.java:6139)
                et.TextView.setText(TextView.java:6091)
                week9.MainActivity.onCreate$lambda$1$lambda$0(MainActivity.kt:31)
                week9.MainActivity.$r8$lambda$TUpXktNTW0rwoxbsM7Z9YBDiWg8(Unknown Source:0)
                week9.MainActivity$$ExternalSyntheticLambda1.run(Unknown Source:4) <1 internal line>
```

Exercise 2: Modify on non-UI Thread

- UI thread matters!
  - If you do long task in UI thread, UI will not work for that time.
  - If you do another task in other thread, it cannot modify UI.
- **While doing the 5-second-task, switch is working**
  - Since it is on the other thread, UI Thread is free enough to handle switch onClick function.
  - But "**the other thread**" is not UI Thread.
- Only UI Thread can modify the view
  - android.view.ViewRootImpl$**CalledFromWrongThreadException**: Only the original thread that created a view hierarchy can touch its views.

Coroutine

- Co(with/together) + Routine
  - Multiple subroutine(≒ function) at the same time
  - Just the term for programming
    (Neither Kotlin nor Android specific)
  - https://en.wikipedia.org/wiki/Coroutine
- **Kotlin Coroutine**
  - Kotlin support coroutine by native
  - Lightweight thread, asynchronous programming
  - https://kotlinlang.org/docs/coroutines-overview.html

Coroutine

- Kotlin **Coroutine + Android**
  - Android have additional supports too!
  - This lecture will **only deal with this one**.
  - https://developer.android.com/kotlin/coroutines

- **Coroutine**
  - Previous solution for contradiction related to UI thread was **AsyncTask**
  - But it was android specific and not efficient
    -> AsyncTask is deprecated…

Coroutine

- Dispatcher
  - There can be multiple coroutine at once
  - Dispatcher is responsible for properly assign a Thread to coroutine.
    - There could exist **multiple threads in one dispatcher** so that multiple coroutines in other threads could be run at the same time.
    - but also one thread can execute multiple coroutines takes turn.

Coroutine

- Dispatcher
  - There are pre-built dispatchers in Android.
  - Dispatchers.Main
    - Main(=UI) thread is in this dispatcher
    - Only interact with UI
  - Dispatchers.IO
    - Optimized to do disk or network I/O
  - Dispatchers.Default
    - Optimized to CPU-intensive tasks
  - We do not need to think about actual Thread Creation!

Coroutine

- Builder
  - launch
    - Use when result is not needed.
  - async
    - **await()** to wait for the result
    - Can get result via **Deferred<I>**
      - I: return type

Coroutine

- There are lot more details in the Coroutine
  - But let's just simply use it.
  - Official documents
    - https://kotlinlang.org/docs/coroutine-context-and-dispatchers.html
    - https://developer.android.com/kotlin/coroutines
  - Korean appendix
    - https://whyprogrammer.tistory.com/596
    - https://velog.io/@soyoung-dev/AndroidKotlin-%EC%BD%94%EB%A3%A8%ED%8B%B4-Coroutine
  - English appendix
    - https://kt.academy/article/cc-dispatchers
    - https://www.kodeco.com/34262147-kotlin-coroutines-tutorial-for-android-advanced

Coroutine

- Simple summary
  - Dispatcher automatically manage Coroutines to its Threads
  - We only need to care Dispatcher, not Thread
  - To be on UI Thread, use Dispatchers.Main
  - Not to be on UI Thread, use Disptachers.IO or Dispatchers.Default
  - Use launch{} block when return is not needed
  - Use async{} block and await() when return is needed

https://gist.github.com/devquint/6e2eb3636ee2d39625eb6d4853e4d5fc

Exercise 3: Coroutine

```
class MainActivity : AppCompatActivity() {
    fun getCurrentTimeString(): String{
        val time = Calendar.getInstance().time
        val formatter = SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss")
        return formatter.format(time)
    }

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

        val button = findViewById<Button>(R.id.button)
        val textView = findViewById<TextView>(R.id.textView)
        button.setOnClickListener { it: View!
            CoroutineScope(Dispatchers.Main).launch(){ this: CoroutineScope
                textView.text = getCurrentTimeString() // button pressed
                textView.text = longTask().await() // wait 5000 ms then change
            }
        }
    }

    fun longTask() = CoroutineScope(Dispatchers.IO).async{ this: CoroutineScope
        Thread.sleep( millis: 5000)
        getCurrentTimeString() ^async // Just value to return result of async. Not return ***.
    }
}
```

Exercise 3: Coroutine

- When we need to get result of time-consuming job.
  - Use Dispatchers.Default for that moment
    - In **exercise 3**, `Thread.sleep(5000)`
  - Use async to return the result
  - Take result with await() `getCurrentTimeString()`

- When we need to modify UI on UI Thread,
  - Use Dispatchers.Main at that moment
    - In **exercise 3**, we need to modify `textView.text`

https://gist.github.com/devquint/6e2eb3636ee2d39625eb6d4853e4d5fc

Exercise 3: Coroutine

- We need to get result of a time-consuming job
  - Use **Dispatchers.IO** in this exercise.
    - Dispatchers.IO for network/file I/O
    - Dispatchers.Default for CPU intensive work
  - **Use async to return the result**

- You can make block main with **async** then specify return with the final value (**without** return expression)

```
fun longTask() = CoroutineScope(Dispatchers.IO).async{ this: CoroutineScope
    Thread.sleep( millis: 5000)
    getCurrentTimeString() ^async // Just value to return result of async. Not return ***.
}
```

https://gist.github.com/devquint/6e2eb3636ee2d39625eb6d4853e4d5fc

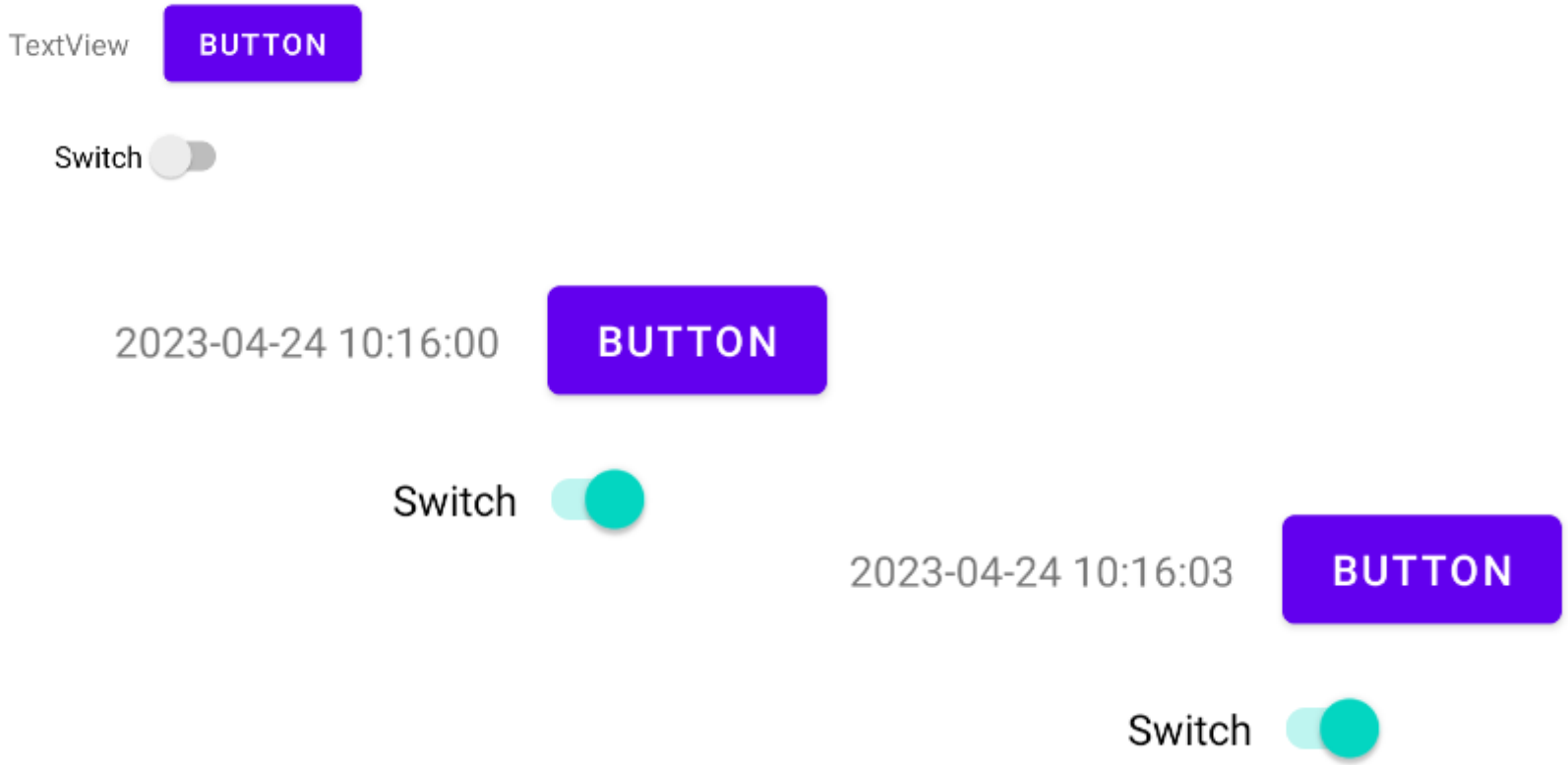
Exercise 3: Coroutine

- We need to modify UI on UI Thread
  - Use Dispatchers.Main at that moment
- We need to get result of time-consuming job
  - Take result with await()

```
button.setOnClickListener { it: View!
    CoroutineScope(Dispatchers.Main).launch(){ this: CoroutineScope
        textView.text = getCurrentTimeString() // button pressed
        textView.text = longTask().await() // wait 5000 ms then change
    }
}
```

Can modify TextView Here!

Exercise 3: Coroutine



Exercise 3: Coroutine

- Dispatchers.Main
  - It will set text when button is pressed,
  - Then await () for Dispatchers.IO returning the value
  - and set the text with that returned value.

- **Q. Why doesn't it make ANR?**
  - One coroutine in Dispatchers.Main waited for result!
  - That means, that coroutine stopped for 5 second

Exercise 3: Coroutine

- **Q. Why doesn't it make ANR?**
  - One coroutine in Dispatchers.Main waited for result!
  - That means, that coroutine stopped for 5 seconds.

```
button.setOnClickListener {
    CoroutineScope(Dispatchers.Main).launch(){
        textView.text = getCurrentTimeString() // button pressed
        textView.text = longTask().await() // wait 5000 ms then change
    }
}
```

This task is detached from UI thread when **await ()** is called.

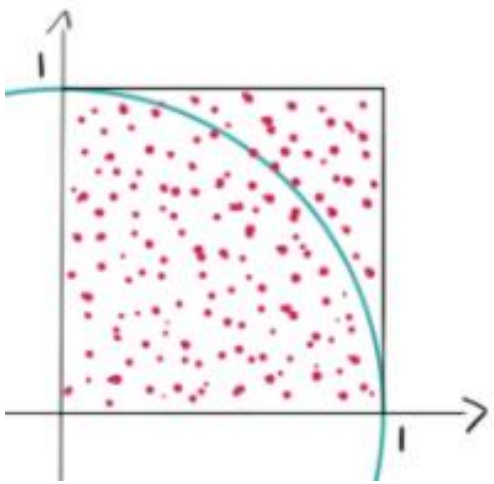
```
fun longTask() = CoroutineScope(Dispatchers.IO).async{
    Thread.sleep(5000)
    getCurrentTimeString() // Just value to return result of async. Not return ***.
}
```

Instead, **a thread of IO Dispatchers** executes longTask()

https://stackoverflow.com/questions/53752991/is-await-blocking-the-ui-thread-on-android

[Lab-Practice #9] Monte Carlo

- Estimate Pi using Monte Carlo Simulation
  - Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables.
- How to estimate pi?

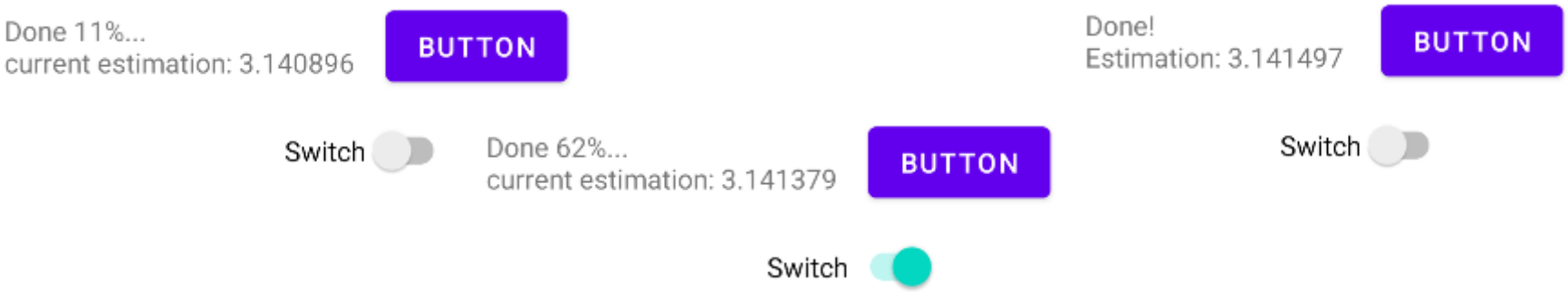


1) Get two random float values range 0~1 (x, y)
2) If (x,y) is inside of circle, $\sqrt{x^2 + y^2} \leq 1$
3) If (x,y) is outside of circle, $\sqrt{x^2 + y^2} > 1$
4) Run (1)-(3) many times
5) Calculate (number of dots inside) / (total dots)
6) The above will be quarter of pi
   (because it is area of quarter circle of radius 1)

[Lab-Practice #9] Monte Carlo

- Estimate Pi using Monte Carlo Simulation
  - Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables.

Execution Examples -



[Lab-Practice #9] Monte Carlo

- Use the same layout as exercises
  - After clicking the button, simulate Monte-Carlo **100,000,000 times**,
    - For each **1,000,000 times**, change the TextView text to **"Done ${x}%... □n current estimation: ${pi value until 6 decimal places}"**
    - After it finishes, change the TextView text to **"Done! □n Estimation: ${pi value until 6 decimal places}"**
  - You must **randomly generate** x and y value

[Lab-Practice #9] Monte Carlo

- Tips
  - intValue.toDouble() to cast Integer value to Double
  - Math.random() to generate random value
    - It will return double value between 0.0f and 1.0f
  - String.format("%.*f", doubleValue) for precision
    - String.format("%.6f", value) will return String until 6 places
  - You can do your own way to pass intermediate value
    - You can CoroutineScope(Dispatchers.Main).launch in non-UI coroutine to change UI. **[RECOMMENDED]**
    - You can use Channel: https://medium.com/swlh/kotlin-coroutines-in-android-channel-fb9b3b65e0b
    - Other methodologies are OK.

[Lab-Practice #9] Monte Carlo

- Criteria
  - Show your code: it must use Coroutine!
  - Execution
    - **Run your application -> press button**
    - **It must show both intermediate status and final estimation**
    - **While estimating, switch must be switchable!**
    - **PI value can differ because this is estimation**
  - Hint? (Spoiler)
    - https://gist.github.com/devquint/442d414d206030043b6c51e4dab82ad5