Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Coroutines in Kotlin

Agenda

- ❑Background
- ❑Introduction
- ❑The Terminology
- ❑Sample Code
- ❑Case Study

Agenda

- ❏ Background
- ❏ Introduction
- ❏ The Terminology
- ❏ Sample Code
- ❏ Case Study

Background

Background

- ❑ **Asynchronous programming** is very important for modern applications. Using it increases the amount of work your app can perform in **parallel**
- ❑ This in turn allows you to run heavy-duty tasks away from the **UI thread**, in the background
- ❑ By doing so, you avoid UI freezes, and provide a fluid experience for your users (good UX)

Background

- ❑ Android provides several asynchronous programming mechanisms, but it’s difficult to find the most appropriate one to use
- ❑ Some mechanisms have a huge learning curve. Others require a ton of boilerplate code to implement, and aren’t that concise
- ❑ This all affects the scalability of your app, and increases the cognitive load for new developers
- ❑ It’s best if the APIs you rely on are easy to use and scale when needed

Background

- ❑ Because all platforms on the JVM had the same problem, the team from JetBrains has come up with a new API.
- ❑ The idea behind it is to help you solve all these problems, without a steep learning curve.
- ❑ Here, hopefully, ☺ you'll learn about that API – **Kotlin Coroutines**

Recap

- ❑ Coroutines help you to write asynchronous code in a more natural, synchronous way

- ❑ That is, in a sequential style of programming, which is more humanly-understandable and readable

- ❑ They offer a number of benefits, which we'll cover later

Coroutines in Kotlin

Introduction

- In Android, coroutines are used mainly to solve two frequently faced problems, namely
  - Long-running tasks that could **block** the main thread
  - Disk operations from the main thread.

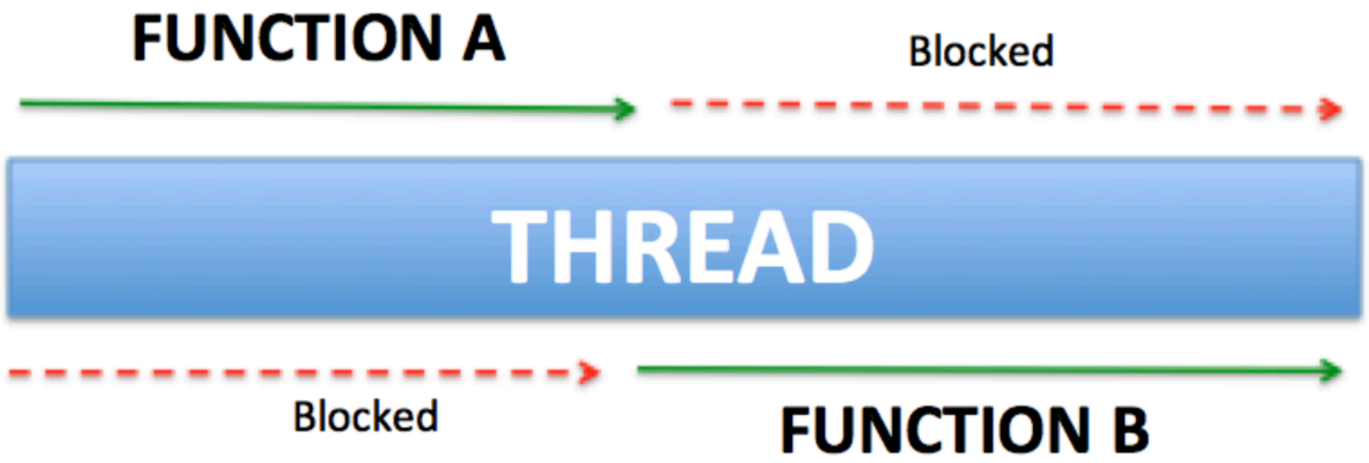
Introduction

- The documentation says Kotlin Coroutines are like lightweight threads. They are lightweight because creating coroutines doesn’t allocate new threads.
- Instead, they use predefined thread pools, and smart scheduling (the process of determining which piece of work you will execute next)
- Additionally, coroutines can be *suspended* and *resumed* mid-execution
- Creating a large number of Kotlin Coroutines won’t bring unnecessary memory overhead to your program

Introduction – Suspending Vs Blocking

- ❑ Suspension and blocking sound similar, but they’re actually very different.
- ❑ A **blocking** call to a function means that a call to any other function, from the same thread, will **halt the parent’s execution.**
- ❑ So if you make a blocking call on the main thread’s execution, you effectively freeze the UI. Until that blocking calls finishes, the user will see a static screen, which is not a good thing to say the least!
- ❑ You can visualize it like this:

Introduction – Suspending Vs Blocking

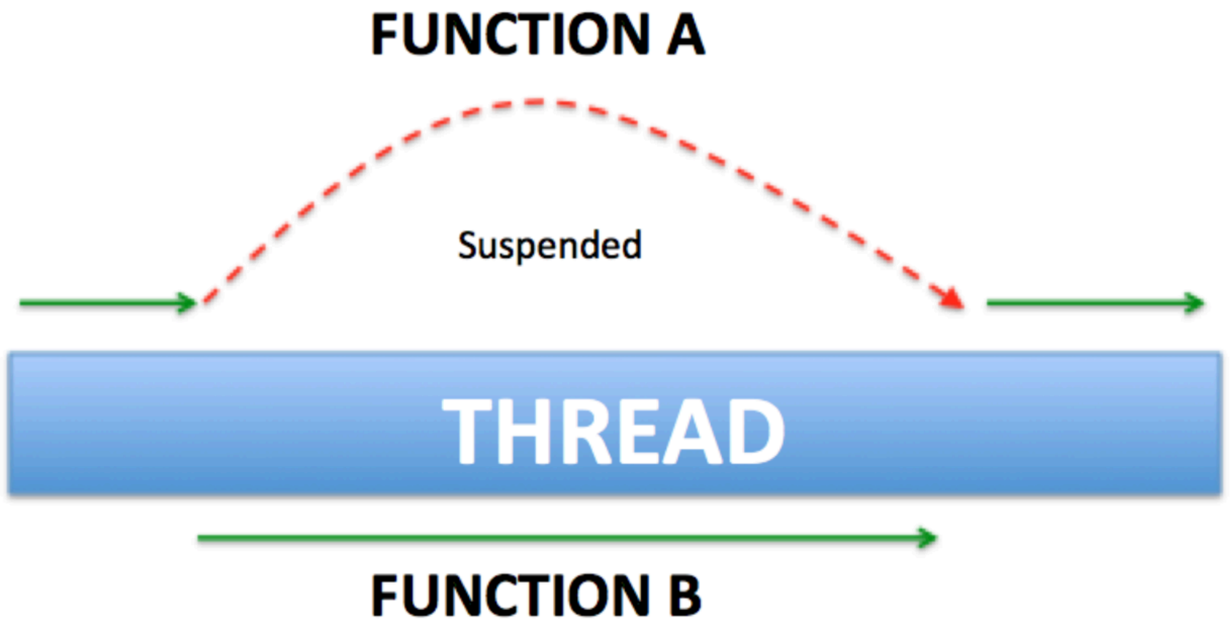


BLOCKING: Function A has to be completed before Function B continue. The thread is locked for Function A to complete it's execution.

Introduction – Suspending Vs Blocking

- ❑ On the other hand, **suspending** doesn’t necessarily block your parent function’s execution.
- ❑ If you call a **suspending** function in some thread, you can easily push that function to a different thread. If it’s a heavy operation, it won’t block the main thread.
- ❑ If you require a result from the function, you can bridge back to the main thread, without a lot of code. That way you can fetch data in a coroutine, from the main thread.
- ❑ You can visualize that like this:

Introduction – Suspending Vs Blocking



SUSPENDING: Function A, while has started, could be suspended, and let Function B execute, then only resume later. The thread is not locked by Function A.

The Terminology

Coroutine Terminology

Suspending function

A Function that is marked with suspend modifier, it may suspend execution of the code without blocking the current thread of execution by invoking other suspending functions. A suspending function cannot be invoked from a regular code, but only from other suspending functions and from suspending lambdas.

Suspending lambda

This is the block of code that runs in a coroutine. It is the Short syntactic form for an anonymous suspending function. Looks exactly like an ordinary lambda expression but with a functional type marked with the suspend modifier

Coroutine Terminology

Suspending function type

Function type for suspending functions and lambdas

It is just like a regular function type, but with suspend modifier.

Suspendable point

A point during coroutine execution where the execution of the coroutine may be suspended. All points in coroutines where suspendable functions are invoked are suspendable points.

Coroutine Terminology

Continuation

Is the state of the suspended coroutine at the suspension point. When a suspendable function is invoked in a coroutine, it suspends the coroutine. Continuation represents the rest of its execution after the suspension point.

Dispatchers

A Dispatchers can be thought of as the thing running the coroutine. They are to Coroutines what the thread pool is to Threads. There are four dispatchers in coroutines namely main, IO, default and unconfined. In Android, the main dispatcher is the android main thread, the default dispatcher for CPU bound tasks which is fixed to the number of cores on the device while the IO is fixed to 64 threads.

Coroutine Terminology

Job

A cancellable thing which represents a background job. It is used when we care more about the side effects and execution process than the return value of a background process.

It has a life-cycle which starts from either new or active state and ends in its completion state. Instances of a Job can either be created with launch coroutine builder or with a `Job()` factory function. Jobs can be used to define the parent-child relationship of coroutines. Passing a job as a coroutine context parameter to a builder will break the parent-child relationship.

Coroutine Terminology

CoroutineScope

An interface which contains a single property only — coroutineContext

It defines the scope for new coroutines

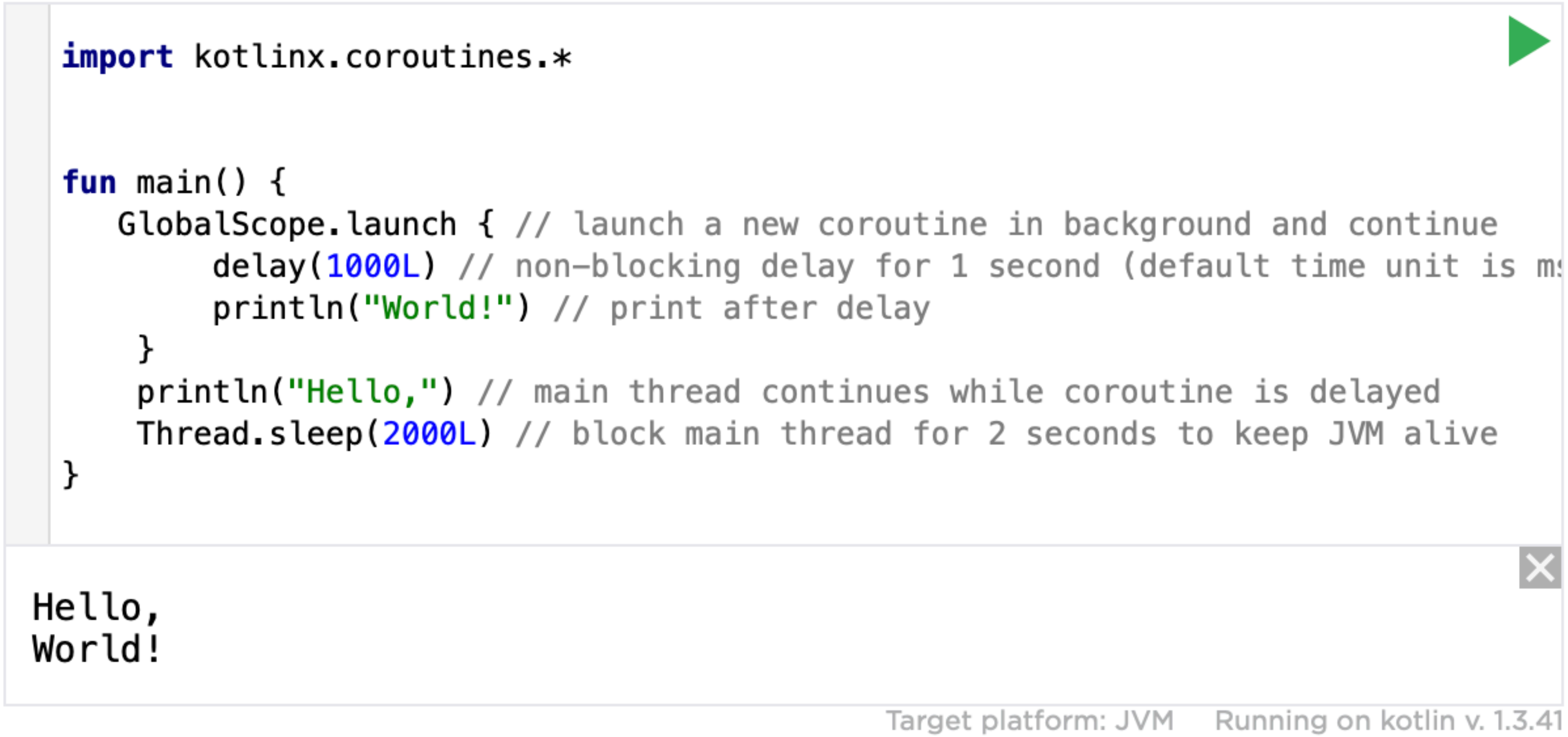
CoroutineContext

Is an indexed set of Element instances. An indexed set is a mix between a set and a map where every element in this set has a unique Key. The CoroutineContext itself is immutable. Its elements are the Job of the coroutine, continuationInterceptor(mainly dispatchers) and coroutineExceptionHandler. New CoroutineContext can be created from the addition of new element using the plus operator which produces a new set containing the newly added one.

Coroutine Terminology

CoroutineBuilders:
These take a suspending lambda as an argument to create a coroutine. There are a bunch of coroutine builders provided by Kotlin Coroutines, including **async()**, **launch()**, **runBlocking**.

Coroutine Example



```
import kotlinx.coroutines.*


fun main() {
   GlobalScope.launch { // launch a new coroutine in background and continue
        delay(1000L) // non-blocking delay for 1 second (default time unit is ms
        println("World!") // print after delay
    }
    println("Hello,") // main thread continues while coroutine is delayed
    Thread.sleep(2000L) // block main thread for 2 seconds to keep JVM alive
}
```

```
Hello,
World!
```

Target platform: JVM   Running on kotlin v. 1.3.41

References

Sources:

https://antonis.me/2018/12/12/an-introduction-to-kotlin-coroutines/
https://proandroiddev.com/kotlin-coroutines-channels-csp-android-db441400965f
https://superkotlin.com/coroutines/
https://www.codementor.io/jimmy_chuks/introduction-to-coroutines-xr674rfpo
https://www.raywenderlich.com/1423941-kotlin-coroutines-tutorial-for-android-getting-started
https://android.jlelse.eu/kotlin-coroutines-threads-concurrency-and-parallelism-101-78a56e09d373

Thanks.