

Programming Ubiquitous Things 2025/26 - Spring 2025 - UiO

Lab 3 – Kotlin Coroutines

Lyla Naghipour Vijouyeh

lylan@ifi.uio.no

UiO/IFI/PTG



Long-running tasks

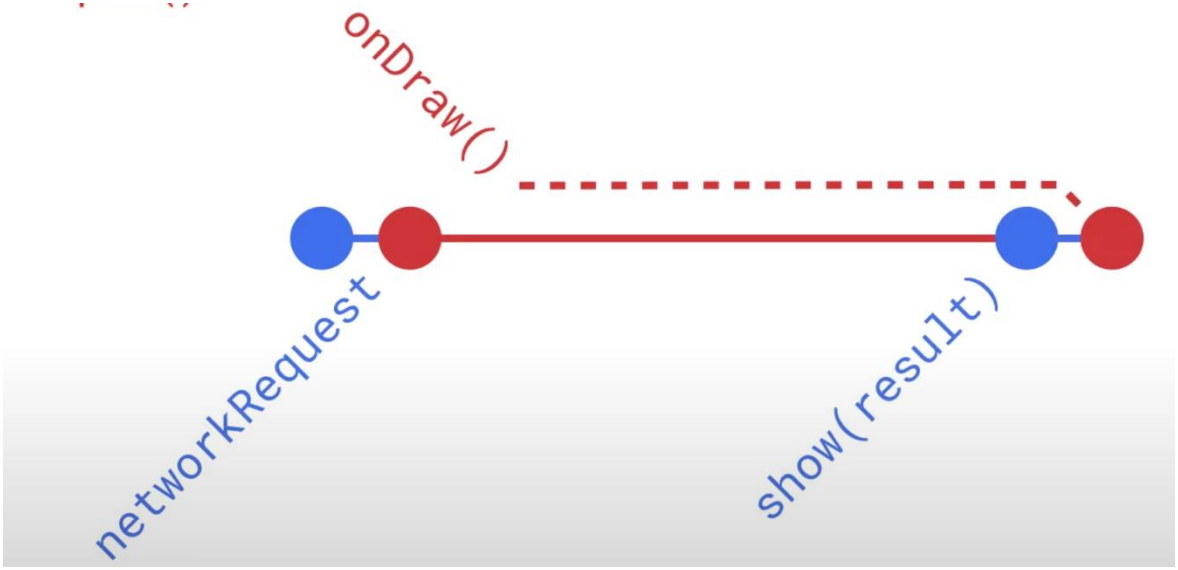
- Long-running tasks block the whole UI:
  - No event can be dispatched (the app appears to hang)
  - If blocked for too much time, the “Application Not Responding” (ANR) dialog pops up

- Long-running tasks such as:
  - Writing data to a database
  - Fetching data from the network
  - Parsing large JSON datasets

1st Solution: Blocking style

- Return the result directly from the Netwotk request blocks the main thread
- The main thread do not respond

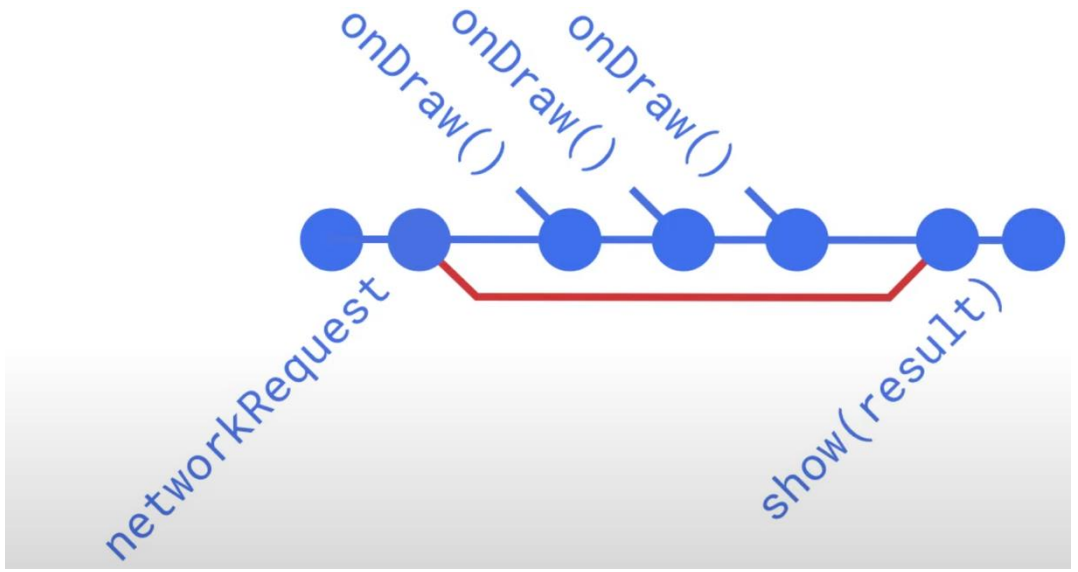
```
fun onDataNeeded() {
    val result = networkRequest()
    show(result)
}
```



2nd Solution: Callbacks

- Using callback prevents our app to block
- Networking library run the Network request on another thread and when the data is ready it will pass the data or callback to the main thread
- The callback is the handle that the library uses to pass data from the background thread to the main thread

```
fun onDataNeeded() {
    networkRequest() { result ->
        show(result)
    }
}
```



Callbacks issues

- Need much work
- Can be overwhelming when many callbacks are used in the same function
- Not using exceptions

3rd Solution: Coroutines

- Coroutines replace callbacks
- Provide Main safety:
  - You can write Coroutines-based functions to make a network request and call it from a main thread
- Coroutines prevent the Main thread (UI thread) pauses and block
- Coroutines convert async callbacks for long-running tasks into *sequential* code.
- Use exceptions

Coroutines instead of callbacks

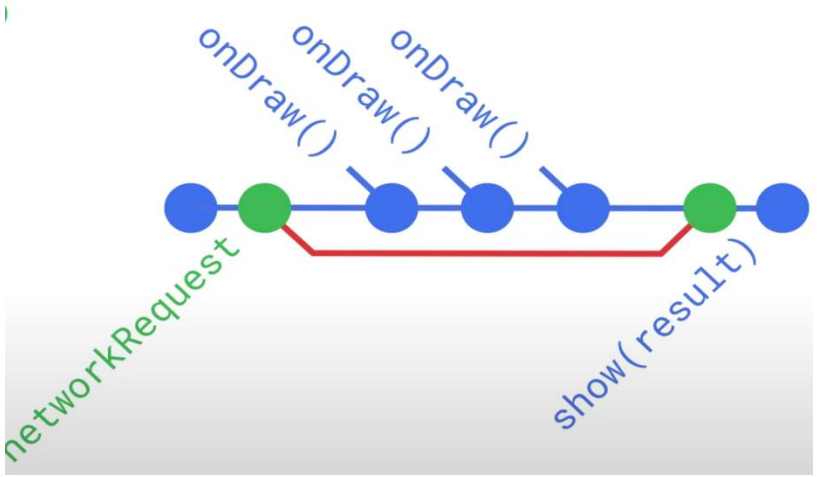
- With Coroutines, you need to suspend the function.
  - A suspend function, instead of blocking the main thread when network request is called, suspends the Coroutine
- The keyword suspend in Kotlin:
  - For marking a function available to coroutines

```
suspend fun onDataNeeded() {
    val result = networkRequest()
    show(result)
}

suspend fun networkRequest() {
    ..
}
```

Networking library with Coroutines

- It still uses another thread to run the Network request
- The main difference:
  - The code is simpler
- Free up the main thread to handle onDraw, and the user touches
- When the result is ready, it will get passed back (instead of a callback), and the Coroutine will resume



Storing states of suspneded Coroutines

- When Kotlin executes OnDataNeeded(), it keeps track of the place it starts the Coroutine.
- Then it executes each line of the code until it reaches a call to another suspend function.
- Before it calls the suspend function, it copies the current state
- This state is seen as a callback that knows how to resume the current function.
- Then Kotlin will run Network requests until it suspends itself to make the Netwerk call.

How Coroutine works

- The Coroutines on the main are suspended so:
  - Main is free to handle UI events when the network request comes back
  - Kotlin will resume these Coroutines.
  - First, it will resume the Network request, and it does this by restoring the state it's saved earlier.
  - Once the Network request is made, it will pass it to OnDataNeeded(), and the result will be available immediately.
- If the Network request fails, it will throw an exception; then, the Coroutine will continue executing just like a normal function until it returns.

Adding Coroutines

- Coroutines on Android are available as:
- a core library
  - **kotlinx-coroutines-core** — Main interface for using coroutines in Kotlin
- Android specific extensions:
  - **kotlinx-coroutines-android** — Support for the Android Main thread in coroutines

```
dependencies {
  ...
  implementation "org.jetbrains.kotlinx:kotlinx-coroutines-core:x.x.x"
  implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:x.x.x"
}
```

CoroutineScope

- In Kotlin, all Coroutines run inside a CoroutineScope which controls the lifetime of coroutines
- When you cancel the job of a scope, it cancels all coroutines started in that scope.
- Scopes specify a default dispatcher

Kotlin dispatchers for Coroutines



- A dispatcher controls which thread runs a coroutine
- **Dispatchers.Main** — Use this dispatcher to run a coroutine on the main Android thread. This should be used only for interacting with the UI and performing quick work.
- **Dispatchers.IO** — This dispatcher is optimized to perform disk or network I/O outside of the main thread.
- **Dispatchers.Default** — This dispatcher is optimized to perform CPU-intensive work outside of the main thread.