Dispatchers

Dispatchers are responsible for “dispatching” coroutines
to the underlying threads

Threads are independent concurrent tasks

Standard Dispatchers in Android:

Dispatchers.Main

Dispatchers.Main.immediate

Dispatchers.Default

Dispatchers.IO

Dispatchers.unconfined

Main Dispatcher

Dispatchers.Main executes code on the UI (main) thread of your Android application

Dispatchers.Main.immediate executes code on the UI (main) thread of your Android application as well

What's the added benefit of "immediate" Main Dispatcher?

`Dispatchers.Main` – behaves like `Handler(Looper.getMainLooper()).post(…)`

`Dispatchers.Main.immediate` – behaves like `Activity.runOnUiThread(…)`

Regular dispatch:

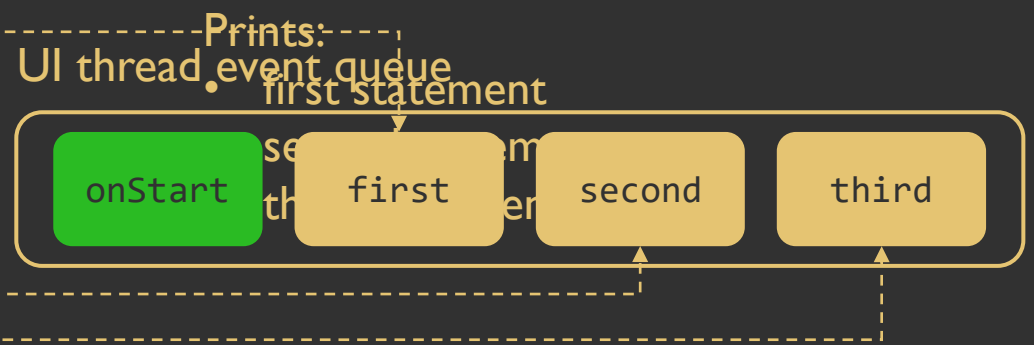
```
override fun onStart() {
    super.onStart()

    val coroutineScope = CoroutineScope(Dispatchers.Main)

    coroutineScope.launch(Dispatchers.Main) {
        println("first statement")
    }

    coroutineScope.launch(Dispatchers.Main) {
        println("second statement")
    }

    coroutineScope.launch(Dispatchers.Main) {
        println("third statement")
    }
}
```



Immediate dispatch:

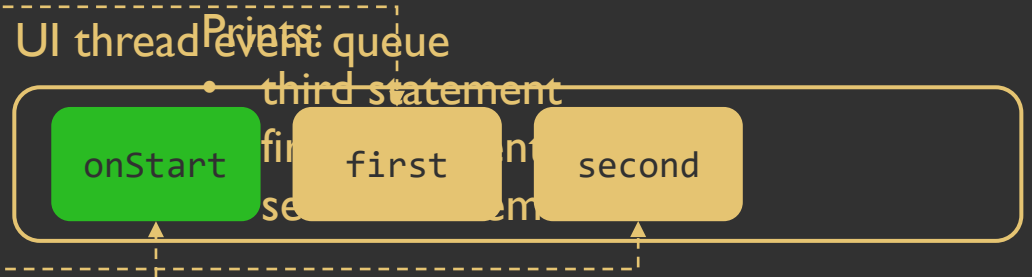
```
override fun onStart() {
    super.onStart()

    val coroutineScope = CoroutineScope(Dispatchers.Main)

    coroutineScope.launch(Dispatchers.Main) {
        println("first statement")
    }

    coroutineScope.launch(Dispatchers.Main) {
        println("second statement")
    }

    coroutineScope.launch(Dispatchers.Main.immediate) {
        println("third statement")
    }
}
```



Use Dispatchers.Main.immediate to execute code on UI thread, unless you have a reason not to

Dispatchers.Default and Dispatchers.IO

“Background” dispatchers:

`Dispatchers.Default:`

- Thread pool with `maxThreads = max(2, NUM_OF_CPU_CORES)`
- Used for computation-intensive tasks

`Dispatchers.IO:`

- Thread pool with `maxThreads = max(64, NUM_OF_CPU_CORES)`
- Maximum number of threads can be further increased by adjusting system proprties
- Used for IO tasks (tasks which are mostly “waiting”)

Dispatchers.Unconfined

Unconfined execution from UI thread:

```
override fun onStart() {
    super.onStart()

    val coroutineScope = CoroutineScope(Dispatchers.IO)

    coroutineScope.launch {
        println("scope default executes on ${threadName()}")
    }

    coroutineScope.launch(Dispatchers.Default) {
        println("default executes on ${threadName()}")
    }

    coroutineScope.launch(Dispatchers.Main) {
        println("main executes on ${threadName()}")
    }

    coroutineScope.launch(Dispatchers.Unconfined) {
        println("unconfined executes on ${threadName()}")
    }
}
```

Prints:

- scope default executes on DefaultDispatcher-worker-1
- default executes on DefaultDispatcher-worker-2
- unconfined executes on main
- main executes on main

Unconfined execution from a background thread:

```
override fun onStart() {
    super.onStart()

    val coroutineScope = CoroutineScope(Dispatchers.IO)

    Thread {
        coroutineScope.launch(Dispatchers.Main.immediate) {
            println("main immediate executes on ${threadName()}")
        }

        coroutineScope.launch(Dispatchers.Unconfined) {
            println("unconfined executes on ${threadName()}")
        }
    }.start()
}
```

Prints:

- unconfined executes on Thread-5
- main immediate executes on main

I can’t think of a reasonable use case for Dispatchers.Unconfined in Android applications

Dispatching Strategy for Android Apps

To execute code on UI thread,
use Dispatchers.Main.immediate

Unfortunately, background execution using “standard” CoroutineDispatchers is not straightforward

“Background” dispatchers:

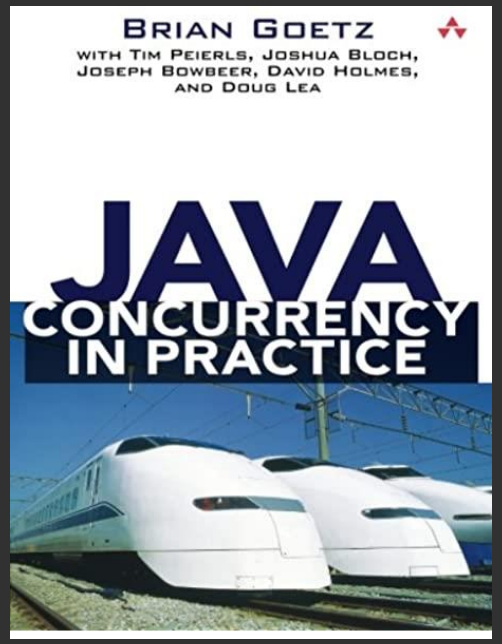
`Dispatchers.Default:`

- Thread pool with `maxThreads = max(2, NUM_OF_CPU_CORES)` **Why?**
- Used for computation-intensive tasks **Why?**

`Dispatchers.IO:`

- Thread pool with `maxThreads = max(64, NUM_OF_CPU_CORES)` **Why?**
- Maximum number of threads can be further increased by adjusting system properties **Why?**
- Used for IO tasks (tasks which are mostly “waiting”) **Why?**

The best book to learn about JVM concurrency and performance:



Concurrency optimization is a delicate and time-consuming process…

…which is completely irrelevant for absolute majority of Android applications

To execute code in background,
use single custom unbounded CoroutineDispatcher!

I will continue using Dispatchers.Default and Dispatchers.IO in demonstrations for rest of this course