kotlinx.coroutines

Introduction + Basic concepts.

Victor Olmo Gallegos Hernández
Android Developer at GoMore

@voghDev

voghDev

GoMore

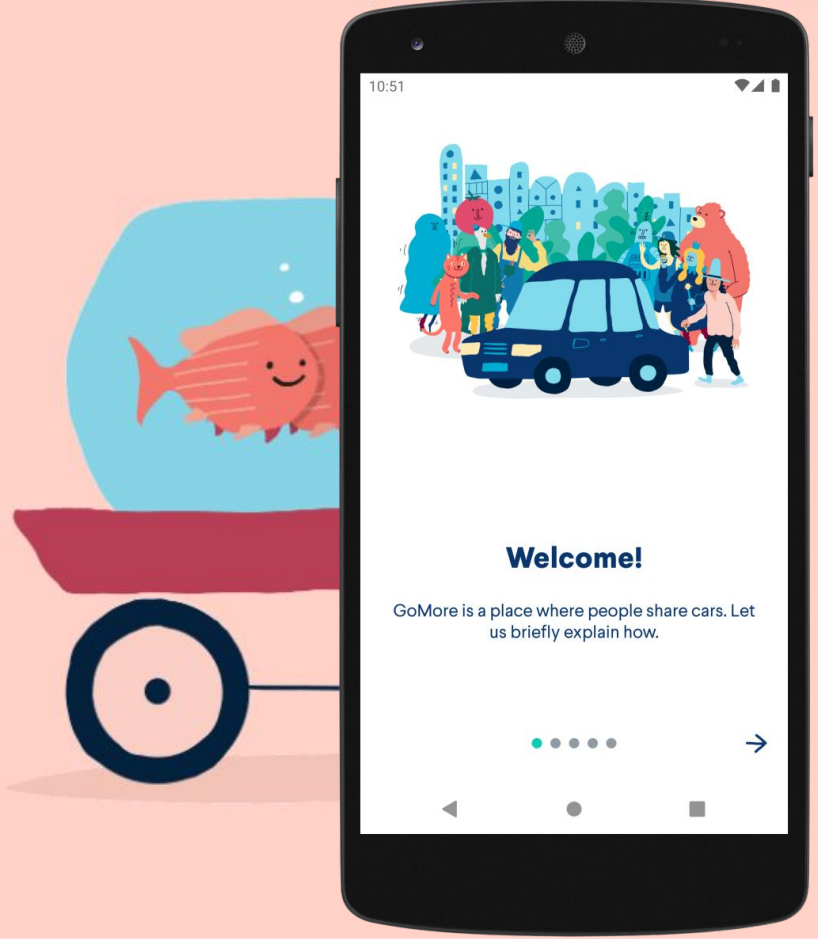
¿Dónde quieres ir?

Nosotros te llevamos

Viaja
conociendo gente increíble por toda España →

Alquila
el coche perfecto cerca de ti →

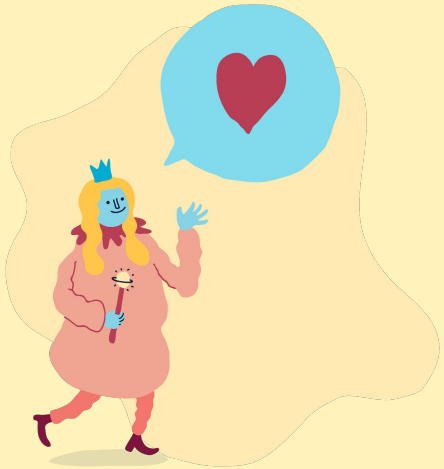
Píllate un Renting
con todo incluido →



kotlinx.coroutines

Index

- What is kotlinx.coroutines?
- Motivation: Why this talk?
- async/await
- Basic concepts
- Testing with coroutines

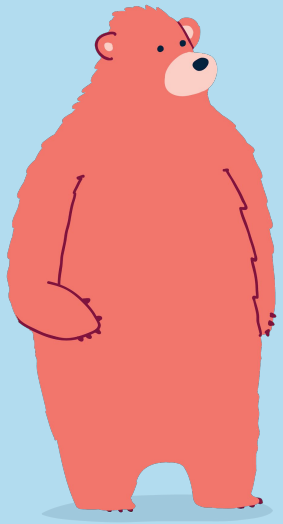


Introduction.

What is kotlinx.coroutines?

kotlinx.coroutines is a **Threading** library.

Developed by JetBrains in early 2017



Introduction.

What is kotlinx.coroutines?

According to **documentation...**

“kotlinx.coroutines is a rich library for coroutines developed by JetBrains. It contains a number of high-level coroutine-enabled primitives that this guide covers, including launch, async and others”

Introduction.

What is kotlinx.coroutines?

According to **documentation...**

“Coroutine Basics - Run the following code”

Introduction.

What is kotlinx.coroutines?

According to **documentation...**

“Coroutine Basics - Run the following code”

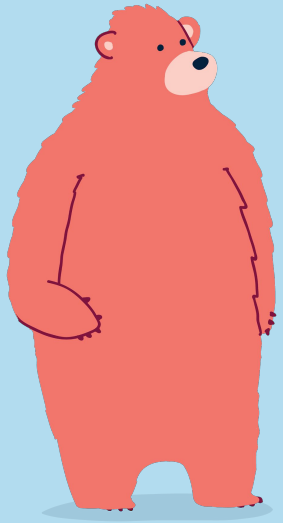
```
import kotlinx.coroutines.*

fun main() {
    GlobalScope.launch {
        delay(1000L)
        println("World!")
    }
    println("Hello,")
    Thread.sleep(2000L)
}
```

Introduction.

What is kotlinx.coroutines?

According to **documentation...**

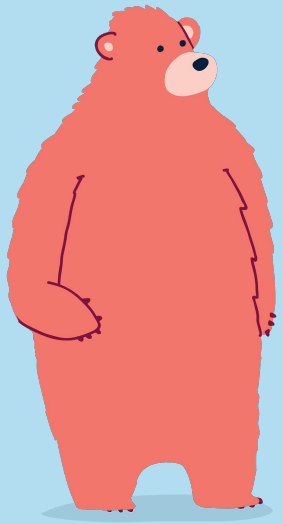
“Essentially, coroutines are light-weight threads”



Motivation

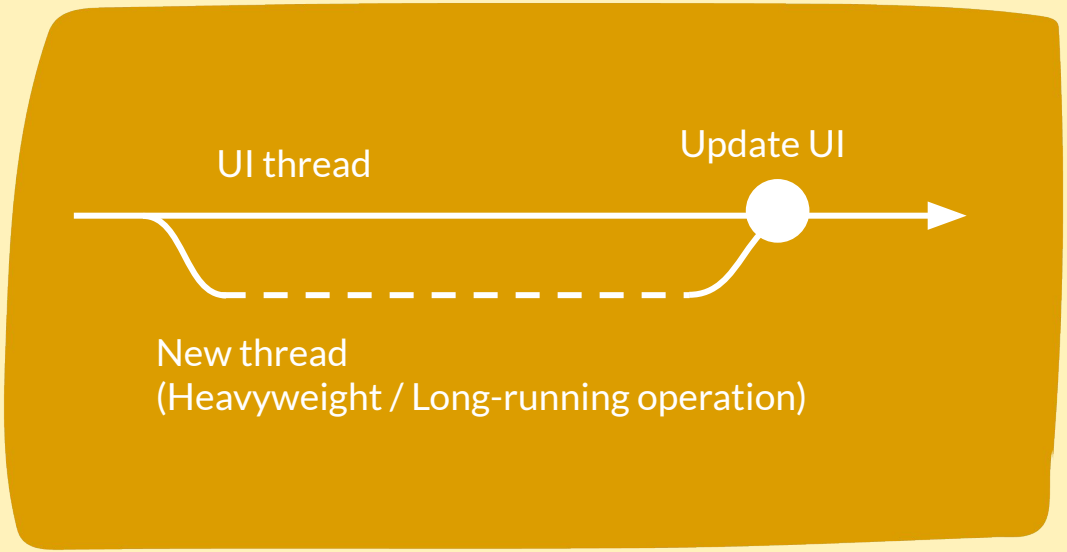
Why this talk?

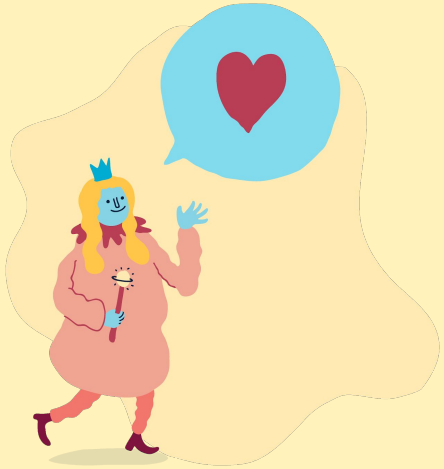
- Started with early Coroutines (0.11 experimental)
- async/await ➡ better than Callback hell
- Knowledge was very widespread
- Concepts?
- Decided to create my own resource



The problem.

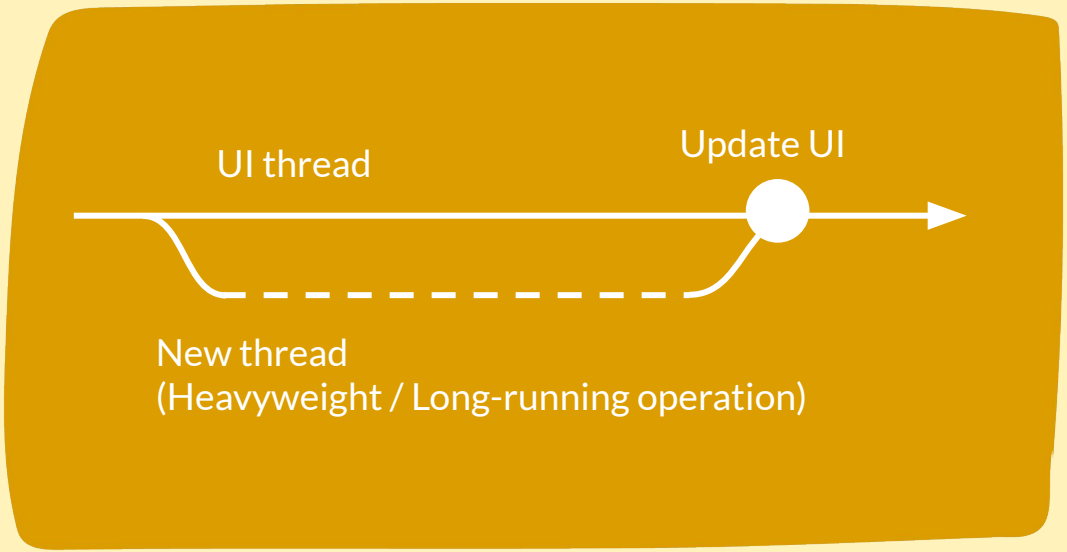
Or one of them.

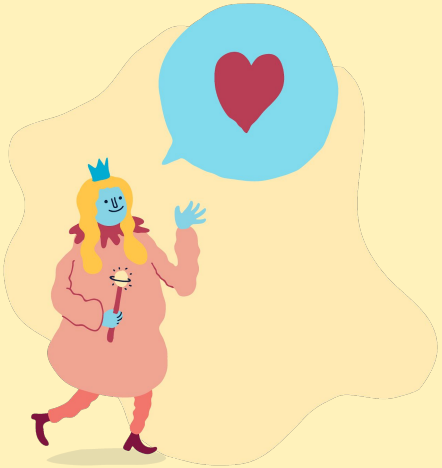




Suspending function

Can suspend the execution of a coroutine





```
suspend fun getRentalCars(): List<Car> = apiClient.getRentalCars()
```

Concepts.

Essentials every kotlinx.coroutines client must know.

- CoroutineContext
- CoroutineDispatcher
- CoroutineScope

- CoroutineBuilders
- Job
- CompletableJob
- SupervisorJob()
- Deferred

CoroutineContext.

Specific execution Context for a coroutine

- A Set of elements associated to each coroutine
- Coroutines don't work as **threads**, they have **Context** instead
- Essentially, a Key-Value map
- *"Persistent Context for the coroutine"*
- *"Indexed set of Element instances, mix between a Set and a Map"*

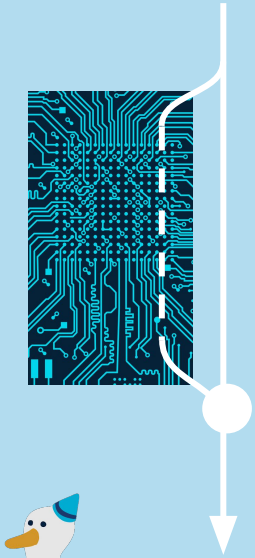
- Four default **CoroutineContext**s provided by the library
- You can create your own in case you need

CoroutineContext.

Specific execution Context for a coroutine

- A Set of elements associated to each coroutine
- Coroutines don't work as **threads**, they have **Context** instead
- Essentially, a Key-Value map
- *"Persistent Context for the coroutine"*
- *"Indexed set of Element instances, mix between a Set and a Map"*

- Four default **CoroutineContext**s provided by the library
- You can create your own in case you need



CoroutineContext.

Four default Contexts provided by the library

Default *IO*

Main *Unconfined*

CoroutineDispatcher.

"These lovely actors who treat our coroutines"

- Sends our coroutine to its destination Context.
- You don't specify a **Context** for your coroutine, you specify a **Dispatcher** instead
- *"Base class that shall be extended by all coroutine dispatcher implementations."*

CoroutineDispatcher.

Four standard **Contexts** - four standard **Dispatchers**

*Dispatchers.**Default***

*Dispatchers.**IO***

*Dispatchers.**Main***

*Dispatchers.**Unconfined***

CoroutineDispatcher.

Four standard **Contexts** - four standard **Dispatchers**

Example

```
val job = launch(Dispatchers.Default) {
    getRentalCars()
}
```

CoroutineScope.

“Parent” of a coroutine.

- Determines the lifecycle of a coroutine
- It is the “Timeline” where the coroutine is attached.
- If the Scope is **destroyed**, all child coroutines are canceled

- Examples (Android): Activity, Fragment, Application, CustomView
- Application-wide scope: **GlobalScope**
- Custom Scopes

CoroutineScope.

“Parent” of a coroutine.

- It is not recommended to override CoroutineScope
- Instead, use inheritance by delegation from **MainScope()** and **CoroutineScope()** factory functions

```
class MyScope : CoroutineScope {
    val job = Job()
    val coroutineContext = Dispatchers.Main + job
}
```

```
class MyScope : CoroutineScope by MainScope()
```

CoroutineScope.

“Parent” of a coroutine.

- It is not recommended to override CoroutineScope
- Instead, use inheritance by delegation from **MainScope()** and **CoroutineScope()** factory functions

```
import kotlinx.coroutines.*

fun main() {
    GlobalScope.launch {
        delay(1000L)
        println("World!")
    }
    println("Hello,")
    Thread.sleep(2000L)
}
```

Concepts.

Essentials every kotlinx.coroutines client must know.

- ✓ CoroutineContext
- ✓ CoroutineDispatcher
- ✓ CoroutineScope

- CoroutineBuilders
- Job
- CompletableJob
- SupervisorJob()
- Deferred

Coroutine Builders.

Bridging blocking and non-blocking worlds.

- Main idea: This code does not compile

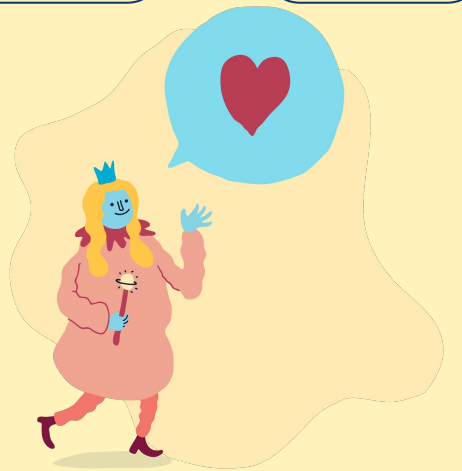
```
suspend fun getRentalCars(): List<Car> = ...

override fun onCreate(savedInstanceState: Bundle) {
  super.onCreate(savedInstanceState)

  getRentalCars() // Compilation error
}
```

```
fun main() {
  getRentalCars() // Compilation error
}
```

Suspend function 'getRentalCars' should be called only from a coroutine or another suspend function

Blocking World

Non-Blocking World

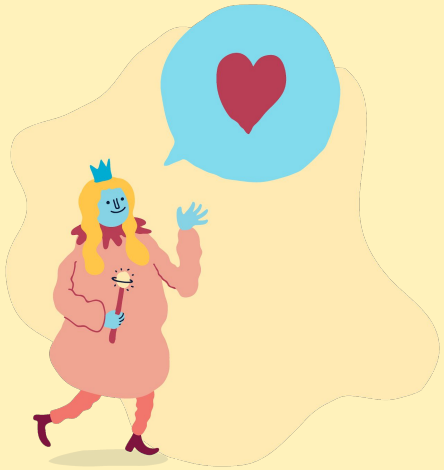


Coroutine Builders.

Bridging blocking and non-blocking worlds.

- launch
- runBlocking
- runBlockingTest (kotlinx.coroutines-test library)

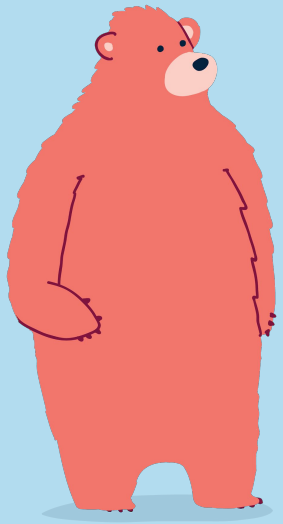
- Special cases of coroutine builders:
  - async
  - withContext



Job.

Conceptually, a background Job

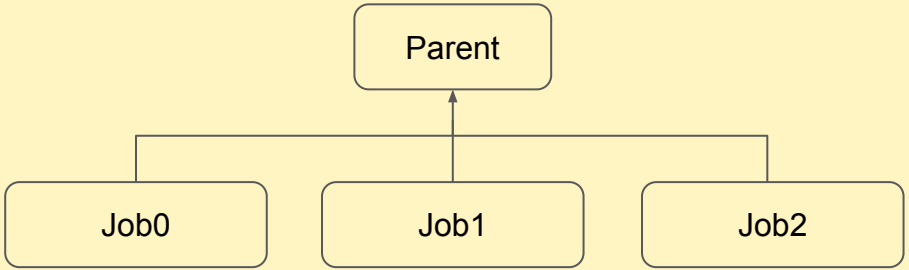
- “Cancelable thing with a lifecycle that culminates in its completion”
- Represents the **execution** of a coroutine
- It is an **abstraction** (interface)
- Jobs can be arranged into parent-child hierarchies
- Created using **launch** coroutine builder or **Job()** factory function
- Conceptually, the execution of a Job does not produce a result

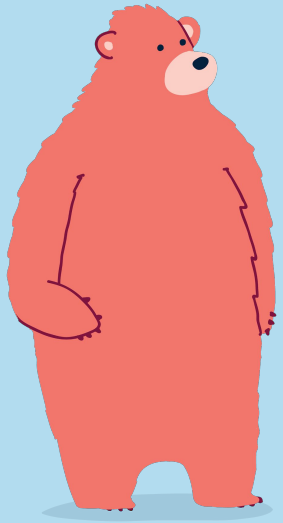


Job.

Conceptually, a background Job

- By default, failure of a child **Job** causes **cancelation of parent** and **all child Jobs**
- This can be customized using **SupervisorJob()**

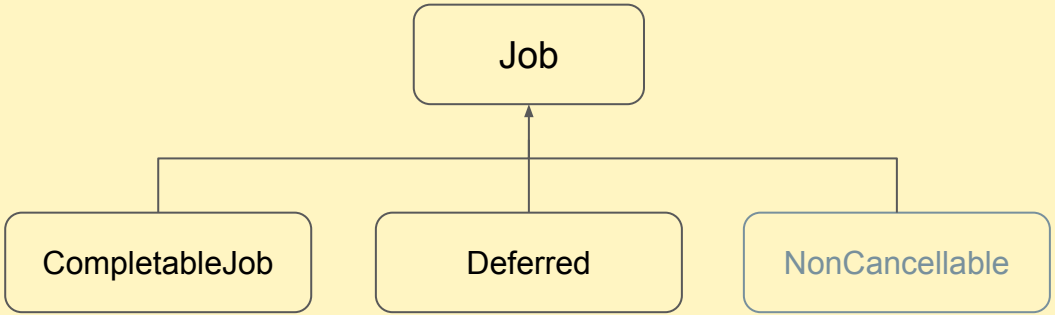


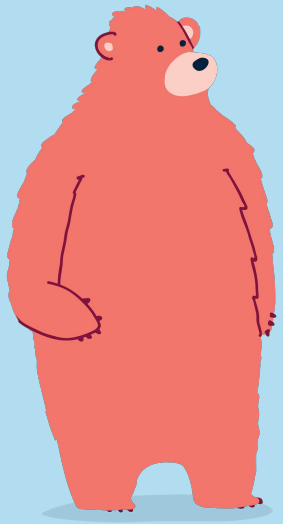


CompletableJob.

Default implementor class for Job.

- A job that can be completed using **complete()** function
- It is returned by **Job()** and **SupervisorJob()** constructor functions.
- For Jobs that produce a result, see **Deferred**



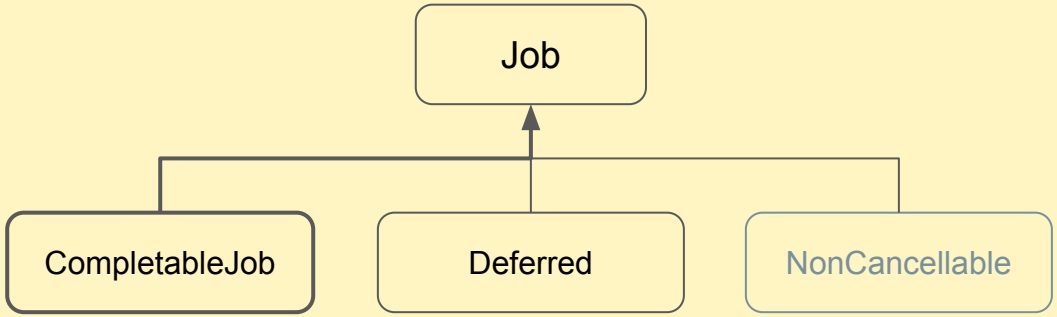


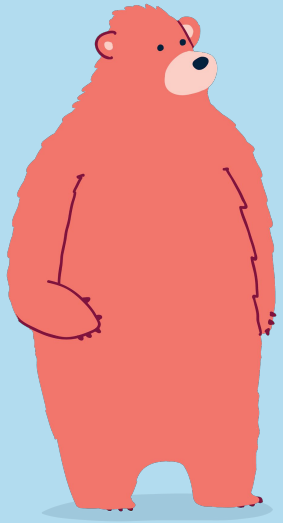
SupervisorJob.

Function returning a “special” CompletableJob.

- Children of a supervisor job can fail independently of each other
  - *“Cancelation of child Job -Parent and other Jobs are not affected”*

```
fun SupervisorJob(parent: Job? = null): CompletableJob
```





Job.

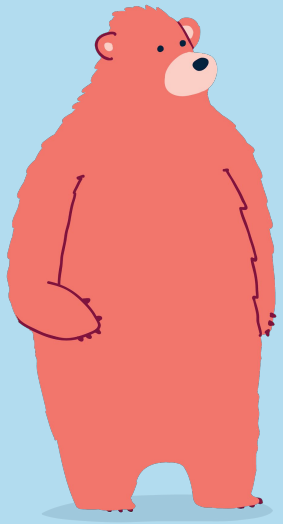
Conceptually, a background Job

- By default, a Job is started on the closing bracket

```
val job = launch(Dispatchers.IO) {
    getRentalCars()
}
```

- It can be created and not launched by using **CoroutineStart.LAZY**

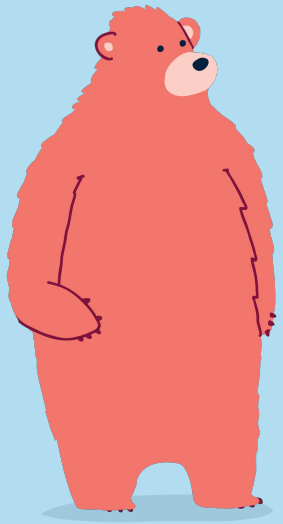
```
val job = launch(start = CoroutineStart.LAZY) {
    getRentalCars()
}

job.start()
```



Deferred.

Non-blocking cancellable future

- It is a Job that returns a result
- Created with the **async** coroutine builder or via the constructor of **CompletableDeferred** class
- The result can be retrieved by **await()** method
- **await()** throws an exception if the Deferred had failed
- Can also be started passing **start = CoroutineStart.LAZY**
- It enables one of the most interesting usages of kotlinx.coroutines

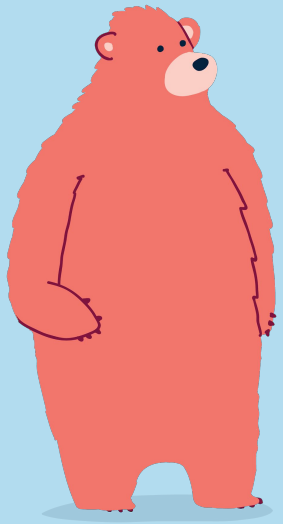


Deferred.

Non-blocking cancellable future

- Example code

```
launch {
  val cars: Deferred = async { getCars() }     // List<Car>
  val users: Deferred = async { getUsers() }   // List<User>

  renderCars(cars.await())
  renderUsers(users.await())
}
```

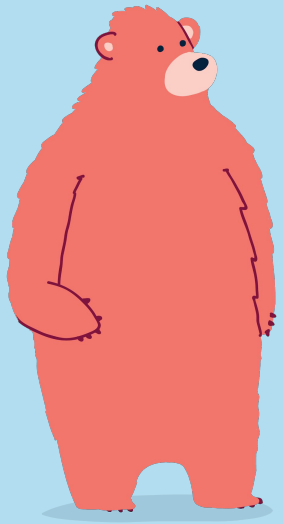


Deferred.

Non-blocking cancellable future

- Example code

```
launch {
  val cars: Deferred = async { getCars() }     // List<Car>
  val users: Deferred = async { getUsers() }   // List<User>

  print("""
    Found a total of ${cars.await().size} cars
    Uploaded by ${users.await().size} users
  """)
}
```



Concepts.

Essentials every kotlinx.coroutines client must know.

- ✓ CoroutineContext
- ✓ CoroutineDispatcher
- ✓ CoroutineScope

- ✓ CoroutineBuilders
- ✓ Job
- ✓ CompletableJob
- ✓ SupervisorJob()
- ✓ Deferred

Examples

Different ways of using kotlinx.coroutines

```
fun main() {
  myScope.launch(Dispatchers.IO) {
    val cars: getCars() // Suspend function

    renderCars(cars)
  }
}
```

```
override fun onCreate(savedInstanceState: Bundle) {
  launch {
    val cars: Deferred = async { getCars() }     // List<Car>
    val users: Deferred = async { getUsers() }   // List<User>

    val totalEntities = cars.await() + users.await()
  }
}
```

Testing with Coroutines.

One common problem

```
@Test
fun `should request a list of cars on start`() {
  givenThereAreSomeCars()

  presenter.start() // Suspend function, executes coroutines

  verify(apiClient).getCars()
}
```

Test execution

Assertion (test end)

Coroutine execution

Test Fails!

Testing with Coroutines.

One common problem

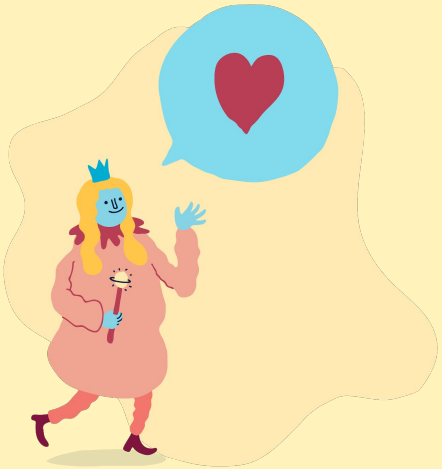
```
testImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-test:1.3.2"
```

```
val testCoroutineDispatcher = TestCoroutineDispatcher()

@Before fun setUp() { Dispatchers.setMain(testCoroutineDispatcher)}
@After fun tearDown() { Dispatchers.resetMain() }
```

```
  @Test
  fun `should request a list of cars on start`() = runBlockingTest {
    givenThereAreSomeCars()

    presenter.start() // Suspend function, executes coroutines

    verify(apiClient).getCars()
  }
```

Credits

- Introduction to Coroutines - Roman Elizarov - Link
- Deep dive into coroutines on JVM - Roman Elizarov - Link
- Understand coroutines on Android - Google - Link
- Coroutines Webinar - Antonio Leiva - Link
- Beyond async/await - Bolot Kerimbaev - Link
- “Structured Concurrency” - Manuel Vicente Vivo - Link
- Coroutines official Guide - JetBrains - Link



GoMore

Thanks!

Q+A time! Any questions?

Victor Olmo Gallegos Hernández
Android Developer at GoMore

@voghDev

voghDev