

Parallel, aber strukturiert

Koroutinen in Kotlin

Werner Eberling

E-Mail: werner.eberling@mathema.de

Twitter: [@Wer_Eb](#)



Der Sprecher



Werner Eberling

Principal Consultant / Autor

Email: werner.eberling@mathema.de

Twitter: [@Wer_Eb](https://twitter.com/Wer_Eb)





Was haben wir vor?

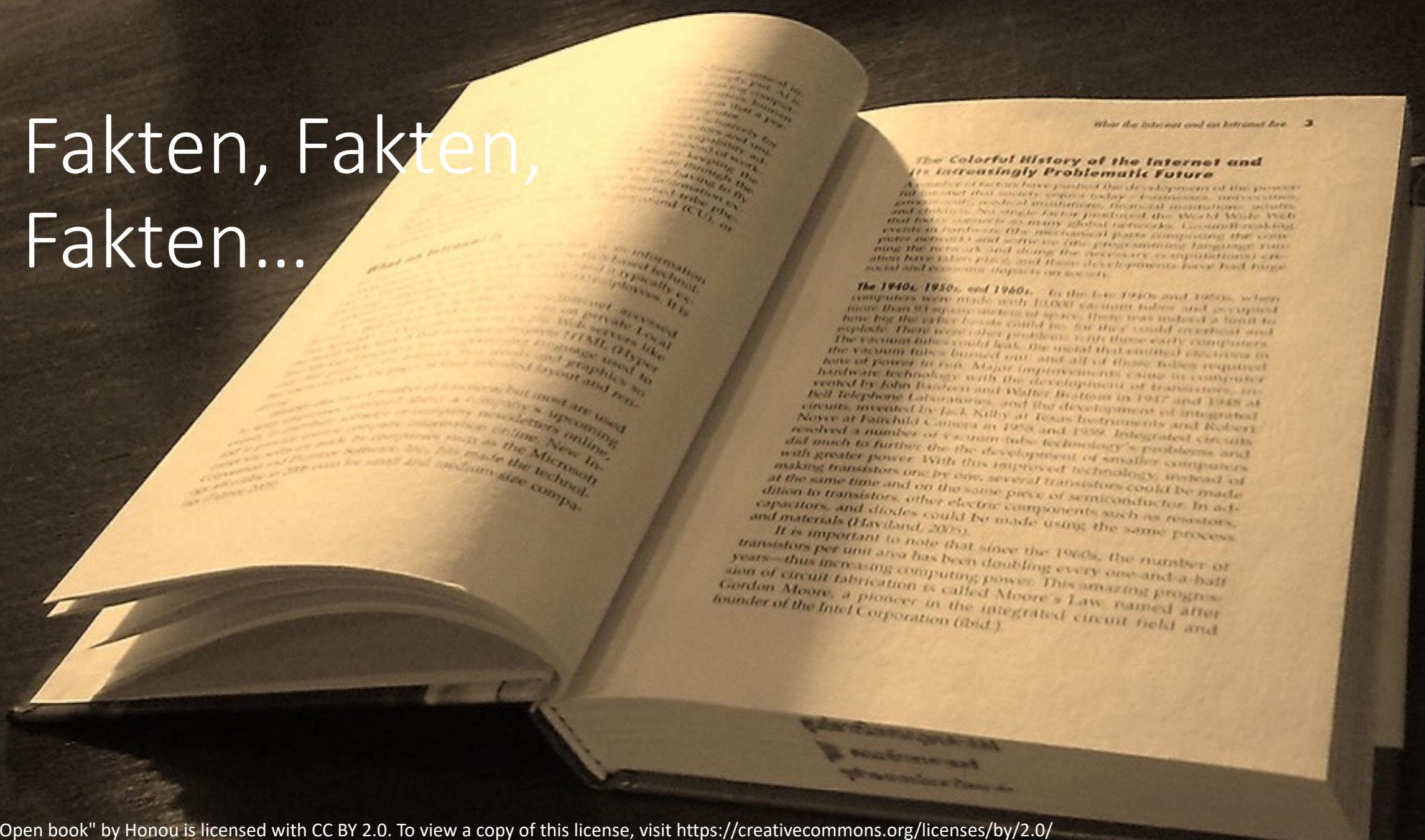
- Grundlagen
- Suspending Functions
- Scopes & Contexts
- Jobs & Cancelation
- Async/await
- Asynchronous Flows & Channels
- Ausblick: Weiterführende Themen



Und vor allem

•

Fakten, Fakten, Fakten...





- Soll eine moderne Alternative zu Java sein
- Syntax zu Java nicht kompatibel
- Interoperabilität sehr wichtig



Was ist Kotlin?

- Statisch typisiert
- Unterstützt sowohl objektorientierte, als auch funktionale Programmierparadigmen
- Läuft in der JVM
- Kann nach JavaScript transpiliert werden
- Unter Android bevorzugte Sprache
- Kann LLVM nutzen (Kotlin/Native)



- IntelliJ
- Android Studio
- Eclipse
- Kommandozeile mit Standalone-Compiler
- Online (play.kotlinlang.org)



Der Klassiker (in Java)

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hallo KKON!");  
    }  
  
}
```

- Ganz schön viel Overhead für ein kleines „Hello World!“, oder?



Und jetzt in Kotlin...

```
fun main() = println("Hallo KKON!")
```

- Beliebige Datei (CamelCase) mit Endung .kt
- main ist Toplevel-Funktion
- Keine Strichpunkte nötig ;)



Beispiele und Materialien

git clone

<https://github.com/wern/kotlin-coroutines-ws-kkon-2022>

Alternativ per Download:

<https://tinyurl.com/ws-kotlin-kkon22>

Passwort: KCR-KKON2022



Nebenläufigkeit mit Koroutinen



Nebenläufigkeit in Java und in Kotlin

```
public class BlockingDemo {  
  
    public static void main(String[] args) {  
        Thread t = new Thread(() -> {  
            try {  
                Thread.sleep(1000);  
                System.out.println("KKON!");  
            } catch (InterruptedException e) {}  
        });  
        t.start();  
        System.out.print("Willkommen zur ");  
        try {  
            t.join();  
        } catch (InterruptedException e) {}  
    }  
}
```

Java

```
fun main() = runBlocking {  
    launch {  
        delay(1000L)  
        println("KKON!")  
    }  
    println("Willkommen zur ")  
}
```

Kotlin

Benötigt
org.jetbrains.kotlinx:kotlinx-coroutines-core:1.6.21



Koroutinen in wenigen Worten

```
fun main() = runBlocking {
    launch {
        delay(1000L)
        println("KKON!")
    }
    println("Willkommen zur")
}
```

- Koroutinen können unterbrochen und wieder fortgesetzt werden
- Kotlin nutzt für die Ausführung „irgendwie“ Threads



Koroutinen – Suspending Functions

```
fun main() = runBlocking {
    launch {
        delayedKKON()
    }
    println("Willkommen zur")
}

suspend fun delayedKKON() {
    delay(1000L)
    println("KKON!")
}
```

- *Suspending Functions* können nur in Koroutinen verwendet werden
 - Kapseln Verhalten wie „normale“ Funktionen
 - Können den Prozessor aufgeben
 - Können weitere *Suspending Functions* rufen (z.B. `delay()`)

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars" style="font-size:18px;"></i></div>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#band">HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#tour">BAND</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#contact">TOUR</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" title="More"><i class="fa fa-caret-down" style="font-size:18px;"></i></button>
    <div class="w3-dropdown-content w3-bar-item w3-padding-large" style="width:150px; margin-top:-10px; background-color:white; border:1px solid #ccc; padding:5px; font-size:14px; border-radius:5px; z-index:1000; position: absolute; left: -10px; top: 0px; transition: all 0.3s ease-in-out; opacity: 0; visibility: hidden; ">
      <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">>MORE</a>
    </div>
  </div>
</div>
```

A scenic view of a rural landscape. In the foreground, there's a dark wooden fence line. Beyond it are several green pastures separated by more wooden fences. A large, two-story house with a dark roof and white trim sits atop a hill in the background, surrounded by trees. The sky is clear and blue.

Strukturierte
Nebenläufigkeit



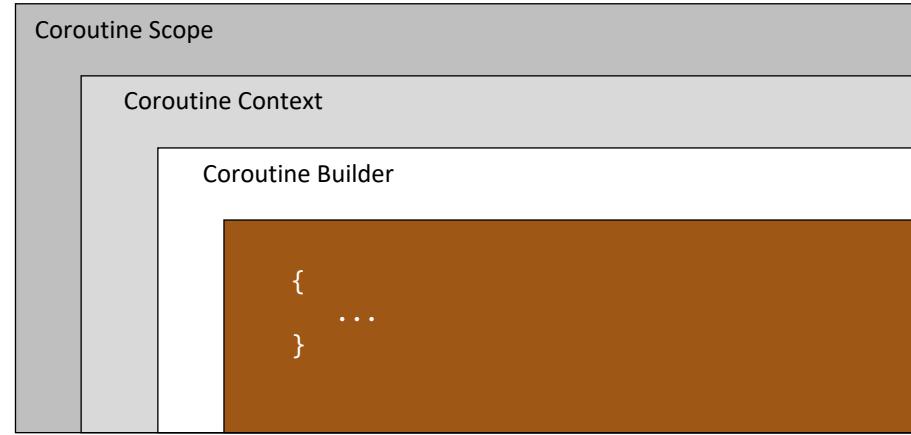
Strukturierte Nebenläufigkeit - Scopes

```
fun main() = runBlocking {
    launch {
        delay(1000L)
        println("KKON!")
    }
    println("Willkommen zur")
}
```

- Koroutinen laufen immer innerhalb eines Scopes
Start u.a. mit `runBlocking{...}` oder `coroutineScope{...}`
- `runBlocking` als Brücke zwischen „normalem“ und „suspendable“ Code
- `coroutineScope` innerhalb von *Suspending Functions*



Strukturierte Nebenläufigkeit



- Scopes
 - Beschränkung der Laufzeit der Koroutinen (nur innerhalb des Scopes)
 - Keine „wild laufenden“ Threads (auch nach Fehlern)
- Contexts
 - Ablaufkontext der Koroutine (Welcher Thread arbeitet die Koroutine ab?)
- Builder
 - Erzeugung konkreter Koroutinen



Koroutinen – Parallele Ausführung

```
fun main() = runBlocking {
    launch {
        delayedKKONwithNewJobs()
    }
    println("Willkommen zur")
}
suspend fun delayedKKONwithNewJobs() = coroutineScope {
    launch {
        delay(2000L)
        println("2022")
    }
    launch {
        delay(1000L)
        println("digital")
    }
    println("KKON")
}
```

- Neuer Scope mit zwei parallelen Koroutinen
Innerer Scope endet erst, wenn beide Koroutinen beendet wurden!



Koroutinen – Was passiert bei Fehlern?

```
fun main() = runBlocking {
    launch {
        delayedKKONwithNewJobs()
    }
    println("Willkommen zur")
}
suspend fun delayedKKONwithNewJobs() = coroutineScope {
    launch {
        delay(2000L)
        println("2022")
    }
    launch {
        delay(1000L)
        throw RuntimeException("Bang!")
    }
    println("KKON")
}
```

- Unbehandelte Fehler führen zum Abbruch des Scopes und aller enthaltenen Koroutinen (=> Strukturierte Nebenläufigkeit)

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars" style="font-size: 1.2em; color: #ccc; margin-right: 10px; font-weight: bold; border-bottom: 1px solid #ccc; padding-bottom: 2px; transition: all 0.3s ease; cursor: pointer; position: relative; z-index: 1; "></i><span style="font-size: 1.2em; color: #ccc; font-weight: bold; position: absolute; right: -10px; top: 0; background-color: #fff; padding: 5px; border-radius: 5px; z-index: 0; transition: all 0.3s ease; ">HOM
    <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">HOME</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">BAND</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">TOUR</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">CONTACT</a>
  </div>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; "><i class="fa fa-caret-down" style="font-size: 1.2em; color: #ccc; margin-right: 10px; font-weight: bold; border-bottom: 1px solid #ccc; padding-bottom: 2px; transition: all 0.3s ease; cursor: pointer; position: relative; z-index: 1; "></i><span style="font-size: 1.2em; color: #ccc; font-weight: bold; position: absolute; right: -10px; top: 0; background-color: #fff; padding: 5px; border-radius: 5px; z-index: 0; transition: all 0.3s ease; ">MORE</span></button>
    <div class="w3-dropdown-content w3-bar-block w3-padding-large" style="background-color: #fff; border: 1px solid #ccc; width: 200px; position: absolute; left: 0; top: 100%; z-index: 1; transition: all 0.3s ease; ">
      <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">About</a>
      <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">Services</a>
      <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">Portfolio</a>
      <a href="#" class="w3-bar-item w3-button w3-padding-large" style="color: #ccc; font-size: 1.2em; font-weight: bold; padding: 10px; border-radius: 5px; transition: all 0.3s ease; ">Contact</a>
    </div>
  </div>
</div>
```



Dispatcher



Dispatcher – Wer macht eigentlich die Arbeit?

```
fun main() = runBlocking {
    launch {
        delayedKKON()
    }
    printlnWithThreadInfo("Willkommen zur")
}

suspend fun delayedKKON() {
    delay(1000L)
    printlnWithThreadInfo("KKON!")
}
```

- *Coroutine Dispatcher* definieren in welchem Thread-Kontext eine Koroutine abläuft
Wird normalerweise vom umgebenen Scope übernommen (hier: Main-Thread)



Dispatcher – Wer macht eigentlich die Arbeit?

```
fun main() = runBlocking {
    launch(Dispatchers.Default){
        delayedKKON()
    }
    printlnWithThreadInfo("Willkommen zur")
}

suspend fun delayedKKON() {
    delay(1000L)
    printlnWithThreadInfo("KKON!")
}
```

- Beim Bau der Koroutine kann der gewünschte Dispatcher explizit gesetzt werden



Dispatcher – Mögliche Optionen

- Das Dispatchers Objekt definiert die standardmäßig verfügbaren Dispatcher

Default: Nutzung des Default Thread Pools

IO: Spezieller Dispatcher für Aufgaben mit Blocking IO

Main: Ablauf im UI-Main Thread (nur mit entsprechender Library Unterstützung)

Unconfined: Keine Bindung an einen konkreten Thread/Thread-Pool

- Zusätzlich kann auch explizit neuer Context mit eigenem Thread gestartet werden

```
launch(newSingleThreadContext("MyOwnThread")) { ... }
```

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars" style="font-size:18px;"></i></div>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#band">HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#tour">BAND</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#contact">TOUR</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" title="More"><i class="fa fa-caret-down" style="font-size:18px;"></i></button>
    <div class="w3-dropdown-content w3-bar-item w3-padding-large" style="width:150px; margin-left:20px; border:1px solid #ccc; background-color:white; padding:10px; font-size:12px; font-family: inherit; font-weight:bold; color:#000; position: absolute; left: -10px; top: 0; z-index: 1; border-radius: 5px; transition: all 0.3s ease-in-out; opacity: 0; visibility: hidden; transform: rotateX(0deg);>
      <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">>MORE</a>
    </div>
  </div>
</div>
```

Kontrolle mit Jobs



"Close-up of a man holding remote control of robotic vacuum cleaner to start cleaning" by Marco Verch is licensed with CC BY 2.0.

To view a copy of this license, visit <https://creativecommons.org/licenses/by/2.0/>

Link to original photography: <https://foto.wuestenigel.com/close-up-of-a-man-holding-remote-control-of-robotic-vacuum-cleaner-to-start-cleaning/>



Jobs – Wenn es „expliziter“ sein soll

```
fun main() = runBlocking {
    val job = launch {
        delayedKKON()
    }
    println("Willkommen zur")
    job.join()
    println("2022")
}
suspend fun delayedKKON() {
    delay(1000L)
    println("KKON")
}
```

- `launch` liefert bei der Erzeugung der Koroutine ein Job Objekt zurück
 - Ermöglicht Abfrage des Status der Koroutine
 - Kann zum expliziten Warten auf die Beendigung der Koroutine genutzt werden



Cancelation – den Job wieder abbrechen

```
fun main() = runBlocking {  
    val job = launch {  
        delayedKKON()  
    }  
    println("Willkommen zum")  
    delay(10L)  
    job.cancel()  
    println("Heute nicht ;)")  
}
```

```
suspend fun delayedKKON() {  
    delay(1000L)  
    println("KKON")  
}
```

- Das Job Objekt ermöglicht auch den Abbruch der Koroutine
 - Bricht auch alle „Subjobs“ ab
 - Benötigt die „Kooperation“ der Koroutine
 - Alle suspending Library Funktionen sind „cancelable“
 - Eigener Code muss den Context Status abfragen (`CoroutineScope.isActive`)



Cancelation – auf den Abbruch reagieren

```
fun main() = runBlocking {
    val job = launch { //...
    //...
    job.cancel()
}
suspend fun delayedKKON() {
    try {
        delay(1000L)
        println("KKON")
    }catch(e : CancellationException){
        println("Abbruch KKON :(")
    }
}
```

- Library Funktionen werfen CancellationException im Falle eines Abbruchs

Gilt bei Nicht-Abfangen als „normales“ Verhalten der Koroutine



Timeouts – Abbruch auch ganz automatisch

```
fun main() = runBlocking {  
    withTimeout(100L) {  
        delayedKKON()  
    }  
}  
suspend fun delayedKKON() {  
    try {  
        delay(1000L)  
        println("KKON")  
    }catch(e : CancellationException){  
        println("Abbruch KKON :(")  
    }  
}
```

- Beim Erzeugen via `withTimeout()` kann einer Koroutine eine max. Laufzeit angegeben werden, nach der diese abgebrochen wird

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars"></i>
    <a href="#" class="w3-bar-item w3-button w3-padding-large">HOME</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large">BAND</a>
    <a href="#tour" class="w3-bar-item w3-button w3-padding-large">TOUR</a>
    <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
  </div>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" title="More">MORE<i class="fa fa-caret-down"></i></button>
    <div class="w3-dropdown-content w3-bar-block w3-padding-large">
      <a href="#" class="w3-bar-item w3-button w3-padding-large">ABOUT</a>
      <a href="#" class="w3-bar-item w3-button w3-padding-large">MEMBER</a>
      <a href="#" class="w3-bar-item w3-button w3-padding-large">CONTACT</a>
    </div>
  </div>
</div>
```

A photograph showing a large-scale conveyor belt system in a warehouse or distribution center. Numerous cardboard boxes are being transported on the belts. The boxes are labeled with various codes, such as "1120 182 006 BT" and "1120 182 006 88".

Ergebnisse zurückliefern



Rückgabewerte mit `async` und `await`

```
fun main() = runBlocking {
    val audience = async { countAudience() }
    val presenter = async { countPresenter() }
    println("${audience.await()} Zuhörer und ${presenter.await()} Sprecher anwesend.")
}
suspend fun countAudience(): Int {
    delay(1000L)
    return 14
}
suspend fun countPresenter(): Int {
    delay(1000L)
    return 1
}
```

- `async` erzeugt eine Koroutine, ähnlich wie `launch`
Statt eines Jobs wird der Rückgabewert als Deferred zurückgegeben
- Mittels `await` kann auf das Beenden der Koroutine und den Rückgabewerte gewartet werden



Mehrere Werte zurückgeben

```
fun main() = runBlocking {
    println("Teilnehmer:")
    val participants = async { participantSuspendList() }
    launch {
        // Start paralleler Verarbeitungen
    }
    participants.await().forEach { println(it) }
}
suspend fun participantSuspendList(): List<String> {
    participantDatabase.forEach {
        delay(100) // Komplexe Verarbeitung der Eintraege
    }
    return participantDatabase
}
```

- Asynchrone Aufrufe können auch mehrere Ergebnisse zurückliefern
- Mittels await wird aber auf das Gesamtergebnis gewartet
 - Reduziert evtl. die Parallelität der Ergebnisverarbeitung
 - Werte können nicht „so früh wie möglich“ verarbeitet werden



Werte parallel Erzeugen mit asynchronen Flows

```
fun main() = runBlocking {
    println("Teilnehmer:")
    launch {
        // Start paralleler Verarbeitungen
    }
    participantFlow().collect(){ println(it) }
}

fun participantFlow() = flow {
    participantDatabase.forEach {
        delay(1000) // aufwendige Datenverarbeitung
        emit(it) // Ausgabe des naechsten Wertes
    }
}
```

- `flow` erzeugt eine asynchronen Flow

Der *Flow Builder* ist **keine Suspending Function**

Ein *Flow* kann *Suspending Functions* aufrufen

Werte werden **einzeln** durch `emit()` zurückgegeben und mit `collect()` abgerufen

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars" style="font-size:18px;"></i></div>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#band">HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#tour">BAND</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#contact">TOUR</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" title="More"><i class="fa fa-caret-down" style="font-size:18px;"></i></button>
    <div class="w3-dropdown-content w3-bar-item w3-padding-large" style="width:150px; margin-left:20px; border:1px solid #ccc; background-color:white; padding:10px; font-size:14px; border-radius:5px; position:absolute; left:0; top:0; z-index:1000; display:none;">
      <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">>MORE</a>
    </div>
  </div>
</div>
```



Koroutinen verbinden



Channels – non blocking Kommunikationskanäle

```
val chars = "Hello KKON digital 2022!".toCharArray()

fun main() = runBlocking {
    val channel = Channel<Char>()
    launch {
        chars.forEach {
            delay(400) // aufwendige Datenverarbeitung
            channel.send(it) // Ausgabe des naechsten Wertes
        }
    }
    repeat(chars.size) { print(channel.receive()) }
}
```

- Um Daten aus einer Koroutinen zu liefern können *Channels* verwendet werden
- *Channels* arbeiten ähnlich wie *BlockingQueues*, nur **suspendable**
Können auch Daten Puffern => `Channel<Char>(10)`



Channels – Mehr gibt's nicht

```
val chars = "Hello KKON digital 2022!".toCharArray()

fun main() = runBlocking {
    val channel = Channel<Char>()
    launch {
        chars.forEach {
            delay(400) // aufwendige Datenverarbeitung
            channel.send(it) // Ausgabe des naechsten Wertes
        }
        channel.close()
    }
    for(c in channel) { print(c) }
}
```

- Um mitzuteilen, dass keine weiteren Daten vorhanden sind, kann ein *Channel* explizit geschlossen werden.

Der Empfänger wird hierüber durch einen speziellen Token benachrichtigt



Producer – Koroutinen mit impliziter Channel Erzeugung

```
val chars = "Hello KKON digital 2022!".toCharArray()

fun main() = runBlocking {
    val charProducer = produce {
        chars.forEach {
            delay(400)
            send(it)
        }
    }
    charProducer.consumeEach { print(it) }
}
```

- Der *CoroutineBuilder* `produce` erzeugt eine neue Koroutine, die Ihre Rückgabewerte über einen `ReceiveChannel` zurückliefert
Passende *ExtensionFunctions* vereinfachen zusätzlich die Datenverarbeitung



Pipelines – Kommunikationkette von Koroutinen (I)

```
fun CoroutineScope.produceWords(charProducer : ReceiveChannel<Char>) :  
    ReceiveChannel<String> = produce {  
    val wordBuffer = StringBuilder()  
    charProducer.consumeEach {  
        if(it == ' ') {  
            send(wordBuffer.toString())  
            wordBuffer.clear()  
        } else {  
            wordBuffer.append(it)  
        }  
    }  
    if(!wordBuffer.isEmpty()) { send(wordBuffer.toString())}  
}
```

- Bei der Erzeugung von *Producer* Koroutinen können vorhandene *Channels* als Parameter übergeben werden, ...



Pipelines – Kommunikationskette von Koroutinen (II)

```
val chars = "Hello KKON digital 2022!".toCharArray()

fun main() = runBlocking {
    val charProducer = produce {
        chars.forEach {
            delay(400)
            send(it)
        }
    }
    val wordProducer = produceWords(charProducer)
    wordProducer.consumeEach { println(it) }
}
```

- ... um eine Verarbeitungskette von Koroutinen (*Pipeline*) zu erzeugen



Fan Out – Daten mit mehreren Koroutinen parallel verarbeiten

```
val chars = "Hello KKON digital 2022!".toCharArray()

fun main() = runBlocking {
    val charProducer = produce {
        chars.forEach {
            delay(400)
            send(it)
        }
    }
    launchWordConsumer(1, wordProducer)
    launchWordConsumer(2, wordProducer)
}
```

- Datenverarbeitung erfolgt „fair“ (nach FIFO Semantik)
- Auch Fan-In möglich

```
fun CoroutineScope.launchWordConsumer(idx: Int,
    wordProducer: ReceiveChannel<String>) =
    launch {
        for (word in wordProducer) {
            println("${word} (${idx}. Consumer)")
        }
    }
```

Hands-on

```
<!-- Navbar -->
<div class="w3-top">
  <div class="w3-bar w3-black w3-hide-large w3-right" href="javascript:void(0)" title="Toggle Navigation Menu"><i class="fa fa-bars" style="font-size:18px;"></i></div>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#band">HOME</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#tour">BAND</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#contact">TOUR</a>
  <a href="#" class="w3-bar-item w3-button w3-padding-large w3-hide-small">CONTACT</a>
  <div class="w3-dropdown-hover w3-hide-small">
    <button class="w3-padding-large w3-button" title="More"><i class="fa fa-caret-down" style="font-size:18px;"></i></button>
    <div class="w3-dropdown-content w3-bar-item w3-padding-large" style="width:150px; margin-left:20px; border:1px solid #ccc; background-color:white; padding:10px; font-size:14px; border-radius:5px; position:absolute; left:0; top:0; z-index:1000; display:none;">
      <a href="#" class="w3-bar-item w3-button w3-padding-large" href="#">>MORE</a>
    </div>
  </div>
</div>
```



Gemeinsamer Zustand

<https://kotlinlang.org/docs/shared-mutable-state-and-concurrency.html>

Wie geht es von
hier weiter?





Weiterführende Themen

- Start Modi
- Exception Handling
- Shared State
- ...



Kotlin
<https://kotlinlang.org/docs/coroutines-guide.html>



Referenzen

- Beispiele & Folien
 - [https://github.com/wern/
kotlin-coroutines-ws-kkon-2022](https://github.com/wern/kotlin-coroutines-ws-kkon-2022)
- Kotlin Dokumentation
 - <https://kotlinlang.org/docs/home.html>
- Kotlin Playground
 - <https://play.kotlinlang.org>



"Cambridge Public Library #4" by brokentrinkets is licensed with CC BY 2.0.
To view a copy of this license, visit <https://creativecommons.org/licenses/by/2.0/>



Vielen Dank! Fragen?

Werner Eberling

E-Mail: werner.eberling@mathema.de
Twitter: [@Wer_Eb](https://twitter.com/@Wer_Eb)

Beispiele:
<https://github.com/wern/kotlin-coroutines-ws-kkon-2022>

www.mathema.de



Beispiele?
Scan me ;)