2023-10-14 L04 Android

**Coding bis 10:40**

==============

**Concurrency with Kotlin Coroutines**

Bei Bedarf:

* Demo-Code im Repo: <https://git-iit.fh-joanneum.at/omd/code-snippets/-/tree/master/kotlin/160_concurrency?ref_type=heads>
* Erklärungen im "study-material...md " : <https://git-iit.fh-joanneum.at/omd/droid-devel/-/blob/master/Part-1-Kotlin/study-material--kotlin.md?ref_type=heads>
* Online: z.B. <https://kotlinlang.org/docs/coroutines-basics.html>
* Ihr Vorschlag, gute Quellen/Referenzen/URLs: ... <hier einfügen, Thx>

Done - Code Discussion

**Source Code - Very Well Done**

string interpolation

* println("it was pressed ${button.amountPressed} times")

immutable

* val

functions

* named params: val aspirin2 = Medicament( name= "Aspirin", dose=0.5, times = "evening")

lists

* private val continents = mutableListOf<Continent>()
* continents.find { it.name == name }
* beers.groupBy { it.brand }
* beers.map { it.brand }.toSet()
* val groupedBeers: Map<String, List<Beer>> = beers.getBeersGroupedByBrand()
* for((brand, beerBrand) in groupedBeers)

op-overloading

* operator fun plusAssign(meal: Meal){

singleton

* object UserIdSeq {
* private var currId = 0L
* fun getNext() = ++currId
* }
* data class User (
* val id: Long = UserIdSeq.getNext(),

**Source Code - Quick Ideas**

* avoid: add / get: fun addContinent(continent: Continent){  ... fun addBeer(beer : Beer){
* => ideas: ....
* Create demo data
* => Initialise with list (= multiple instances, created on the go)
* functions what do the values mean val africa = Continent("Africa", 2345.343, 2345, 34)
* => ...
* type inference
* var calories: Int = 0
* val today : Date = Date()

**Source Code - To Disc**uss

* Readability: not-ok or ok ?
* val roomDescription = "\nThis room is called $name and has a $objects in it.\n".also(::println)
* @Volatile
* private var instance: RoomData? = null
* val room2 = Room("Kitchen", 12, 8, "Mixer")
* Date(System.currentTimeMillis() - 60\*60\*6000)
* val user0 = User(
* minWordCount = entries.minOfOrNull { it.content.text.length } ?: 0
* println(diary.getDiaryStats())

- - - Backup the full agenda from the markdown file ----

\* 14.10.2023 (Samstag)

        \* 08:45 - 11:00        IMS 2023 / UE        MoSWDev        Johannes Feiner        X/ONL/Teams

\* Moodle: <<https://elearning.fh-joanneum.at/course/view.php?id=3153>>

\* Almaty: <<https://almaty.fh-joanneum.at/stundenplan/index.php?submit=Suche&q=ims23+lv%3AMoSWDev>>

\* GitLab:

        \* Droid-Devel: <<https://git-iit.fh-joanneum.at/omd/droid-devel>>

        \* Kotlin-Snippets: <<https://git-iit.fh-joanneum.at/omd/code-snippets/-/tree/master/kotlin>>

# Agenda - Concurrency

\* Review:

        \* L01 Intro = Orga

        \* L02 Android Tools & Basic Architecture

        \* L03 Kotlin

                \* Model data + service

                \* Coding (in scratch files)

\* Idiomatic Kotlin

\* Code Discussion (from L03)

\* Kotlin Concurrency

        \* Kotlin \*\*Coroutines\*\*

\* Outlook

        \* Check out: MAD

- - -

### Idiomatic Kotlin

\* Idiomatic / Effective Kotlin Training (essence of the book)

        \* (A) Good Code:

                \* 1 Safety:

                        \* a Limit mutability

                        \* b Close resources with use

                \* 2 Readablility:

                        \* a Properties represent state not behavior

        \* (B) Code Design:

                \* 3 Reusability:

                        \* a Use property delegation

                \* 4 Abstraction Design

                        \* a Specify API stability

                \* 5 Object Creation

                        \* a Primary construcor with named optional arguments

                \* 6 Class Design

                        \* a Composition over inheritance

                        \* b Extract non-essential parts into extensions

        \* (C) Efficiency

                \* 7 Make it cheap

                        \* a Avoid object creation

                \* 8 Efficient collection processing

                        \* a Use mutable collections for...

### Review

\* Review -- Discuss Kotlin Features:

        \* From Java to Kotlin <= \*\*idiomatic kotlin\*\*

                \*\*...Idiomatic Kotlin means to let go old Java habits...\*\*

                \* (-) mutable data (variables, data-structures)

                \* (+) `val` for immutable data

                \* - - - - - - -

                \* (-) Check for null

                \* (+) Null Safety: if-not-null `?.`, Smart casts, if-not-null else elvis operator `?:`

                \* - - - - - - -

                \* (-) No \*getters\* and \*setters\*

                \* (+) Use \*properties\* (also: \*lazy properties\*)

                \* - - - - - - -

                \* (-) Classes with boilerplate code for: `hashCode`, `equals`, `toString`, `copy`, ..

                \* (+) Shortcuts: `data classes` (avoid boilerplate code) with `var`/`val` primary constructor`

                \* - - - - - - -

                \* (-) No boilerplate (multiple lines of) code for \*singletons\*

                \* (+) Use keyword `object`

                \* - - - - - - -

                \* (-) Avoid builder pattern

                \* (+) use Scope functions: call multiple methods `with`, configure properties with `apply`, ... `also`, `let`, `run`

                \* - - - - - - -

                \* (-) inheritance

                \* (+) extension functions

                \* - - - - - - -

                \* (-) only few built-in overloaded operators: +

                \* (+) OOP advanced: operator overloading

        \* <<https://kotlinlang.org/docs/idioms.html>>

### Feedback to your code:

```kotlin

// TODO: ...

// (1)

// (2)

```

### Summary of suggestions:

\* Java Idiom or Pattern Idiomatic => Solution in Kotlin

        \* Optional => \*\*Nullable\*\* Types

        \* Getter, Setter, Backing Field => \*\*Properties\*\*

        \* Static Utility Class => Top-Level (\*\*extension\*\*) functions

        \* Immutability => \*\*data\*\* class with \*\*immutable\*\* properties, copy()

        \* Value Objects => inline class with \*\*immutable\*\* properties

        \* Fluent Setter (Wither) => \*\*Named arguments\*\*, and default arguments, apply()

        \* Method Chaining => \*\*Default arguments\*\*

        \* Singleton => \*\*object\*\*

        \* Delegation Delegated => \*\*properties by\*\*

        \* Lazy Initialization (thread-safe) => Delegated properties \*\*by: lazy\*\*()

        \* Observer => Delegated properties \*\*by: Delegates.observable()\*\*

- - -

### Coroutines

\* Ways of concurrency

        \* ???

        \* In other programming languages: ???

\* Coroutines:

        \* <<https://kotlinlang.org/docs/reference/coroutines/basics.html>>.

        \* function(s) which be are suspendable: `suspend fun myf(..){ ... }`.

        \* create coroutine scopes

                \* e.g. using `fun main () = runBlocking {...}`                 \* or your own coroutineScope

        \* and inside this scope use coroutine builder launch to call the prepared suspendable function(s) inside

                \* (e.g. `myjob = launch { ... myf() ... }`.

                \* The coroutine builder returns a job. One can wait for completion of jobs with myjob.join().

### Coding

bis 10:40... dann diskussion

\* Kotlin-Week

        \* <<https://git-iit.fh-joanneum.at/omd/droid-devel/-/tree/master/Part-1-Kotlin/kotlinweek>>

### Outlook L05

\* Outlook

        \* Guide to \*\*app architecture\*\*

                <<https://developer.android.com/topic/architecture>>

        \* Details:

                \* Check out: MAD = suggestions by Google: MAD Modern Android Development

                        \* Google Developer Video <<https://www.youtube.com/c/AndroidDevelopers/videos>>

                \* JetPack <<https://developer.android.com/jetpack/getting-started>>

                        \* \*\*WorkManager\*\* for your background scheduling needs.

                        \* \*\*Room\*\* for data storage persistence.

                        \* \*\*Navigation\*\* to manage your application navigation flow.

- - -

# NOT DONE

### Coding in teams of two (and present demo implementation)

...todo add names of students to topic in the Etherpad...

\* Implement: ... Tasks for all students:

        \* Team of two (or three). For example:

\* (1) by ...

        \* How to serialise data (use data objects, lazy properties, default arguments,... evtl. @Parcelize)

        \* Optionally, check out: <<https://github.com/Kotlin/kotlinx.serialization/blob/master/docs/serialization-guide.md>>

\* (2) by ...

        \* How to sort, filter, map a list / collection of GPS data (use "High Order Functions", lambdas, ....).

You might add such functionality with extension functions to existing types/class, ..

\* (3) by ...

        \* Provide an API for a "closed" image conversion "service" (access modifiers for strict privacy, no inheritance allowed, sealed classes, ...) and more "open" (allow inheritance,...) classes (e.g. with functionality: translate messages).

        \* Optionally, for safety reasons use sealed classes instead of exceptions: <<https://phauer.com/2019/sealed-classes-exceptions-kotlin/>>.

\* (4) by ...

        \* A "global" service (singleton,..), to store GPS location changes over time (optional log current position changes with propery observers, ...).

Return multiple values from a method using destructing, e.g. val (isInRange, distance) = distBetweenLocations(work,home,radius).

\* (5) by ...

        \* load some files from the filesystem or a web service (emply null safety: nullability, exception handling, ...)

You might check out: <<https://kotlinlang.org/docs/reference/exceptions.html>>

\* (6) by ...

        \* Trigger log entries to save timestamp for every change of your data. Traceability. Observable properties using delegation: delegated properties.

Maybe you manage to explain the difference to property delegation (which is used instead of inheritance).

\* (7) by ...

        \* Provide an internal "service" which can handle (return latest three posts/messages, filter by date) multiple types of data you get from a web service. Use generics (with Invariance).

        \* Optional shorten code by using type aliases.

\* (8) by ...

        \* Use structured concurrency with Coroutines to, for example, fetch remote data, create thumbnails, store data to cache/database: <<https://kotlinlang.org/docs/reference/coroutines/basics.html>>.

Prepare function(s) which be are suspendable: suspend fun myf(..){ ... }. Then create coroutine scopes (e.g. using fun main () = runBlocking {...} or your own coroutineScope) and inside this scope use coroutine builder launch to call the prepared suspendable function(s) inside (e.g. myjob = launch { ... myf() ... }. The coroutine builder returns a job. One can wait for completion of jobs with myjob.join().

        \* Optionally, check out code examples at <<https://github.com/Kotlin/kotlinx.coroutines/tree/master/kotlinx-coroutines-core/jvm/test/guide>>.

\* (9) by ...

        \* Store points of interests (POIs, location) and show how to store some of the properties (POI is hotel, pool, beach) as enums.

Show how to use smart casts (as?) to avoid if-type checking.

\* (10) by ...

        \* Provide classes with operator overloading for a more natural use. E.g. calc the diff for POI distances (or time calculations)

Show how to use apply for (grouping) thei initialisation of an POI object.

Show how to use single expression functions to map one type (e.g. POI) to another type (e.g. DTO) fun mapToDTO(entity: POI) = PoiDTO(..) and val dto = mapToDTO(poi).

\* Discuss some of the implementations