bis **20:45** code 1 oder im 2er Team...

Step 1 Model

* single data point
* management (service) of many data points

Step 2 Code

* e.g. data class
* e.g. singleton

....

* 13.10.2023 (Freitag)
* 19:00 - 21:15 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?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>

L03 KotlinAgenda = Overview

* Review
* L01 Intro
* Battle - Teams: <https://elearning.fh-joanneum.at/mod/etherpadlite/view.php?id=55320>
* L02 Android App Basics
* Tools and App-Structure
* Kotlin Feature - Overview (what's available, possibly new)
* Plan for your app (= Model data + service)
* data modelling (e.g. POIs, GPS-Track, Slideshow, Game-Players,...)
* Coding
* e.g. Android Studio Scratch Files, or Scripts, or App, ...
* Implement, Refactor, Try out features, Upload
* Outlook:
* Uploaded Code Discussion
* Coroutines

Details

* Review Android App - Basics
* Tools: pathes, debug, adb, ...
* Architecture:
* basic structure, basic blocks (resources, code,..) \* Activity vs. Fragment **LifeCycle**
* Have we discussed this?: <https://developer.android.com/guide/fragments/lifecycle> in addition to <https://developer.android.com/guide/components/activities/activity-lifecycle>
* App versus **service** vs. **broadcast receiver** vs. **content provider** (as discussen in detail at: <https://developer.android.com/guide/components/fundamentals> )
* App vs. Services (running in the background, sometimes in the foreground), IPC
* start **explicit** (deprecated: implicit) **intents** <https://developer.android.com/guide/components/intents-filters>
* Explain: an app calls bindService()
* broadcast receiver
* Explain: why register for most "implicit broadcasts" is not allowed anymore
* content provider (resolvers and URIs)
* Explain: who calls the methods provided, such as query or insert/update/delete
* Discuss Kotlin Features:
* <https://git-iit.fh-joanneum.at/omd/droid-devel/-/blob/master/Part-1-Kotlin/study-material--kotlin.md>
* From Java to Kotlin <= **idiomatic kotlin**
* **...Idiomatic Kotlin means to let go old Java habits...**
* (-) mutable data (variables, data-structures)
* (+) val for immutable data 1
* (-) Check for null
* (+) Null Safety: if-not-null ?., Smart casts, if-not-null else elvis operator ?: 2
* (-) No *getters* and *setters*
* (+) Use *properties* (also: *lazy properties*) 3
* (-) Classes with boilerplate code for: hashCode, equals, toString, copy, ..
* (+) Shortcuts: data classes (avoid boilerplate code) with var/val primary constructor` 4
* (-) No boilerplate (multiple lines of) code for *singletons*
* (+) Use keyword object 5
* (-) Avoid builder pattern
* (+) use Scope functions: call multiple methods with, configure properties with apply, ... also, let, run 6
* (-) inheritance
* (+) extension functions 7
* (-) only few built-in overloaded operators: +
* (+) OOP advanced: operator overloading 8
* <https://kotlinlang.org/docs/idioms.html>
* Snippets: <https://git-iit.fh-joanneum.at/omd/code-snippets/-/tree/master/kotlin>

Now start coding (on your own)

* Step 1 **Your Plan**
* i.e. model your data (structures) (e.g. POIs, GPS-Track, Slideshow, Game-Players,...) and prepare at least three \*\*demo data
* minimal necessary actions for your data
* e.g. create many data points (objects) (with given values)
* e.g. output current state of a single data point
* minimal necessary actions for your set of data
* e.g. create one management
* e.g. add further data
* e.g. delete existing data points
* e.g. get list of data points (by given criteria)
* e.g. get list of sorted data points (by given criteria)
* e.g. output current state of data management (service)
* Step 2: **Code**
* data classes (for single data points)
* default values
* singleton (managing a list of data points)
* (mutable) list of elements (reverse sort, random, ...)
* elements possibly null (elvis-operator, smart cast for null-safety)
* functions with default argumens, call by named arguments
* objects with getter/setter
* Step 3: **Test + Refactor**
* Somehow test all your planned functionality (include *edge cases*!)
* Example
* #!/usr/bin/env kotlinc -script
* println("Selected Kotlin tasks:")
* println("Task: Create a slideshow management with different types of slides")
* // slideshow management://    add/remove/list/sort/filter slides// types of slides: //    text slides, image slides, animated slides, video slides
* // think of using://    data classes//    inheritance//    (lazy) properties
* Step 4 **Improve** and try (possibly **new**) features
* **filter** elements: use lambda expressions
* difference from-to POI: **operator overloading**
* "static" functionality: **companion objects**
* add features to existing objects: **extend** string
* Optional: use generics and type checking
* Upload current state
* ... so we can discuss some snippets tomorrow ...
* Outlook:
* concurrency with **coroutines**
* = **MORGEN**
* 14.10.2023 (Samstag) 08:45 - 11:00 IMS 2023 / UE MoSWDev Johannes Feiner X/ONL/Teams

1. 020g\_null.kt, 020h\_cast.kt ↩
2. 020i\_smart-casts.kt and 020j\_smart-cast+null-safety.kt ↩
3. 110h\_delegates.kt ↩
4. 110l\_data-class.kt ↩
5. 110j\_singleton.kt ↩
6. 065a\_let.kt, 065b\_run.kt, ... ↩
7. 110i\_extensions.kt ↩
8. 110c\_operator-overloading.kt ↩

TODO for JF:

* check out if "User-defined explicit and implicit conversion operators" (C#) would work in Kotlin....