**Due:** Monday, 18 December 2023, 11:59 PM

* Upload App (i.e. working App, apk installation)
* Link to source code in GIT repo
* Short statement about status of the App: which features are included, which features are work-in-progress (WIP), known bugs, next steps
* **Beta version might contain bugs, but should include at least one main feature. Bear in mind, during product presentations you should be able to convince your customers (despite having a buggy, beta app).**

*Note: It would be fine, if EVERY student could upload the documents (or a text file with the exact link to the documents in the git repo). It is easier and safer in terms of assigning the points to the right team members. Thank you very much.*

**10% Final (Deadline 2024-01-08)**

**10% Final (Deadline 2024-01-08)**

Completion requirements

**Due:** Monday, 8 January 2024, 11:59 PM

Upload your final **unobfuscated** **release APK**. Optional: Add an obfuscated variant.

**A few more links...**

**A few more links...**

* Kotlin Standard Library Cookbook June 2018 (eBook, FH Library: <http://permalink.obvsg.at/fhj/AC15521894>
* Find many, many great tutorials on **Google Developer Codelabs** [https://codelabs.developers.google.com](https://codelabs.developers.google.com/).
* For example you might start with the Android Basics in Kotlin <https://developer.android.com/courses/android-basics-kotlin/course>, then Android Basics with Compose <https://developer.android.com/courses/android-basics-compose/course>
* and move on the the expert session Advanced Android in Kotlin <https://codelabs.developers.google.com/advanced-android-kotlin-training/> later.
* Check out the free **Udacity lectures** Developing Android Apps with Kotlin <https://www.udacity.com/course/developing-android-apps-with-kotlin--ud9012> and
* **Advanced Android with Kotlin** <https://www.udacity.com/course/advanced-android-with-kotlin--ud940> (both by Google).

Last modified: Friday, 28 October 2022, 11:19 AM

[◄ 25% Battle (Deadline 2024-01-13)](https://elearning.fh-joanneum.at/mod/assign/view.php?id=55343&forceview=1)

Formularbeginn

2023-10-20 - L05

Review:

* L01 Intro = Orga
* L02 Android Tools & Basic Architecture
* Views and navigaton
* Activity + (implicit/explicit) intents (how to pass data)
* ...versus services, broadcast receiver, service provider
* How code interacts with UI (XML), R, findViewById
* Lifecylce (onResume, onDestroy)
* L03 Kotlin
* Model data + service
* Coding (in scratch files)
* L04 Kotlin Advanced
* Concurrency, deferable functions (suspend), Coroutines (-Scopes), call async:
* suspend fun,    launch { ... },    join ... wait ...  waitAll
* Idiomatic Kotlin
* immutable data, Null Safety, properties, data classes, object, scope functions (apply..), extension functions, operator overloading
* Kotlin Week (Selected Feedback)

L05

* **Infos:**
* Android Jetpack:
* \* called "Architectural components"
* \* DataBinding, Lifecycles, LiveData, Navigation, Paging, Room, ViewModel, WorkManager,...
* More Details on the (Jetpack) libraries
* \* ViewModel
* \* LiveData
* \* DataBinding
* \* Repository
* \* persistence, local-db: Room
* \* web-service: Retrofit
* \* Workmanager
* \* Load in Background, Don't block UI
* \* Navigation (Flow)
* \* Declarative UI (code)
* \* Jetpack Compose
* \* Dependency Injektion
* \* Hilt
* \* Dagger
* Architecture
* <https://developer.android.com/topic/architecture/>
* Videos
* <https://www.youtube.com/c/AndroidDevelopers/videos> ... check out channels
* MAD: <https://developer.android.com/series/mad-skills>
* or Jetpack Compose: <https://www.youtube.com/playlist?list=PLWz5rJ2EKKc9Ty3Zl1hvMVUsXfkn93NRk>
* Demos
* from git Part4 Android UI
* <https://git-iit.fh-joanneum.at/omd/droid-devel/-/tree/master/Part-4-Android-APIs>
* ... old style (activities, intents)
* ... fragments
* ... compose
* structure:
* data
* service (network, persistence)
* ui
* **Coding**
* Compose - Tutorial <<https://developer.android.com/jetpack/compose/tutorial>>

**25% Battle (Deadline 2024-01-13)**

**25% Battle (Deadline 2024-01-13)**

Completion requirements

**Due:** Saturday, 13 January 2024, 8:00 AM

* Optional: upload of slides about your findings.

**Schedule**

* Teams present their reviews (about 10-15 minutes)
  + e.g. Team T4: give a **Live Demo** of how reviewed the **app (of T3) pretends to work**, then present your **review**: how we found which hidden features (we found evil behaviour of a, b, c… by analysing db/nw/code ....)
  + e.g. Team T3 responds: yes, but we have **also done** d (and you have not found it)
  + e.g. Team T4 explains: we could **exploit** your lousy App by x
* ...then T3 goes on with analysis of T2 and then ...

*Note:* present finding, *how* what was found. Show (demo, code, report) the findings. Slides are optional.

**1/4 Upload Sensitive Information from Locating Secrets (until 22. Dec)**

**1/4 Upload Sensitive Information from Locating Secrets (until 22. Dec)**

Upload following information:

* Admin Access Code
* Username & Password for webservice
* Result from webservice (with admin access)
* short description of application code. (what is the app doing? 3-4 lines)
* **2/4 Upload Screenshot of working HTTPS Interception (until 31. Jan)**
* **2/4 Upload Screenshot of working HTTPS Interception (until 31. Jan)**
* Follow the guide in the exercise description to configure an interception proxy. Upload an image showing the intercepted traffic and the emulator.
* **3/4 Upload Frida Script from Uncrackable App (until 31. Jan)**
* **3/4 Upload Frida Script from Uncrackable App (until 31. Jan)**
* Upload your working Frida script for the uncrackable App.
* **4/4 Upload Take aways from guest lectures (until 31. Jan)**
* **4/4 Upload Take aways from guest lectures (until 31. Jan)**
* Describe your take aways from the guest lecture in 3 to 6 lines. What did you enjoy the most? What was new to you? Did you ever encounter a tamper protection from Denuvo? etc.

**Deadline 7. Nov. 2023**

* Title
* Team members
* Description of App idea, slogan
* Features (prioritised, three *must-have* features)
* Hidden feature(s)
* Software engineering method
* Time estimate
* Paper prototype

*Max 4. Pages*

*Hint 1:* Draw your App (idea), i.e. create a paper prototype. We suggest to focus on the problem / persona first, then draft your solution. Explain user interaction and features(see Hint 2). Priorise your features and mark two or **three** as **MUST-HAVES**. Compare top features of similar apps.

*Hint 2 / Brainstorming*: Technical "features" could be Location Based Services (Maps), Data Synchronisation via Cloud, Real-Time-Communication between multiple apps, use of Compass/Barometer/Accelerometer sensors, Local and Push-Notifications, connection to IoT devices such as weather station or home automation, usage of (low-energy) bluetooth-network, camera/audio to scan QR-codes, Custom-Gesture-Input, using crypto-algorithms to store data on SD cards in a secure way, integration of system services (addresses, photos, calendar), Social Media Sharing, Background-Sync, using the Native SDK, Dynamic Code Loading/Execution, Game, useful (and nice) animations/visualisations, .... augmented/virtual reality (AR/VR) or machine learning (ML sentiment analysis, image classification).

*Note: It would be fine, if EVERY student could upload the documents (or a text file with the exact link to the documents in the git repo). It is easier and safer in terms of assigning the points to the right team members. Thank you very much.*

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

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....

**5% Peer Review (Deadline 2024-01-11)**

**5% Peer Review (Deadline 2024-01-11)**

Completion requirements

**Due:** Thursday, 11 January 2024, 11:59 PM

Analyse the project of your competitors. Upload your findings as Markdown document or PDF. About 10 pages including screenshots. Structure: What you plan to find (hardware/sensors, software, data,..), how you will inspect (static, dynamic, environment,...), results.

* App **a.apk** will be analysed by Team-----

**WARNING**: the app you review might contain malicious code! Run on your own risk!!

**Possible Exam Questions**

In Section Overview: Droid Development

The answers to following questions can be found in the [Study Material](/omd/droid-devel/-/blob/master/Overview/study-material--overview.md).

Hint: Explain your answers to a fellow student!

**Apps and Architecture**

* How does the **architecture of Android apps** differ from typical iOS-Apps?
* What is the main entry point into an app. Explain what the *manifest* file is used for. Explain, what are *activities* and what are *intents*?
* Explain and give examples of permissions granted at install time. Compare to runtime permissions.
* Could you explain terms such as activities, fragments, and views.
* Which unique ids and which certificates are necessary for development, signing and distribution?

**Kotlin Programming**

* How and where is Kotlin code executed?
* How does garbage collection work?
* Explain some of the key language features of Kotlin. For example: **type inference**, strong typing.
* Discuss compile-time vs. run-time errors/exceptions.
* What ar nullable types.
* Kotlin idioms (*idiomatic programming*): compare two of selected features (such as data classes, smart casts, operator overloading, generics, coroutines, default values, lazy properties, singletons, not-null shorthands, if-expressions...) to other languages.
* Can you show the usage of object-oriented programming (including initialisers, inheritance, **properties**, **extensions**, …) in Kotlin?
* Are calsses or data classes **passed by value**? Explain the (dis-) advantages of **pass by reference**.
* What is special with **Nullable types** and with the **save call operator** ?.?
* What are **default arguments** for functions?
* Fuctional programming: explain the use of **high-order-functions** and **lambdas**.
* Can I explain and demo advanced features of the Kotlin programming language: generics, **operator-overloading**, …
* Could you explain the usage, the advantages and disadvantages of advanced Object Oriented Programming (OOP) with Interfaces and Extensions?
* Explain the differences between **primary** and **secondary constructors**.
* Delegates: what could be an usage of **observable** properties?
* Why would many consider val more secure than var?
* Argue for the usage of == over ===.
* Compilation (on demand): could you explain the work of the Android Runtime (**ART**), which combines the advantages of *just in time* compiling with *ahead of time* compiling.

**GUIs, Navigation, and Accessibility**

* Explain what is necessary to create interfaces for Android Apps.
* Explain the use of an activity and a fragment.
* How can developers assign code for life-cycle methods (view appears) and user interactions (swipe gestures, tap a button).
* Name ways for **navigation**, and explain how to **pass data** from one activity to another.
* Name ways for **navigation**, and explain how to **pass data** from one fragment to another.
* Code generation: explain how (two-way) **data binding** works.
* Could you explain the most important **life-cycle** events?

**Concurrency and Web Services**

* How to keep the UI responsive with background tasks for longer operations?
* How to retrieve data in the background from web services?
* Explain the use of **suspendable functions** with **coroutines** to prevent blocking UIs and to improve performance?
* How and why are ReSTful web services called **asynchronously** (in concurrent threads)?
* Explain ways of **JSON serialisation** and the elaborate on the problem of type checking?
* How and when would you use **threads** over **coroutines**?

**Saving State and Persistency**

* How do **shared preferences** work? Give an example.
* Name three different ways to persist data.
* How to setup the O/R-Mapping (ORM) and how to update a schema?
* For which kind of data is it necessary to use secure storage?
* Explain the disadvantage of keeping data in a keystore?
* What is the difference between eager and lazy loading!
* Explain the main features of **firebase** platform, expecially authentication and the cloud firestore.

**Location Based Services**

* How to minimize user input? How to suggesting location, movement and context?
* What is necessary to use maps and add custom landmarks and points-of-interest (POIs)?
* How to use existing web services to improve user-experience?
* Could you explain the usage, the advantages and disadvantages of Location Based Services concerning privacy?
* Maps: what differences are between *Overlays*, *Markers*, and *Tiles*?
* Which accuracy can we expect from different ways of determining the location of a user?
* Could you name the idea and application of **geofencing**?
* What are the use and (practical) limitations of the **(reverse) geocoding**?

**Sensors and Actuators**

* Explain the advantages and disadvantages (according to a context) of polling versus callbacks to retrieve sensor data.
* How and when to employ different sensors (and different frameworks)? (Proximity, Ambient Light, ...) to provide feedback to users?
* Could you explain the usage, the advantages and disadvantages as well as the proper technical terminology for at least two actuators and five sensors?
* What is special (concerning the performance, power, connection time, kind of connection, security, ...) about bluetooth low energy (**BTLE**)?

**Operating System Security and Insecurity, Forensics, App Analysis and Secure Coding.**

* Explain the **Android Virtual Machine** and how it is embedded into the **Android Stack**.
* Consequences of **sandboxing** an app. How are restrictions/permissions enforced.
* What does it mean to code in a secure way? Give 3 examples!
* How and where to save client data (user credentials) in a secure way?
* Why and how is code quality related to security?
* Explain approaches for testing.
* What means code integrity and how to enforce code integrity.
* How can usability build trust, and could you as developer care for user privacy?
* Which steps are necessary/possible for an \*.apk debugging session?
* (OWASP) **risks**: give examples for *insecure data storage*, *insecure communication*, *insufficient cryptography*, and *insecure authorisation*.
* Explain the deployment certificates needed to put an app to the App Store?
* Could you draw the architecture and flow during usage of **biometric sensors**?
* Which limitations and restrictions are given for **inter-app communication**?
* Name ways for inter-process communication (**IPC**) and known is uses?
* Differences between **certificate** and **public key** pinning.
* Explain the use of **URL schemes** in Android?
* Could you discuss the different approaches of **separating apps** on Android and on iOS including the consequences?
* Can you explain the terms and specific tasks of **Secure Elements**.
* Draft the steps to **decompile** .apk files.
* Name relevant aspects of the Android *permission model*. Detail on differences of **install-time** (normal/signature), **runtime** (prompting users), and **special** (platform/OEM) permissions.
* Limitation of **code obfuscation**. What does it mean that the R8 compiler shrinks, obfuscates and optimises an app.
* **Certificates**: Explain the terms (and differences): *client certificate*, *root certificate*, *certificate authority*, *self-signed certificate*, and *certificate chain*.
* Name at least three ways to harden your app.
* Compare **static** to **dynamic** (malware) **analysis**.

**System Services and Social Frameworks**

* Name and explain kinds of data exchange (not) possible between apps and the system, such as clipboard, file system, inter-process communication (IPC). Explain consequences to security and privacy. How are restrictions enforced?
* How to integrate App logic with system services such as AddressBook, PhotoAlbum or Calendar? Could you sketch the steps for accessing contacts?
* In which ways it is possible to integrate social media services for sharing or login?
* Which way of communication from Android to Android watches can be implemented?
* Explain the basic concepts of selected IPC methods, such as **binder, signals, sockets/ports, streams, pipes, or shared memory**.

**Cloud Services, Monitoring, and Testing**

* Which Google cloud services are available to developers and how to integrate them into own apps? Name limits and consequences, state ways to store documents transparently in the cloud.
* Can you explain different ways of synching to cloud storage?
* Which privacy and security issues have to be considered?
* How to **test asynchronous code**?
* What are the requirement for stable **UI-Tests**? What are the limits for UI-Testing?
* Which drawbacks concerning performance, security, footprint, cross platform usage are to expect with different cloud service providers?
* How could a cloud based password store be safe?

**Local and Remote (Push) Notifications**

* What are the important elements of the Google **push notification architecture**?
* Could you name the main differences between the Apple, Google and Microsoft Push Services?
* State ways to consume the notification inside the client app?
* Can local and/or remote push notifications mitigate the problem of battery draining background processes?
* Could you draft first the **registration process**, the creation and use of credentials/logins/tokens, and then the flow of notifications. Explain the flow (step-by-step) and show how to setup push notification on the server?
* How and where can push notification services be attacked? Consider the security implications for developers, especially the app server with custom setup and custom logic.
* How **background services** work with Android compared to push notifications.

**3% Upload Security Improvements**

**3% Upload Security Improvements**

**Security Improvements, Hardening for Source Code.**

Zip and upload a (might be minimal) Android Studio Project which includes

* (a) code to save credentials or private data into the keystore or at least to secured shared prefs,
* (b) code to encrypt data, and
* (c) prove that you shrinked, obfuscated, and optimized your app (e.g. upload different apks or just use "ls" or "du" or " jadx -d /tmp/out \*.dex" or screenshot or ... to show the differences before/after your changes n gradle-config). Optional: apktool / jadx are possibly useful, to view source code (and size) of \*.apk / \*.dex.
* (d) a readme.md which contains the steps/configs special to your solution and the effect of optimising, for example the reduced size of apk (smaller in %).

Possibly useful: <https://developer.android.com/studio/build/shrink-code>