Effective Kotlin workshop

During the workshop, we'll dive deeply into Kotlin best practices and idiomatic Kotlin use. We'll also talk about more general concepts like readability, abstraction design or contract setting.

The workshop is based on the book Effective Kotlin that is soon to be published. The content is well thought out and consulted with many Kotlin developers.

In the first 2 days, we are covering general best practices and class design. The 3rd day is dedicated to code efficiency.

Most important best practices we'll be covering are:

Safety

- Limit mutability
- Eliminate platform types as soon as possible
- Do not expose inferred types
- Prefer composition over inheritance
- Make it clear that top-level functions are not member functions

Readability

- Design for readability
- Use operator overloading methods as their names indicate
- Consider naming arguments
- Avoid returning or operating on Unit?
- Specify variable type when it is not clear
- Consider referencing receiver explicitly

Reusability

- Do not repeat knowledge
- Do not repeat common algorithms
- Reuse between different platforms by extracting common modules

Design abstractions

- Each function should be written in terms of a single level of abstraction
- Define abstractions
- Respect abstraction contract
- Restrict visibility
- Use abstraction to protect code against changes
- Consider wrapping external API

Objects creation

- Consider factory functions instead of constructors
- Consider primary constructor with named optional arguments
- Consider defining DSL for complex object creation

Class design

- Use data modifier to represent a bundle of data
- Use function types instead of interfaces to pass operations and actions
- Do not use properties to express behaviour
- Respect contract of equals
- Respect contract of hashCode
- Respect contract of compareTo
- Respect contract of arithmetic operators

Included additionally in a 3-day workshop:

Make it cheap

- Avoid unnecessary object creation
- Consider inline modifier for higher-order functions
- Eliminate obsolete object references
- Use tail recursion to achieve efficient recurrence

Efficient collection processing

- Prefer Sequence for bigger collections with more than one processing step
- Consider Arrays with primitives for performance critical processing
- Consider using mutable collections

On this way, we'll also review more advanced topics including:

- Collections
- Delegates
- Platform types
- Interoperability with Java
- Typing system
- Operator overloading
- DSL creation
- Functional programming
- Computational complexity

Included additionally in a 3-day workshop:

- Inline classes
- Computational complexity