Kotlin language

Introduction

What is Kotlin ?

Kotlin is a statically-typed programming language that runs on the Java Virtual Machine and also can be compiled to JavaScript source code or uses the LLVM compiler infrastructure.

Developed by JetBrains.

Kotlin is designed to interoperate with Java code and is reliant on Java code from the existing Java Class Library, such as the collections framework.

Timeline & history

- July 2011 - JetBrains unveiled Project Kotlin
- **February 15, 2016** - Kotlin v 1.0 released
- May 18, 2016 - Kotlin Meets Gradle - writing gradle build scripts in Kotlin
    - https://blog.gradle.org/kotlin-meets-gradle
- March 1, 2017 - Kotlin v 1.1 released
- **May 17, 2017** - Google announced first-class support for Kotlin on Android
- July 2017 - Spring 5 adds Kotlin support
    - https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0

Runs on:

- Java Virtual Machine

- Android

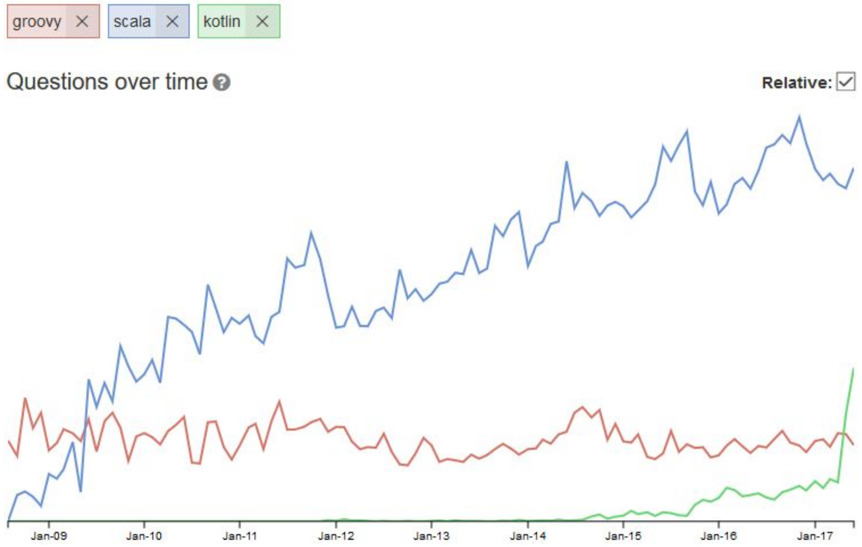
- JavaScript

Kotlin
KETCHUP ŁAGODNY
MILD KETCHUP
60% mniej kalorii
Ten sam pomidorowy smak i mniej kalorii
niż ketchupy z dodatkiem cukru
NOWOŚĆ

- Kotlin/Native
- using LLVM
- iOS but not only

Popularity

By number of questions on StackOverflow (06.2017):



Very popular cosidering how young it is (1 year after 1.0).

Popularity driven by Android platform (because using Java 1.8 was not possible).

DEMO - Live Coding

https://github.com/kkot/kotlin-intro

Features #1

- **Null safety**, smart cast
- Nice clean syntax: types go last, optional semicolon, no *new*, “==” calls equals
- Default values, Named parameters
- String templates and “””
- If is expression, *when* expression
- Primary constructors, secondary constructors
- Data class
- No static method, support for singletons (object)
- Extension methods
- Properties (see slide)
- Seprate interfaces for read-only collections
    - cleaner API, easier to keep track
    - readonly collections are invariant

Features #2

- Inline functions
- Declaration-site variance
- **Coroutines !!! (beta in 1.1)**
    - **async, awaits implemented as library**
    - **cheap threads (similar to goroutines in Go)**

Properties

In some cases functions with no arguments might be interchangeable with read-only properties. Although the semantics are similar, there are some stylistic conventions on when to prefer one to another.

Prefer a property over a function when the underlying algorithm:

- does not throw
- has a O(1) complexity
- is cheap to calculate (or cached on the first run)
- returns the same result over invocations

Differences from Java

full list: https://kotlinlang.org/docs/reference/comparison-to-java.html

- no checked exception
- classes and members are by default **public** but final :(
- no package private visibility modifier
  - there is **internal**
  - private top members behave as package private
- multiple public classes per file are possible
- **Any** class similar to Object but has only equals, hashcode, toString
- no raw types
- package and directory may be different :(
- no implicit conversions between number (for ex. between Int and Long)

Final by default

Unlike in Java classes and methods are final by default (Interfaces are open).
To override a method or extend a class **open** modifier must be added.

Why ??????

- Effective Java, Item 17: Design and document for inheritance or else prohibit it.
- But actually:
  - https://discuss.kotlinlang.org/t/a-bit-about-picking-defaults/1418
  - smart cast of val is not possible when class is open
  - open var can’t have private setters
- Little annoying but: Mockito 2 can mock final classe and for Spring there is compiler plugin that open

Coroutines

Coroutines are computations that can be suspended without blocking a thread. Blocking threads is often expensive, especially under high load, because only a relatively small number of threads is practical to keep around, so blocking one of them leads to some important work being delayed.

- Examples:
    - async/await from C# and ECMAScript
    - channels and select from Go
    - and generators/yield from C# and Python

In Polish: współprogram

Kotlin and Lombok

Kotlin doesn't see Java Lombok accessors?

Generally, no, it doesn't. The reason of that behavior is that Lombok is an annotation processor for javac but when the kotlin compiler runs it uses javac as well but with no annotation processing so this is why kotlin don't see declarations that wasn't yet generated.

https://stackoverflow.com/questions/35517325/kotlin-doesnt-see-java-lombok-accessors

No good solution, partial:

- Delombok (removing lombok) and convert to Kotlin
- Putting Java and Kotlin code in different modules

Extension methods and mocking

Extension method are implemented as static methods calls.

It is not possible to mock them same way as ordnary method and this might cause problem with unit testing. For example if we call an extension method on our dependency.

// This is just my feeling need to be check in pactice.

https://discuss.kotlinlang.org/t/mocking-extension-functions/2835/5

Thank you for listening