Kotlin Programming

Alexandre Luiz de Borba Silva

Linkedin: albsilva0 - GitHub: lekzinn

Objective

Kotlin - Why?

- JetBrains - 2010
- We kind of needed a language that was..
  Concise, Expressive, Interoperable, but above all, pragmatic
- Two Options: Ceylon e Scala
- Apache 2 OSS License

Objective

Kotlin

- Statically Typed Language
- Inspired by Java, Scala, C# and Groovy
- Targets: JVM, JavaScript, Native
- Very good adoption in Android
- Similarity to other languages
- Interoperable allows gradual adoption

Objective

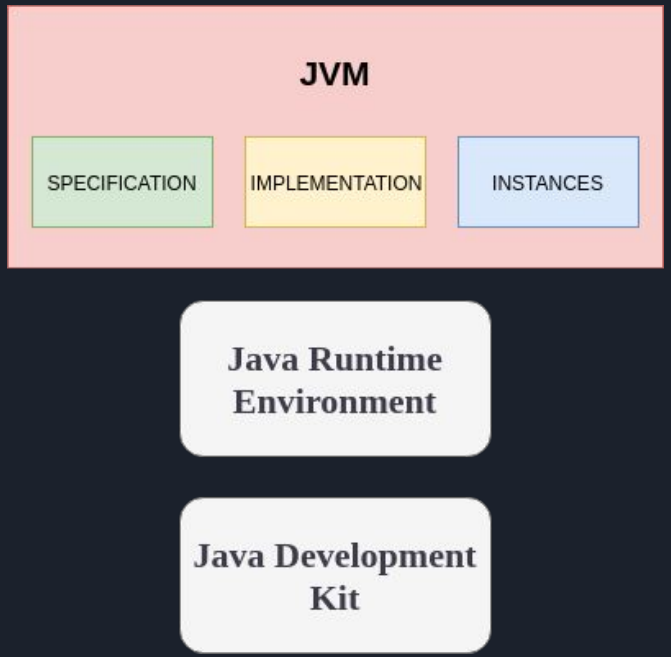
Current state

- The first official 1.0 release was in February 2016.
- The currently released version is 1.3.50, published on August 22, 2019.
- Current team size is 50+
- 250+ external contributors on GitHub.
- Amex, Netflix, NBC News, Square, Pinterest, Basecamp or Corda.

Objective

The Java Virtual machine
A.K.A The JVM

- Abstract machine running Java Applications
- Single specification, Multiple implementations
- An instance is what runs a Java application
- Applications compiled to Bytecode
- JRE and JDK



Objective

Languages on JVM

- Truly Polyglot ecosystem
- All languages either compile to bytecode (or transpile to Java first)
- Java, Kotlin, Scala, Ceylon, Clojure, Frege

Objective

Kotlin and JVM

Customer.kt

```
class Customer(val name: String) {
    fun validate(): Boolean {
        …
    }
}
```

compiler

Customer.class

```
L0
    LINENUMBER 5LD
    ALOAD 0
    GETFIELD Customer.name
    ARETURN
L1
    LOCALVARIABLE this LCustomer
    MAXSTACK = 1
```

Execute using the JVM
java myApp

Objective

Kotlin and Java

- Java and Kotlin are 100% interoperable
- You can call Java from Kotlin and vice-versa

Objective

Kotlin Basics

- Two ways to declare variables in Kotlin
  - var for mutable
  - val for immutable
- For loops can use ranges (1..100) for iteration
- if and when can be used as expressions

```
// mutable variable
var stringVar: String = "High
Street"
var anotherStreetName = "High
Street"
// immutable variable
val myLong = 10L
```

```
for (a in 1..100) {
    println(a)
}

val capitals = listOf("London",
"Paris", "Rome", "Madri")
for (capital in capitals) {
    println(capital)
}
```

```
val whenReturn = when (myString) {
    "Not Empty" -> {
        println("Nao Esta vazio!")
        "Retorno: Nao Esta vazio!"
    }
    is String -> "Eh String"
    else -> "Default Value"
}
```

Objective

Kotlin Functions

- Functions are declared using keyword `fun`
- by default return type is `Unit`, equivalent to `void`
- functions allow
  - default parameters
  - named parameters
  - unlimited arguments
- single expression functions don't need function block

Objective

Kotlin Functions

```kotlin
fun hello(): Unit {
    println("Hello")
}

fun trowException(): Nothing {
    throw  Exception("This fun throw a exception")
}

fun sum(x: Int, y: Int, w: Int = 0, z: Int = 0) = x + y + w + z
```

Objective

Kotlin Visibility Modifiers

- 4 visibility modifiers
  - public - Default and anywhere accessible
- Top level declarations
  - private - Available inside file containing declaration
  - internal - Anywhere in the same module
- Classes
  - private - Only available to class members
  - protected - Same as private and subclasses
  - internal - Any client inside the module

Objective

Kotlin Classes

- Classes are first class citizens in Kotlin
- Provide expected functionality such properties and functions
- The are no fields in kotlin
- We have access to "backing" field if required in custom getters/setters
- Data classes provide functionality to reduce boiler-plate code
- Objects provide an easy way to create singletons

Objective

Kotlin Inheritance et al.

- By default classes are final in Kotlin
  - open annotation allows classes to inherit
- Abstract classes are similar to interfaces, but allow state
- Kotlin support for generic types and functions

Objective

Kotlin Null Safety

- By default types cannot be null
- We can declare types as nullable by suffixing a `?`
- The `?` operator also providers us the ability to access members when value not null
- When we have a nullable reference r, we can say "if r is not null, use it, otherwise use some non-null value x", using the `?:` operator
- the `!!` operator provide a way to access members without the null checks

Objective

Kotlin Tidbits

- Smart cast in Kotlin can save on boiler-plate code
- Tuples are restricted to Pairs and Triples
- Value Deconstruction allows better semantics
- No checked exceptions in Kotlin
- Constants can be objects or top-level properties
- Annotations are available in Kotlin and compatible with java

Objective

References

- Github repository
- Functional Calisthenics
- Kotlin Docs reference
- Functional Programming (Uncle Bob): What? Why? When?
- Introduction to Kotlin Programming
- Advanced Kotlin Programming

Objective