CN GROUP
A CIKLUM COMPANY

Martin Müller

Java developer since 2010

adopted Groovy (Spock) testing in 2018

replaced Java with Kotlin in 2019

lives happily ever after with Kotlin & Spock ♥


Kotlin

What is Kotlin

- Statically typed language for JVM, Android, browser and native

What is Kotlin

- Statically typed language for JVM, Android, browser and native
- Developed by JetBrains, recommended by Google for Android

What is Kotlin

- Statically typed language for JVM, Android, browser and native
- Developed by JetBrains, recommended by Google for Android
- Similar to Java, less boilerplate code, more readable code

What is Kotlin

- Statically typed language for JVM, Android, browser and native
- Developed by JetBrains, recommended by Google for Android
- Similar to Java, less boilerplate code, more readable code
- Can be used together with Java and all Java tools and frameworks

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}
```

Functions

```java
// Java
public int sum(int a, int b) {
    return a + b;
}
```

```kotlin
// Kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun sum(a: Int, b: Int): Int = a + b

fun sum(a: Int, b: Int) = a + b
```

Named arguments

```
fun build(title: String, width: Int = 800, height: Int = 600) {
   Frame(title, width, height)
}
```

Named arguments

```
fun build(title: String, width: Int = 800, height: Int = 600) {
   Frame(title, width, height)
}

build("PacMan", 400, 300)
build(title = "PacMan", width = 400, height = 300)
build(width = 400, height = 300, title = "PacMan")
```

Default values

```
fun build(title: String, width: Int = 800, height: Int = 600) {
   Frame(title, width, height)
}
```

Default values

```
fun build(title: String, width: Int = 800, height: Int = 600) {
   Frame(title, width, height)
}

build("PacMan")
build("PacMan", 400)
build("PacMan", height = 300)
// still the same method, no method overloading needed
```

Variables

```
val a: String = "A" // val is unmodifiable
val b = "B" // type can be omitted

var c = "C" // var is modifiable
c = "CC"
```

Null safety

```
var a: String = "abc"
a = null                // compile error - not nullable type

var b: String? = "xyz"
b = null                // no problem
```

Null safety

```
val c: String? = "xyz"
val x = c.length       // compile error: c might be null

if (c != null) {
   val x = c.length       // no problem
}
```

Null safety

```
val c: String? = "xyz"
val x = c.length       // compile error: c might be null

if (c != null) {
   val x = c.length       // no problem
}

val x = if (c != null) c.length else null
val x = c?.length
val x = c?.length ?: 0 // 0 instead of null on the output
```

Null safety

```
val c: String? = "xyz"
val x = c.length       // compile error: c might be null

if (c != null) {
   val x = c.length       // no problem
}

val x = if (c != null) c.length else null
val x = c?.length
val x = c?.length ?: 0 // 0 instead of null on the output
```

Safe call operator

```java
// Java
public String getZipCode(User user) {
   if (user != null && user.getAddress() != null) {
      return user.getAddress().getZipCode();
   }
   return null;
}
```

Safe call operator

```java
// Java
public String getZipCode(User user) {
   if (user != null && user.getAddress() != null) {
      return user.getAddress().getZipCode();
   }
   return null;
}
```

```kotlin
// Kotlin
fun getZipCode(user: User?) = user?.address?.zipCode
```

Scope Function

```
data?.let{
    logger.info ( "Doing my stuff" )
    callExternalService(data)
}

return getSessionFactoryBase(config).apply {
    setPassword(config.password)
}
```

When expression

```
when (x) {
    1 -> print("One")
    2, 3 -> print("Two or Three")
    in 5..10 -> print("is 5 to 10")
    else -> print("unknown")
}
```

When expression

```
val res = when(x) {
   null -> false
   is Long -> x > 0
   is String -> x.length > 0  // smart cast used here
   else -> throw IllegalStateException()
}
```

Extension functions

```
fun String.toCamelCase(): String = this.replace(...)
// effectively adding new method to Java String

"hello word".toCamelCase()
```

Extension functions

// standard library extension functions:

"hello word".*reader*().*forEachLine* **{ ... }**

Extension functions

```
// standard library extension functions:

val file: File = File("readme.txt")

val writer = file.bufferedWriter()

file.writeText("simple")
```

Collections

```
// Java
list.stream()
        .filter(number -> number > 0)
        .collect(Collectors.toList());


// Kotlin
list.filter { it > 0 }
```

Collections & extension

```
operator fun List<Int>.times(by: Int): List<Int> {
    return this.map { it * by }
}

val foo = listOf(1, 2, 3) * 4      // [4, 8, 12]
```

Classes

```
class Person(
        val name: String,
        var email: String,
        var age: Int
        var title: String = ""
) {
    fun greet() {
        print("Hello, here $name")
    }
}
```

Classes

```
class Person(
        val name: String,
        var email: String,
        var age: Int
        var title: String = ""
) {
   fun greet() {
        print("Hello, here $name")
   }
}

val john = Person("John", "john@gmail.com", 112)
val john = Person(name = "John", email = "john@gmail.com", age = 112)
val john = Person(name = "John", email = "john@gmail.com", age = 112, title = "Dr")
```

Inheritance

```
open class Shape {
   open fun draw() { /*...*/ }
   fun fill() { /*...*/ }
}

class Circle : Shape() {
   override fun draw() { /*...*/ }
}
```

Singleton

```
object MySingleton {
    fun doSomething() {}
}
```

Data classes

```
data class Person(
        val name: String,
        val email: String,
        val age: Int
)
```

Data classes

```
data class Person(
        val name: String,
        val email: String,
        val age: Int
)

val john =       Person("John", "john@gmail.com", 50)
val john2 =      Person("John", "john@gmail.com", 50)
val johnson = Person("Johnson", "johnson@gmail.com", 20)

john == john2  // true
john == johnson // false
```

Data classes

```
val jack = Person(name = "Jack", age = 1)

val olderJack = jack.copy(age = 2)
```

Data classes

```
val jack = Person(name = "Jack", age = 1)

val olderJack = jack.copy(age = 2)


val jane = Person("Jane", 35)

val (name, age) = jane  // destructuring declaration

println("$name, $age years of age") // prints "Jane, 35 years of age„
```

Data classes

```
val jack = Person(name = "Jack", age = 1)

val olderJack = jack.copy(age = 2)


val jane = Person("Jane", 35)

val (name, age) = jane  // destructuring declaration

println("$name, $age years of age") // prints "Jane, 35 years of age"


println(jane) // data class implements toString
```

Kotlin features

- Null safety
- No checked exceptions
- Extension functions
- Higher-order functions
- Function types & lambdas
- Default & named arguments
- Properties
- Type inference
- Operator overloading
- Smart casts
- Data classes
- Immutable collections
- Enhanced switch-case
- String interpolation
- Ranges
- Inline functions
- Infix notation
- Tail recursion
- Coroutines (async/await)
- Great Standard Library
- Sealed classes
- Delegated & lazy properties
- Class delegation
- Singletons
- Nested functions
- Object decomposition
- Top-level functions
- Reified generics
- Raw Strings
- And more…

Demo

Java -> Kotlin using IntelliJ

Gradle

scratch files (prototyping, playground)

Kotlin Bytecode (see Java code equivalent)

Spring, JPA, Lombok plugins

Demo

Check project Wishlist to compare

- Java & JUnit
- Kotlin & Spock

(simple Spring Boot app with Hibernate and REST API)

What else?

- Pod Vocasem
  - Funkcionální programování v Javě
  - Artificial Intelligence v recruitmentu, Azure Cloud a Kotlin
- Speaker’s Corner
  - Spock: Test Fast & Prosper!

Try Spock as JUnit & Mockito replacement! (link)

CN GROUP
A CIKLUM COMPANY