SOURCELABS

Kotlin for Spring Developers

About me

- 40 years of experience
- Father of 2 kids
- Hobbies: CrossFit and windsurfing
- Co-founder of Sourcelabs
- Programming Java since version 1.1
- Tried other languages like VB, PHP, Ruby, C++, JavaScript, Scala, Groovy
- Like to work with new technologies and to figure out if it brings value

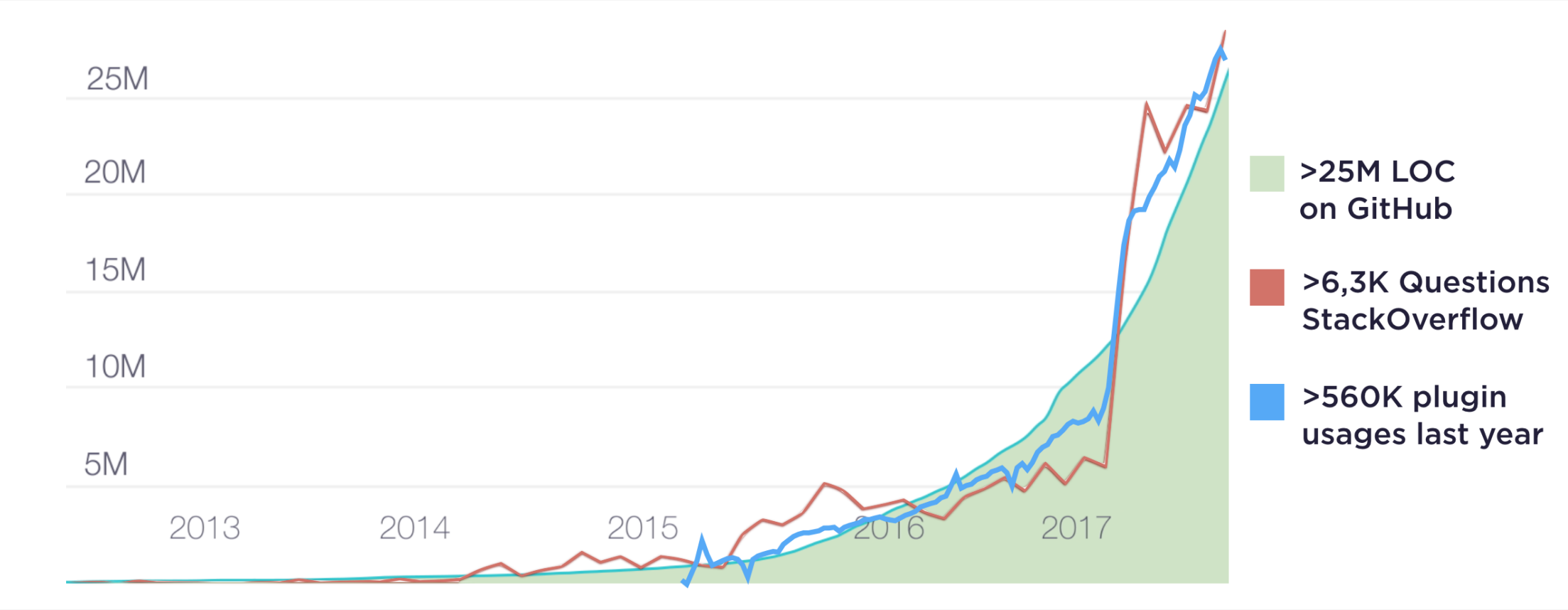
Raise hands

- Who is longer than 5 years in software development
- Who has already worked with Kotlin?
- Who (still) likes to write Java code?
- Who has been writing Java code longer than James Gosling?
- Who has more than 5 years experience with Spring Boot?

Practical information

- Morning
  - More than an introduction to Kotlin
- Lunch
  - 60 minutes
- Afternoon
  - What Spring has to offer to Kotlin developers

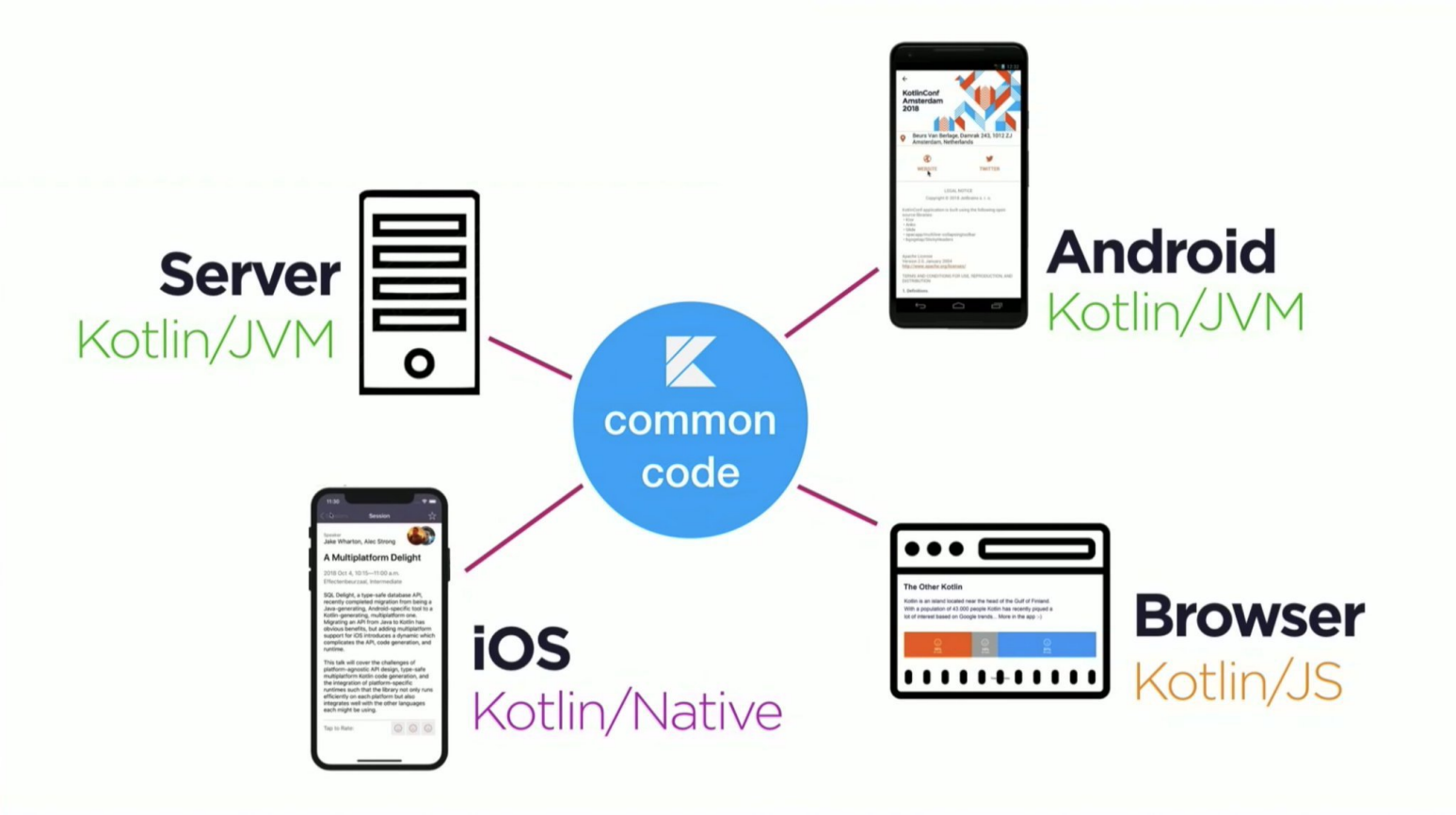
Kotlin adoption



Kotlin

- General purpose language
  - Supports both OO and functional programming paradigms
- Interoperability with Java
  - Re-use existing tools and ecosystem (IDE, build tools, libraries, frameworks, etc.)
- Pragmatic
  - Not a research project
- Kotlin is safe
  - Kotlin compiler can help to prevent even more possible types of errors.
- Open-source project
  - Mainly developed by JetBrains
- Staticly typed
  - Mostly types can be omitted

Kotlin = multiplaform



How to start with Kotlin

- Introduce in small parts of your existing application
- Gain some real world experience with it

From Java to Kotlin

- Video: 01. From Java to Kotlin.webm

Converted Kotlin code != Idiomatic Kotlin code

- Converting Java code to Kotlin results in Java style Kotlin code
- Writing **idiomatic Kotlin** code is **manual work**

Hello, World in Kotlin

- Video: 02. Hello, World Kotlin.webm

Exercise 1: getting started with Kotlin in your (Maven) project

Objectives

- Learn how to configure a Kotlin in a Maven project using IntelliJ
- Learn how to convert Java code to Kotlin using IntelliJ

Exercise 1: configuring the maven pom.xml

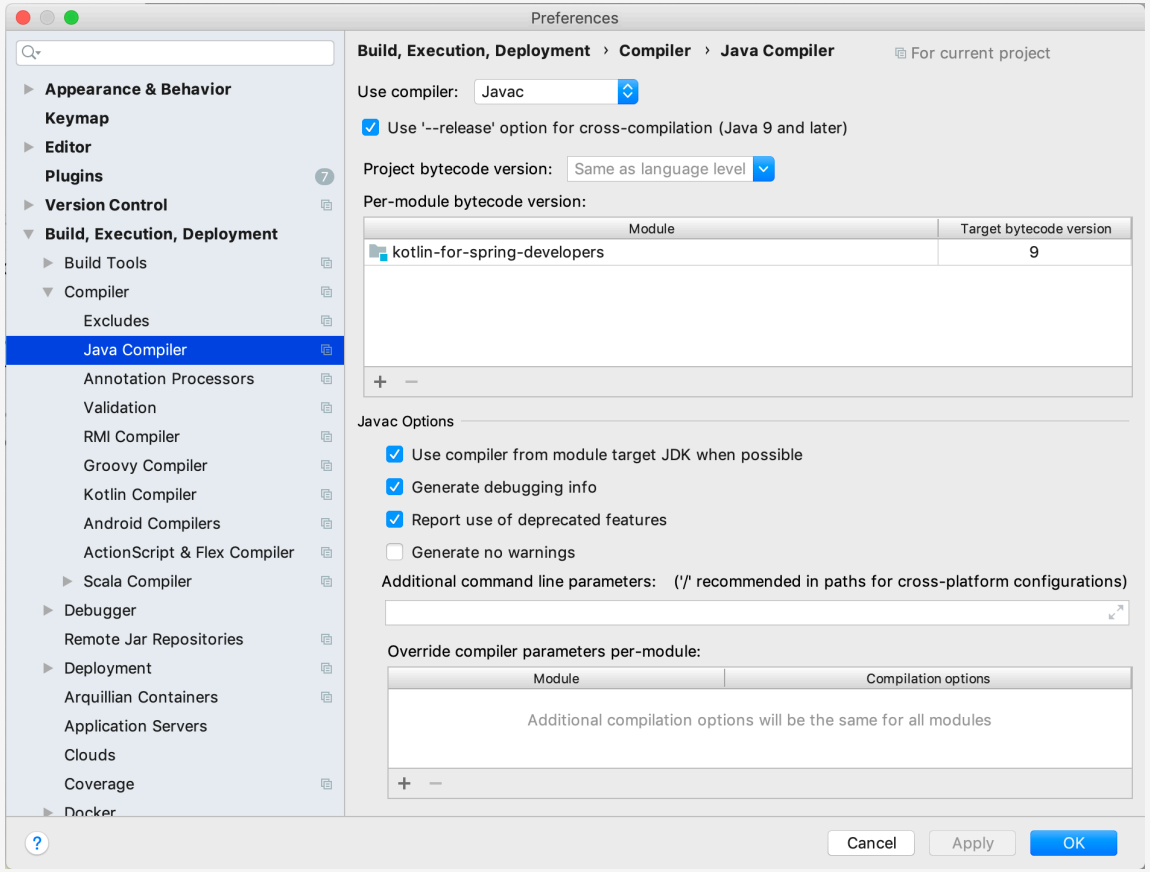
Java compiler

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>11</source>
    <target>11</target>
  </configuration>
</plugin>
```

Kotlin compiler

```
<plugin>
  <groupId>org.jetbrains.kotlin</groupId>
  <artifactId>kotlin-maven-plugin</artifactId>
  <executions>
    <execution>
      <id>compile</id>
      <phase>process-sources</phase>
      <goals>
        <goal>compile</goal>
      </goals>
    </execution>
    <execution>
      <id>test-compile</id>
      <phase>test-compile</phase>
      <goals>
        <goal>test-compile</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <jvmTarget>1.8</jvmTarget>
  </configuration>
</plugin>
```

IntelliJ compiler settings



Exercise 1

- Create a new Maven project in IntelliJ
- Create Hello.java
- Convert Hello.java to Kotlin
- Configure Kotlin in Maven
  - Manually (see cheat sheet)
  - Using IntelliJ
    - IntelliJ will warn that Kotlin is not configured
    - Configure as Java with Maven
    - Enable auto import for Maven

file: Hello.java

```
package java;

public class Hello {

    public static void main(String[] args) {
        String name = "Java";
        System.out.println("Hello, " + name + "!");
    }
}
```

Variables

- Video: 03. Variables

Variables

- var: mutable
- val: read-only (after assignment) like **final** in Java
- Mutability also explicit for collection types
- Local type inference by compiler

Functions

- Video: 04. Functions

Functions

- Function body can be an expression
- Different types: top-level, member and local function
- Top-level function are like static functions in Java

Named & default arguments

- Video: 05. Named & default arguments

Exercise 2: functions and variables

Objectives

- Understand how to write functions and arguments
- Understand the difference between val and var
- Understand how type inference works

Exercise 2

- Implement in Kotlin

file: Calculator.java

```
package java;

public class Calculator {

    public int sum(int a, int b) {
        return a + b;
    }

    public int sum(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String[] args) {
        Calculator f = new Calculator();
        System.out.println(f.sum(0, 1));
        System.out.println(f.sum(0, 1, 2));
    }
}
```

Conditionals if & when

- Video: 6. Conditionals if & when

Conditionals if & when

- **if** and **when** are expressions
  - You can return or assign the result to a variable
- Smart casts by using the **is** keyword

Loops

- Video: 07. Loops

Loops

- Iterate using **in** keyword
- Use **.withIndex** for a for-loop with an **(index,element)** variable

Exceptions

- Video: 09. Exceptions

Exceptions

- Kotlin has no checked exceptions
- **throw** and **try** are expressions
  - You can return or assign the result to a variable
- **@Throws** for Java interop

Exercise: handling exceptions

```
public class FileReader {

    public byte[] readBytes(String filename) throws IOException {
        return Files.readAllBytes(Paths.get(filename));
    }

    public static void main(String[] args) {
        // TODO catch the IOException
        new FileReader ().readBytes("nonexisting");
    }
}
```

TODO

- Create this Java class
- Run it

Exercise: handling exceptions

```
class FileReaderKotlin {

    fun readBytes(filename: String) = File(filename).readBytes()
}

public class FileReader {

    public static void main(String[] args) {
        // TODO catch the IOException
        new FileReaderKotlin().readBytes("nonexisting");
    }
}
```

TODO

- Create the FileReaderKotlin
- Call FileReaderKotlin from FileReader
- Catch the IOException

Extension functions

- Video: 10. Extension functions

Extension functions

- An extension function extends a Class
- The type is called the receiver
- Inside extension we can access the receiver by **this** but can be omitted
- Needs be imported for usage
- From Java a regular static function with receiver as first argument
- Typically defined as top-level functions in separate file

Exercise: create an extension function: *countUpperCase*

- Count the uppercase characters in a String

```
fun main() {
    println("abCdE".countUpperCase())
}
```

Std lib extensions

- Video: 11. Std lib extensions

Std lib extension

- Kotlin std lib = Java standard library + extensions
- Spring Kotlin support = Spring Java libraries + extensions
- No Kotlin SDK but small runtime jar

Calling extensions

- Video: 12. Calling extensions

Calling extensions

- Same semantics as with static functions in Java
- Member functions always win from extensions when shadowing

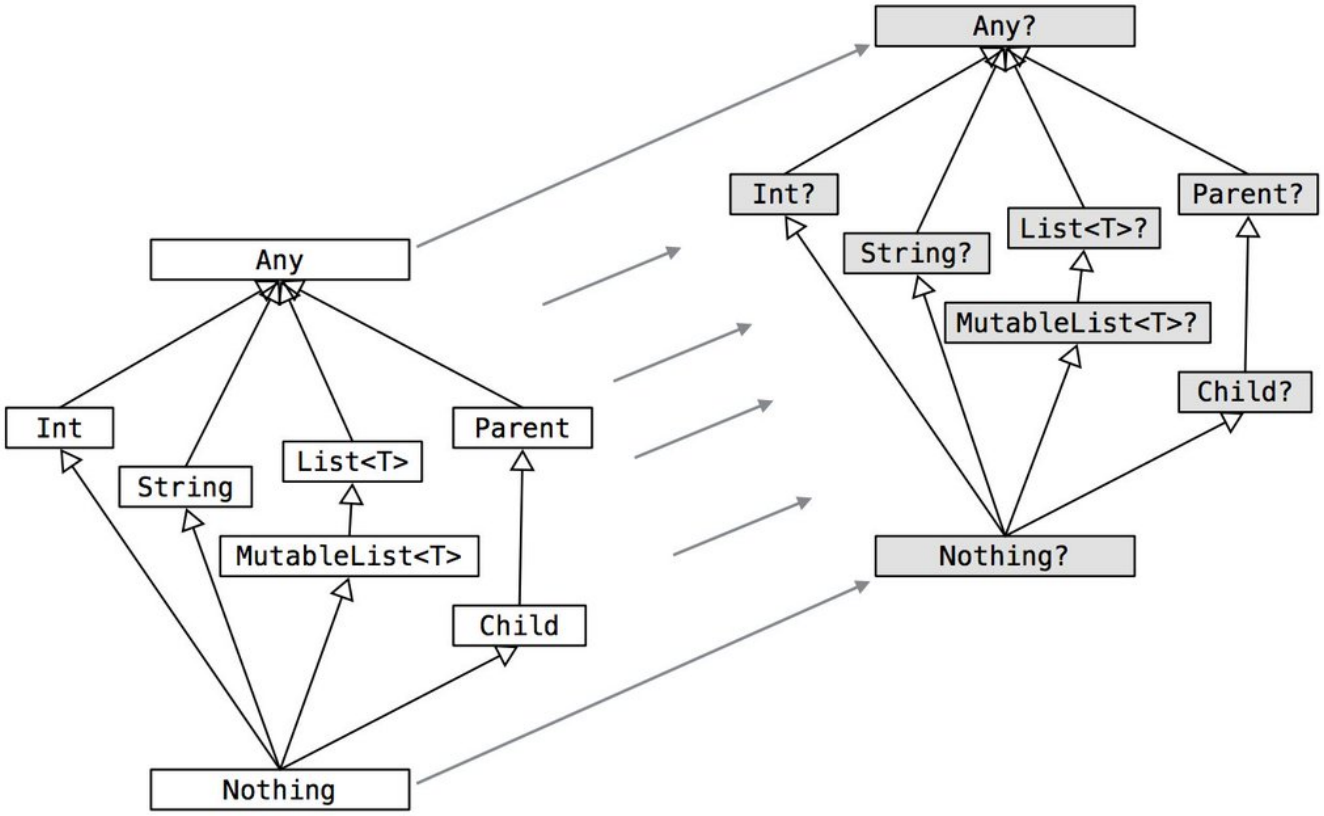
Nullably types

- Video: 14. Nullable types

Nullable types

- Nullability should be compile time, not a runtime problem!
- Types can be either nullable or non nullable
- Smart cast applied by compiler to non nullable type
- Safe access operation **?.**
- Default value using elvis operator **?:**
- Not null assertion operator **!!**
- Be aware of operator precedence

Type hierarchy



Nullable types under the hood

- Video: 15. Nullable types under the hood

Nullable types under the hood

- Kotlin adds @Nullable and @NotNull annotations
- Compiler checks annotations during compilation
- Nullable types != Optional
- No overhead (like Optional wrappers)
- Nullability also reflected in generics

Lambdas

- Video: 18. Lambdas

Lambdas

- Lambda is an **anonymous function** that can be **used as an expression**
- Defined in curly braces **{ }**
- Syntax: **{** arguments **->** body **}**
  - Example for plus: **{** a: Int, b: Int **->** a + b **}**
- When used as last argument in function can me moved outside of the function
- When used as only argument in function empty parenthesis can be omitted

```
listOf("a", "b").filter({ item -> item == "a" })

listOf("a", "b").filter { item -> item == "a" }

listOf("a", "b").filter { it == "a" }
```

Common operations on collections

- Video: 19. Common operations on collections

Common operations on collections

- Commonly used: filter, map, any, none, all, first, firstOrNull, find, count, etc.
- Lambda syntax used
- Alternative for Java stream operations (non lazy)

Function types

- Video: 20. Function types

Function types

- Function type is a stored lambda in a variable
- Type declaration: () -> T
  - () defines input params
  - -> T the return type
- Can be used with SAM (single abstract method) classes
  - val runnable = *Runnable* { println(42) }
  - val rowMapper = *RowMapper<Person>* { rs, index -> Person(rs.getString("name")) }

Exercise: write a lambda

- Define a function type for a lambda that adds two numbers and returns the result
- Create the lambda expression
- Write a main and Invoke it

- Example: val x: (p1, p2) -> T = **{** p1, p2 -> body **}**
  - () defines input params
  - -> T the return type
  - **{}** defines the lambda expression

Object oriented programming in Kotlin

- Video: 23. OOP in Kotlin

Object oriented programming in Kotlin

- Default visibility public final
  - This breaks the Spring cglib enhancement!
- open makes something non-final
- internal for module accessibility
- Package structure different
  - Multiple classes allowed in a single file
  - Package name does not need to represent directory structure

Constructors and inheritance

- Video: 24. Constructors, Inheritance syntax

Constructors and inheritance

- No **new** keyword
- init {} represents the constructor body
- Primary and secundairy constructors
- Constructor and annotations!!

```
@Service
class OtherService

@Service
class MyService @Autowired constructor(otherService: OtherService)
```

Class modifiers

- Video: 25. Class modifiers

Class modifiers

- Data keyword adds equals, hashCode, toString, copy functions to a class

Objects, object expressions and companion objects

- Video: 27. Objects, object expressions and companion objects

Objects, object expressions and companion objects

- Object is same as a singleton pattern in Java
  - Special INSTANCE variable available
- Object expression: object : SomeClassToOverride() {}
- Kotlin does not have static methods, @JvmStatic for interop
- Classes van have companion object
  - Companion object can implement an interface
  - Accessible via Companion instance

Generics

- Video: 29. Generics

Generics

- Result can be nullable: T?
- Makes the input type explicit non nullable: fun <T : Any>
- Use @JvmName on function with different generified types

Library functions looking like built-in constructs

- Video: 30. Library functions looking like built-in constructs

Library functions looking like built-in constructs

- run, let, takeIf, takeUnless

Lambda with receivers

- Video: 31. Lambda with receiver

Lambda with receivers

- Also called extension lambdas
- Ideal for writing DSLs
- Removes boilder-plate like instantiating objects

Example: build-in std library functions

```
val sb = StringBuilder()

// public inline fun <T, R> T.run(block: T.() -> R): R
sb.run {
    append("Hello,")
    append("Kotlin!")
    toString()
}

// public inline fun <T, R> with(receiver: T, block: T.() -> R): R
with(sb) {
    append("Hello,")
    append("Kotlin!")
    toString()
}
```

Lambda with receivers (builder)

```
buildString {
    append("Hello,")
    append("Kotlin!")
}

fun buildString(function: StringBuilder.() -> String): String {
    val builder = StringBuilder()
    builder.function()
    return builder.toString()
}
```

Exercise

- Build your own lambda with receiver for creating an URL

Exercise: lambda with receiver

```
class URL(val host: String, val port: Int) {

    class Builder {
        var host: String = "localhost"
        var port: Int = 80
        fun build(): URL = URL(host, port)
    }
}

fun main() {
    val builder = URL.Builder()
    builder.host = "www.nu.nl"
    builder.port = 8080
    builder.build()
}
```

```
fun url(function: URL.Builder.() -> Unit): URL {



}

fun main() {
    url {
        host = "www.nu.nl"
        port = 8080
    }
}
```

More useful library functions

- Video: 31. More useful library functions

More useful library functions

- Build in functions: with, run, apply,
- Using lambda with receivers

Spring Kotlin support

- Spring Java libraries + annotations + extensions
- Added @NotNull and @Nullable to Spring code base
- Bean Definition DSL
- Spring Boot Coroutines support

Spring Bootique

- Use what you have learned and apply it to an existing Spring Boot Java project

Thats all!

- Next steps
  - You will receive the slides
  - And an evaluation form (anonymous)

- Thanks for attending!!