JHUG Meetup
February 2018

A short introduction to the Kotlin language

for **Java** developers

Antonis Lilis
Mobile Engineer @ ADVANTAGE
FINANCIAL SYSTEMS EXPERTS

Some History

- **2011:** JetBrains **unveiled** Project Kotlin, a new language for the JVM
- **2012:** JetBrains **open sourced** the project under the Apache 2 license
- **2016:** Kotlin **v1.0** is released
- **2017:** Google announced **first-class support for Kotlin on Android**
- Kotlin is technically 7, but in reality **2 years old**

Trivia: *The name comes from a small island in the Baltic Sea, near St.Petersburg. The language team decided to* ***name it after an island just like Java*** *(though Java was perhaps named after the coffee)*

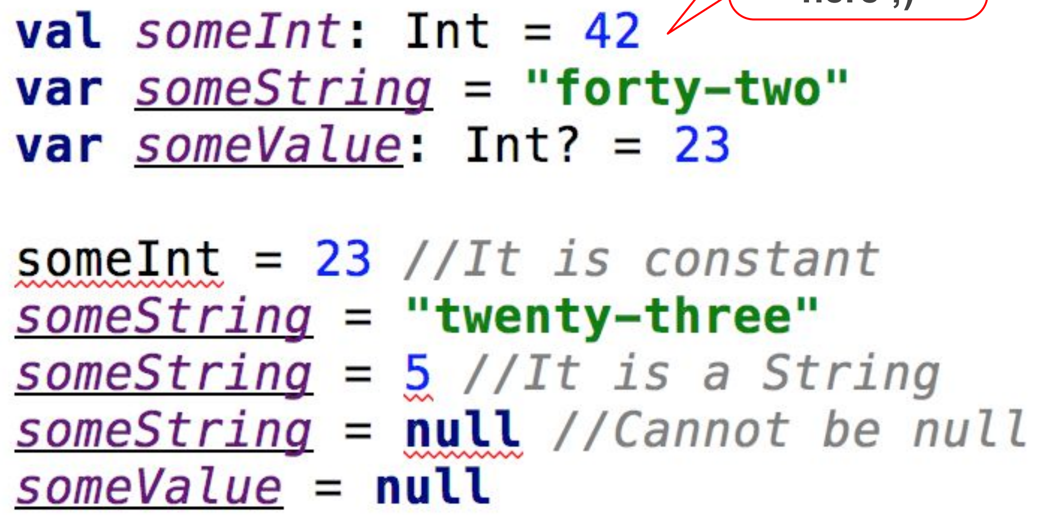
The Kotlin Language

- **Statically Typed**
  - Type validation at compile time
- Supports **Type Inference**
  - Type automatically determined from the context
- Both **Object Oriented** and **Functional**
- **First-class functions**
  - You can store them in variables, pass them as parameters, or return them from other functions
- Was designed with **Java Interoperability** in mind

Constants and Variables

- **val** (from value)
  - **Immutable** reference
- **var** (from variable)
  - **Mutable** reference
- **Nullable** Types
  - Defined **Explicitly**

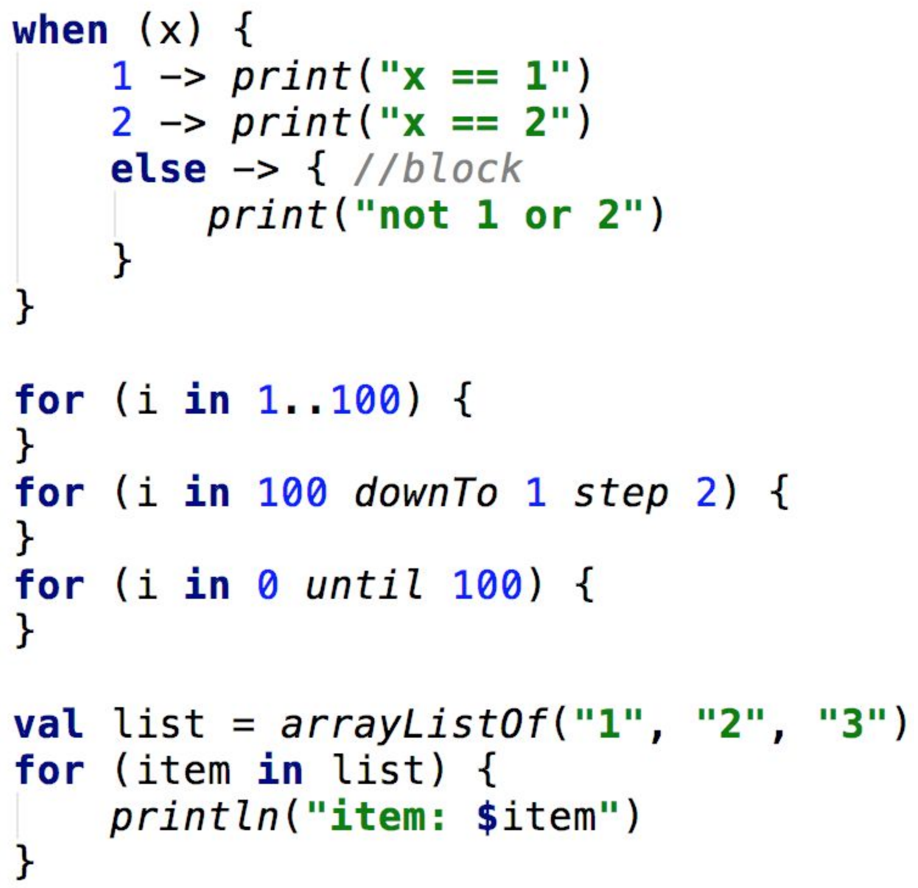
No semicolon here ;)



```
val someInt: Int = 42
var someString = "forty-two"
var someValue: Int? = 23

someInt = 23 //It is constant
someString = "twenty-three"
someString = 5 //It is a String
someString = null //Cannot be null
someValue = null
```

Control Flow

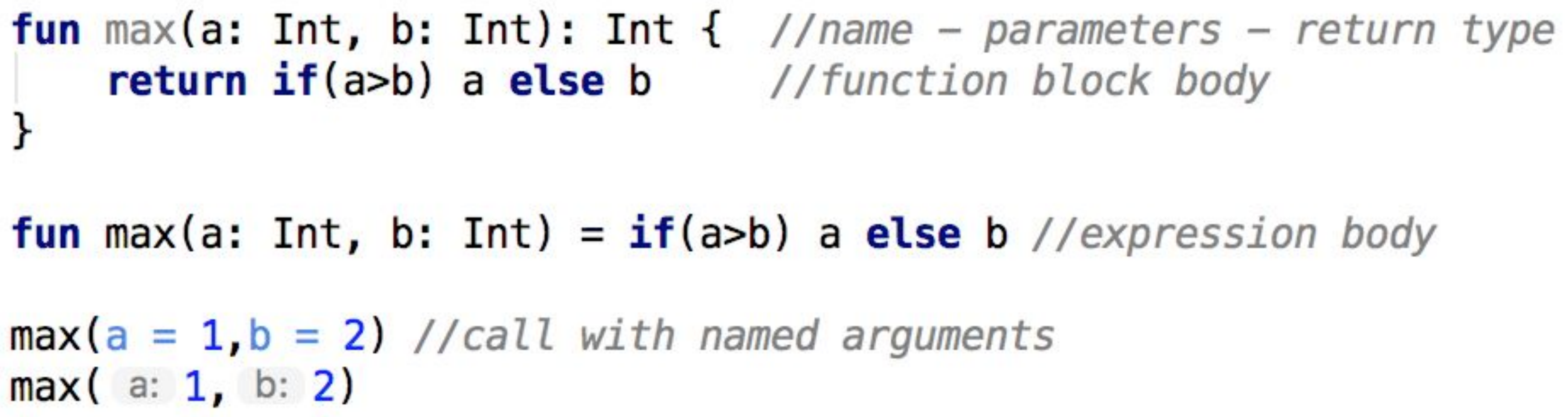
- Classic loops:
  - **if**
  - **for**
  - **while** / **do-while**
- **when**
  - Replaces the switch operator
  - No breaks, no errors



```
when (x) {
    1 -> print("x == 1")
    2 -> print("x == 2")
    else -> { //block
        print("not 1 or 2")
    }
}

for (i in 1..100) {
}
for (i in 100 downTo 1 step 2) {
}
for (i in 0 until 100) {
}

val list = arrayListOf("1", "2", "3")
for (item in list) {
    println("item: $item")
}
```

Functions

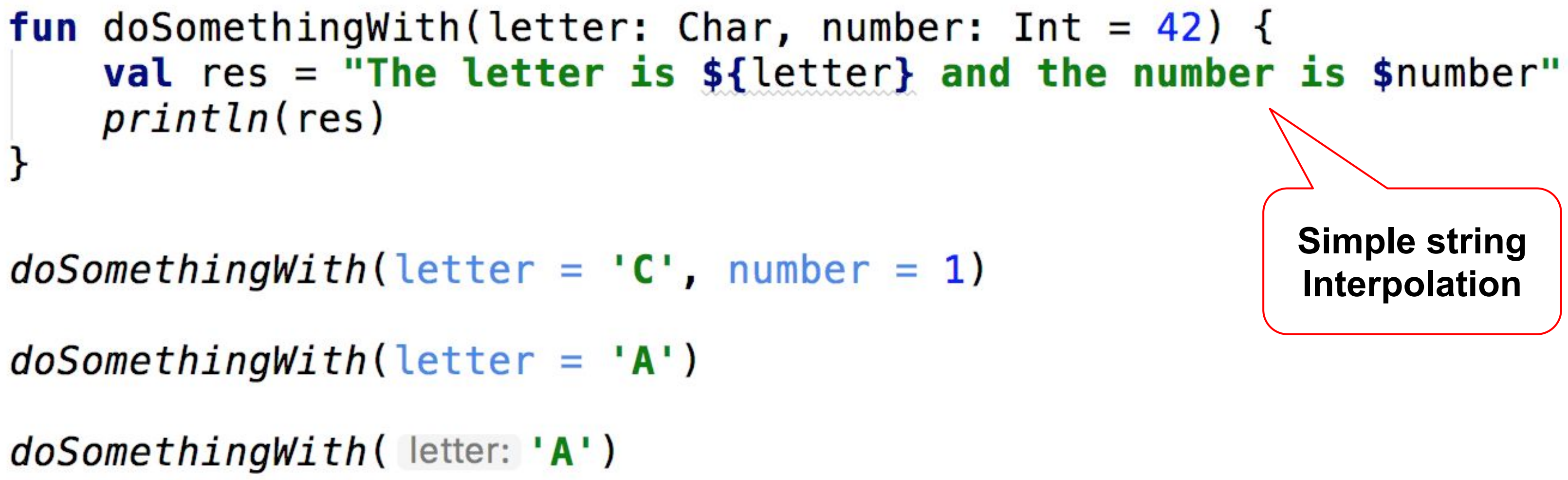
- **Named** arguments
- Can be declared at the **top level** of a file (without belonging to a class)
- Can be **Nested**
- Can have a **block or expression body**



```
fun max(a: Int, b: Int): Int {  //name - parameters - return type
    return if(a>b) a else b     //function block body
}

fun max(a: Int, b: Int) = if(a>b) a else b //expression body

max(a = 1,b = 2) //call with named arguments
max( a: 1, b: 2)
```

Functions

- **Default** parameter values
  - Avoids method overloading and boilerplate code



```kotlin
fun doSomethingWith(letter: Char, number: Int = 42) {
    val res = "The letter is ${letter} and the number is $number"
    println(res)
}

doSomethingWith(letter = 'C', number = 1)

doSomethingWith(letter = 'A')

doSomethingWith( letter: 'A')
```

Simple string Interpolation

Classes

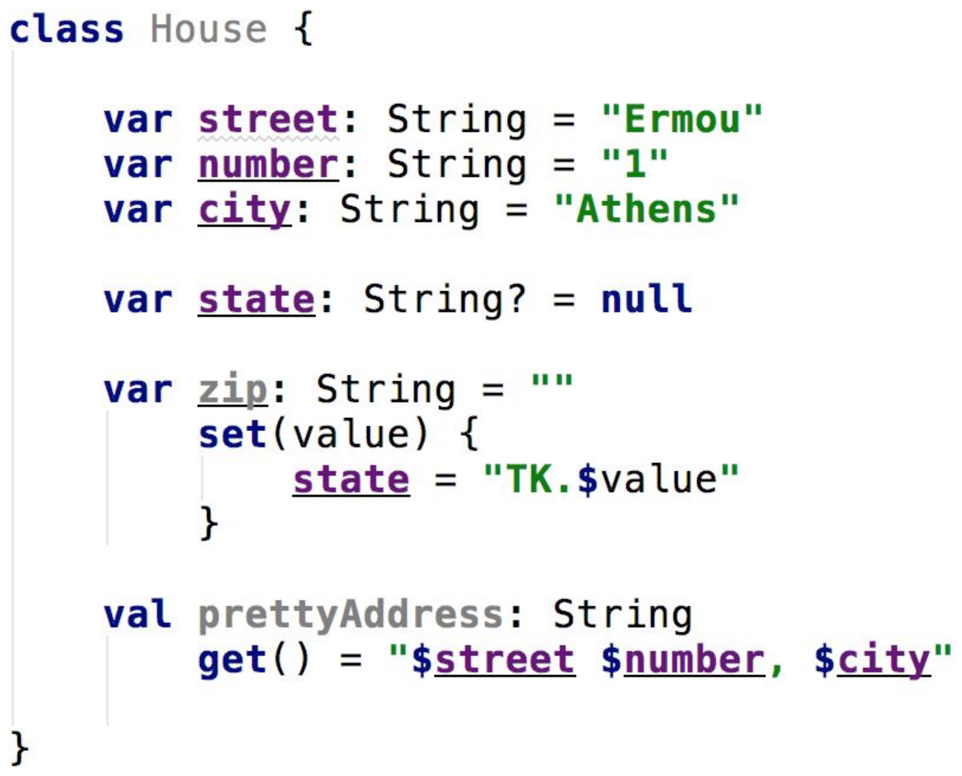
```
class MyView : View {
    constructor(ctx: Context): super(ctx) {
        //Initialization stuff
    }
    //...
}

class MyViewShort(ctx: Context) : View(ctx) {
    //...
}

class Car(val brand: String, val isUsed: Boolean = false)

val car = Car( brand: "Ford")

data class Bike(val brand: String, val isUsed: Boolean = false)
```

"**Any**" is the analogue of java **Object**: a superclass of all classes

data classes: **autogenerated** implementations of universal **methods** (equals, hashCode etc)

Properties

- **first-class** language feature
- combination of the **field** and its **accessors**



```
class House {

    var street: String = "Ermou"
    var number: String = "1"
    var city: String = "Athens"

    var state: String? = null

    var zip: String = ""
        set(value) {
            state = "TK.$value"
        }

    val prettyAddress: String
        get() = "$street $number, $city"

}
```

Modifiers

- **Access** modifiers
  - **final** (default)
  - **open**
  - **abstract**
- **Visibility** modifiers
  - **public** (default)
  - **internal**
  - **protected**
  - **private**

"Design and document for inheritance or else prohibit it"

Joshua J. Bloch, Effective Java

ps. Lukas Lechner has written a series of articles on "How Effective Java influenced Kotlin" (http://lukle.at)

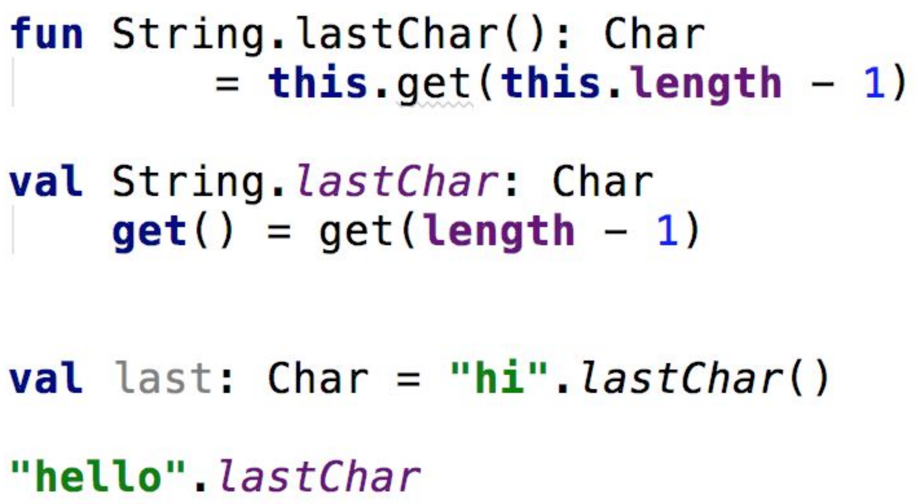
No static keyword

- **Top-level** functions and properties
  (e.g. for utility classes)
- **Companion objects**
- The **object keyword**:
  declaring a class and creating an instance
  combined (**Singleton**)

```
class Foo {
    companion object {
        fun bar() {
            //...
        }
    }
}

object Singleton {
    fun doSomething() {
        //..
    }
}


Foo.bar()

Singleton.doSomething()
```

Extensions

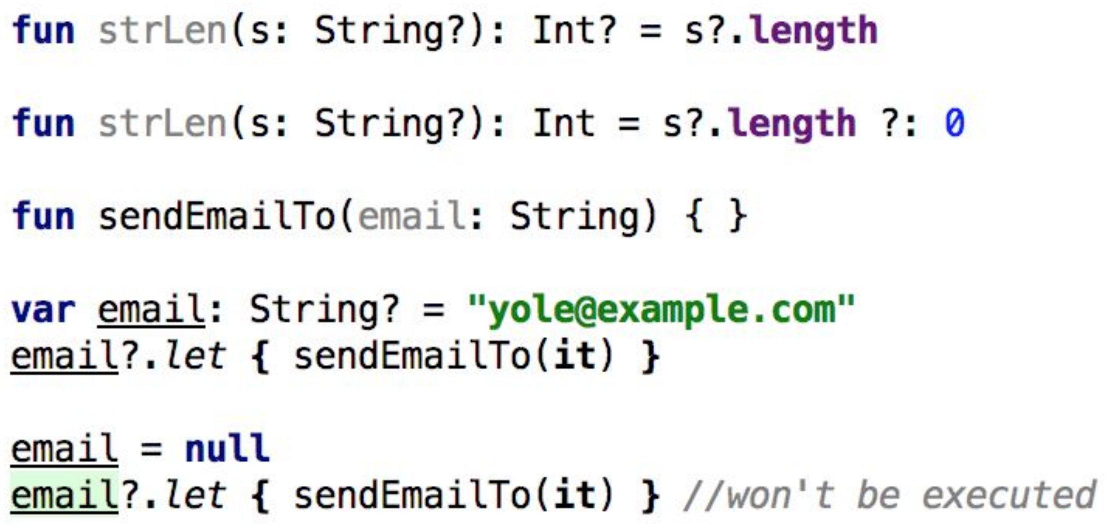
- Enable **adding methods and properties** to other people’s classes
  - Of Course without access to private or protected members of the class



```kotlin
fun String.lastChar(): Char
        = this.get(this.length - 1)

val String.lastChar: Char
    get() = get(length - 1)


val last: Char = "hi".lastChar()

"hello".lastChar
```

Null Checks

- **Safe-call** operator **?.**
- **Elvis** operator **?:**
- The **let** function



```kotlin
fun strLen(s: String?): Int? = s?.length

fun strLen(s: String?): Int = s?.length ?: 0

fun sendEmailTo(email: String) { }

var email: String? = "yole@example.com"
email?.let { sendEmailTo(it) }

email = null
email?.let { sendEmailTo(it) } //won't be executed
```

"I call it my billion-dollar mistake.
It was the invention of the null reference in 1965"
Tony Hoare

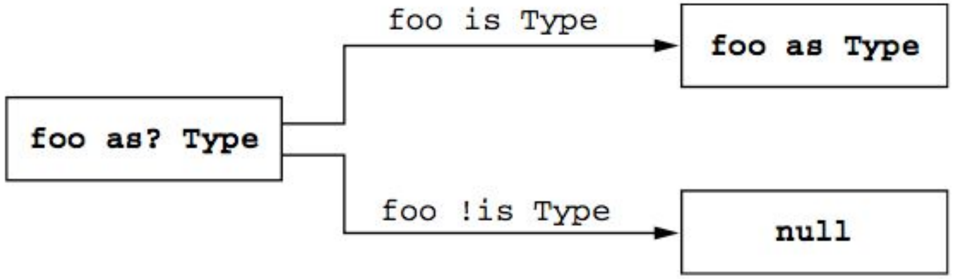
Not-null assertion operator !!

```kotlin
fun rootOfAllEvils(s: String?) {
    val sNotNull: String = s!!
    println(sNotNull.length)
}
```

Safe Casting

- **Safe cast** operator **as?**
- **Smart cast**
  - combining type checks and casts

```
var myview: MyView? = MyView(ctx)

val view = myview as? View

if (view is MyView) {
    view.bar()
}
```



Collections

- Kotlin **enhances** the **Java** collection classes (**List, Set, Map**)

```
class Car(val brand: String, val age: Int, val horsePower: Int)

val fleet = listOf(
        Car( brand: "Ford",  age: 1,  horsePower: 100),
        Car( brand: "Mazda",  age: 2,  horsePower: 120),
        Car( brand: "Opel",  age: 2,  horsePower: 95))

fleet.maxBy { it.horsePower }

fleet.filter { it.age == 2 }

fleet.filter { it.age == 2 }.maxBy { it.horsePower }

fleet.forEach { print("brand: $it.brand") }
```

Chained Calls

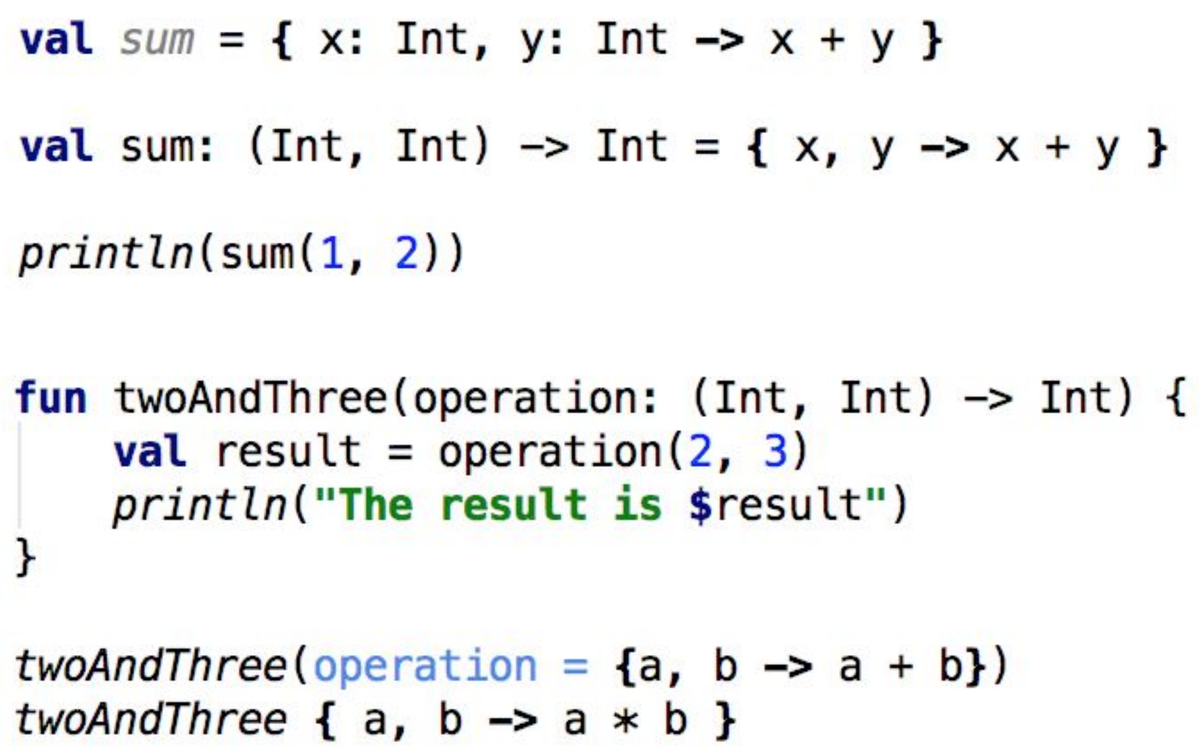
Delegation

- **Composition over Inheritance** design pattern
- **Native support** for delegation (**implicit** delegation)
- **Zero Boilerplate** code
- Supports both **Class Delegation** and **Delegated Properties**

```
interface Nameable {
    var name: String
}

class Ford : Nameable {
    override var name = "Ford"
}

class Car(name: Nameable)
    : Nameable by name


val car = Car(Ford())
print(car.name) //Ford
```

Class Car **inherits from an interface** Nameable and **delegates** all of its public **methods to** a delegate **object** defined with the **by keyword**

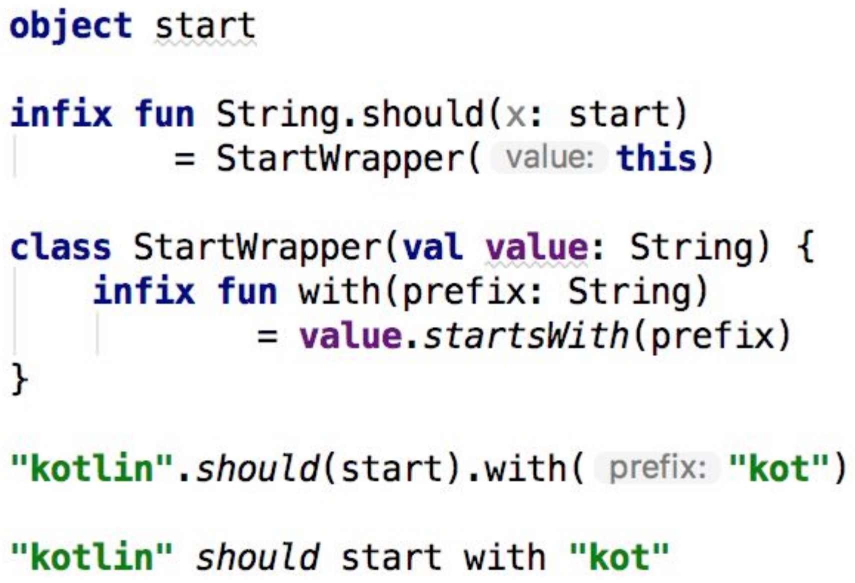
Lamdas and Higher Order Functions



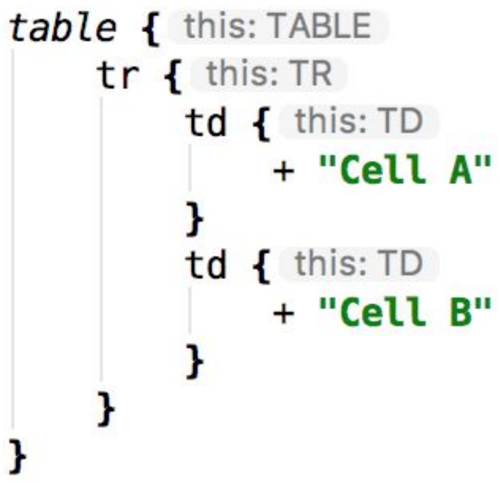
```
val sum = { x: Int, y: Int -> x + y }

val sum: (Int, Int) -> Int = { x, y -> x + y }

println(sum(1, 2))


fun twoAndThree(operation: (Int, Int) -> Int) {
    val result = operation(2, 3)
    println("The result is $result")
}

twoAndThree(operation = {a, b -> a + b})
twoAndThree { a, b -> a * b }
```

Domain-specific language construction

- Kotlin provides mechanisms for creating internal DSLs that use exactly the **same syntax** as all language features and are fully **statically typed**



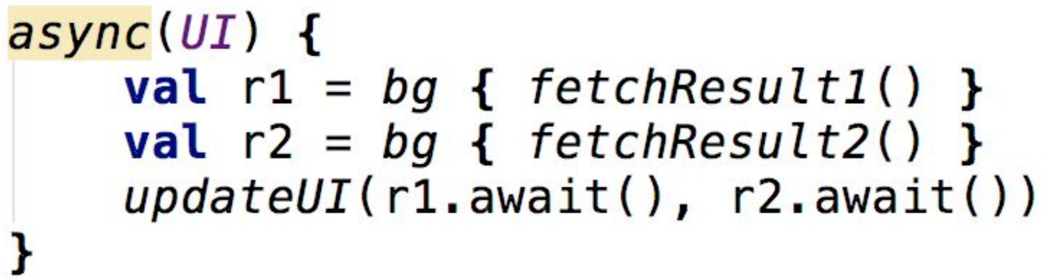
```
object start

infix fun String.should(x: start)
        = StartWrapper( value: this)

class StartWrapper(val value: String) {
    infix fun with(prefix: String)
            = value.startsWith(prefix)
}

"kotlin".should(start).with( prefix: "kot")

"kotlin" should start with "kot"
```



```
table { this: TABLE
    tr { this: TR
        td { this: TD
            + "Cell A"
        }
        td { this: TD
            + "Cell B"
        }
    }
}
```

Coroutines

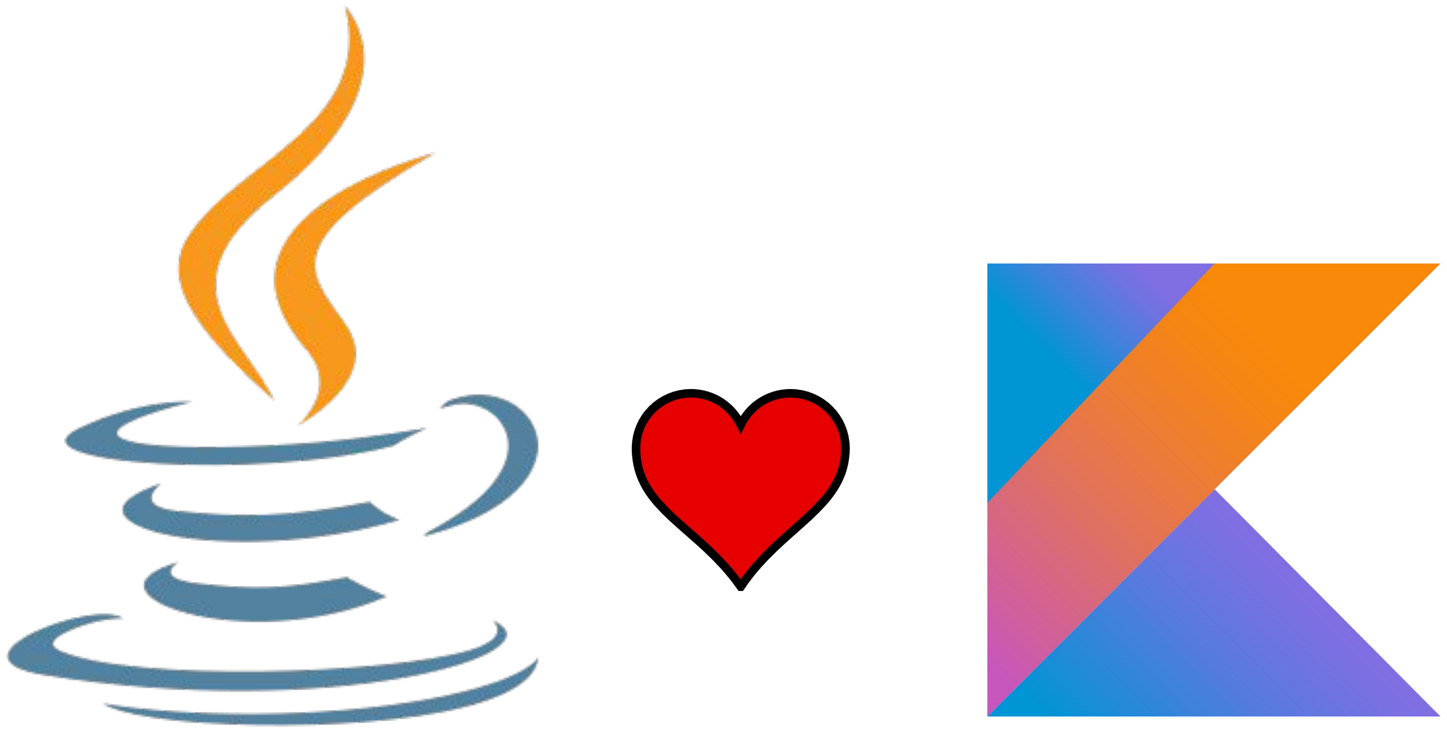
- Introduced in Kotlin **1.1** (March 2017)
- A way to write **asynchronous code sequentially**
- **Multithreading** in a way that is easily debuggable and maintainable
- Based on the idea of **suspending function execution**
- More **lightweight** and efficient than threads



```
async(UI) {
    val r1 = bg { fetchResult1() }
    val r2 = bg { fetchResult2() }
    updateUI(r1.await(), r2.await())
}
```

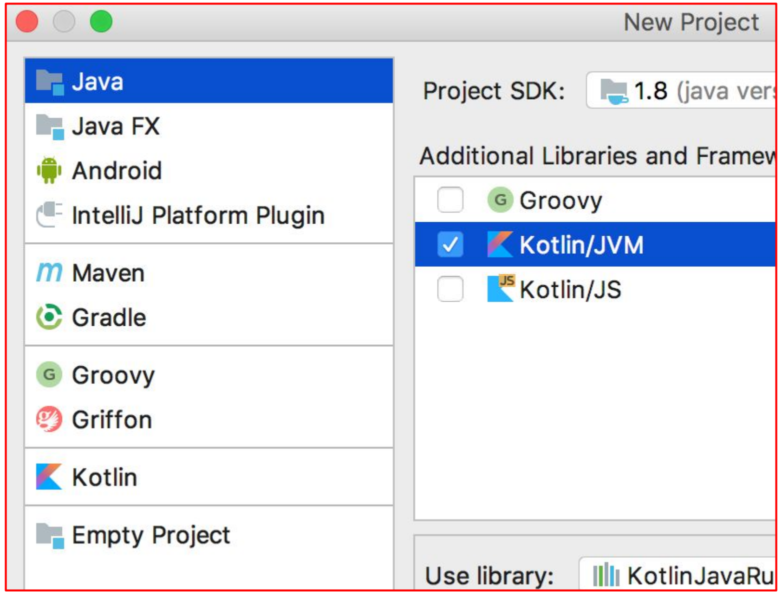
Source Code Layout

- **Packages** (similar to that in Java)
- **Multiple classes** can fit **in the same file**
- You can choose **any name for files** (not restricted to class name)
- The **import** keyword is not restricted to importing classes
    - Top-level functions and properties can be imported
- Kotlin does **not** impose any **restrictions** on the layout of source files on disk
- Good practice to **follow Java's directory layout**
    - Especially if mixed with java



Kotlin IDEs

- You can write Kotlin next to Java in your favorite IDE
- Kotlin works with **JDK 1.6+**
- IntelliJ **IDEA** (and **Android Studio**) support Kotlin **out of the Box**
- There is a **plugin for Eclipse** too



Project SDK: 1.8 (java vers

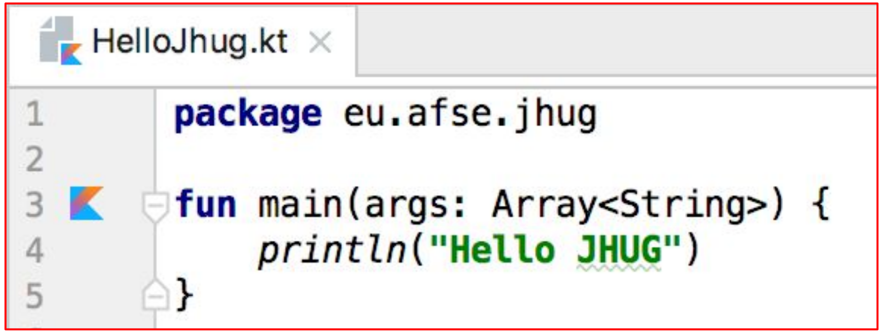
Additional Libraries and Framew

- [ ] Groovy
- [x] Kotlin/JVM
- [ ] Kotlin/JS

Use library: KotlinJavaRu

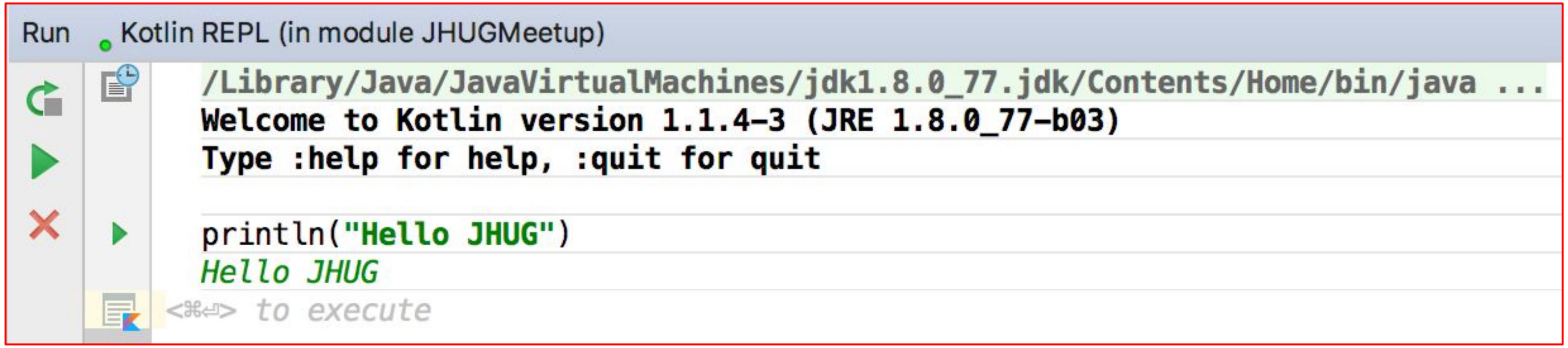
Hello World

- Kotlin files have **.kt extension**
- You can also try your code in **REPL** (Read-Eval-Print-Loop)



HelloJhug.kt

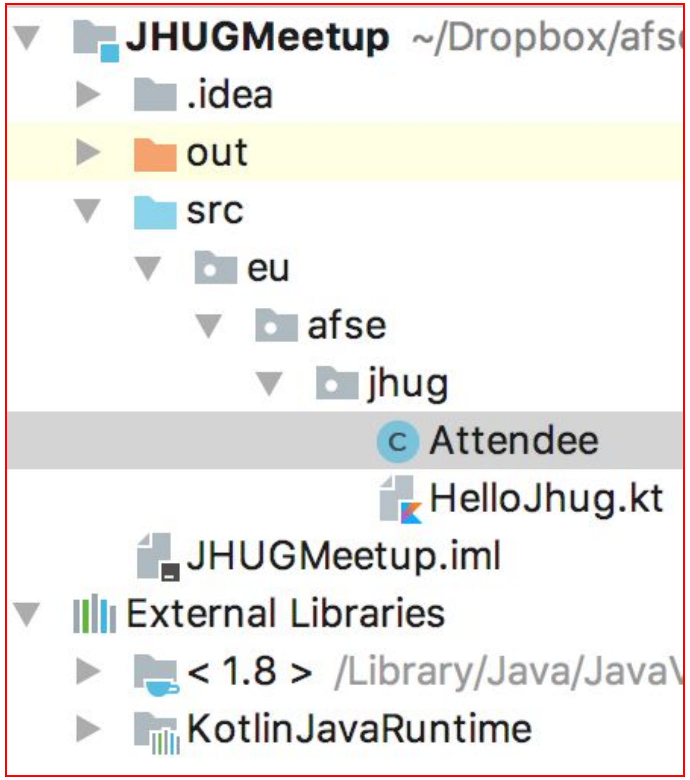
```kotlin
package eu.afse.jhug

fun main(args: Array<String>) {
    println("Hello JHUG")
}
```



Run   Kotlin REPL (in module JHUGMeetup)

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_77.jdk/Contents/Home/bin/java ...
Welcome to Kotlin version 1.1.4-3 (JRE 1.8.0_77-b03)
Type :help for help, :quit for quit

println("Hello JHUG")
Hello JHUG
<⌘⏎> to execute
```

Let's Mix with some Java

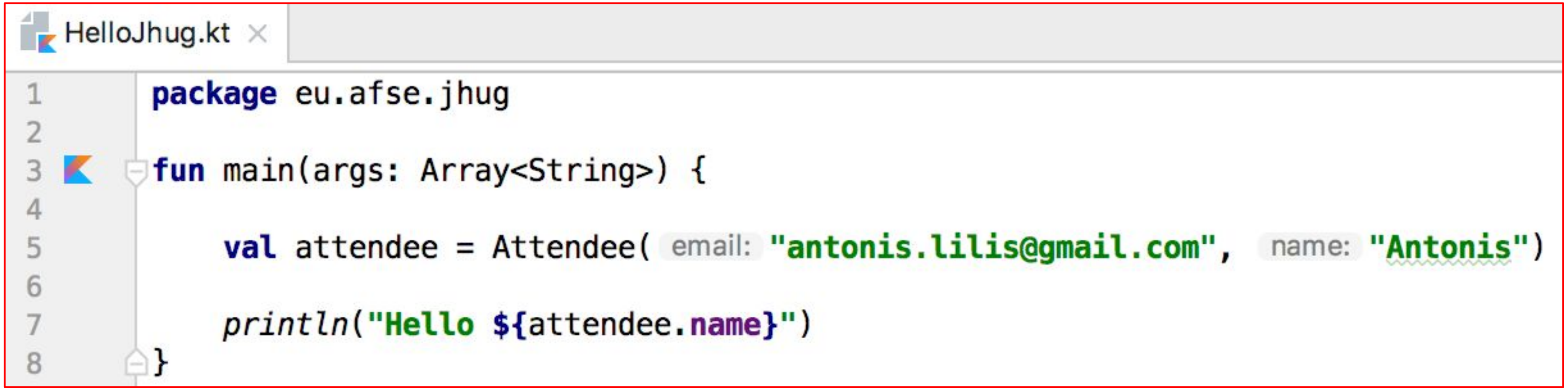


```
JHUGMeetup ~/Dropbox/afs
  .idea
  out
  src
    eu
      afse
        jhug
          Attendee
          HelloJhug.kt
  JHUGMeetup.iml
External Libraries
  < 1.8 > /Library/Java/JavaV
  KotlinJavaRuntime
```

```java
package eu.afse.jhug;

public class Attendee {

    private String email;
    private String name;

    public Attendee(String email, String name) {
        this.email = email;
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

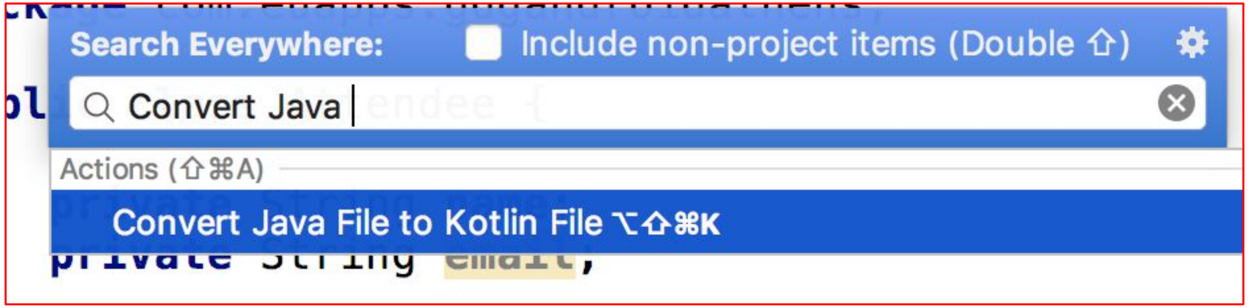
Java from Kotlin

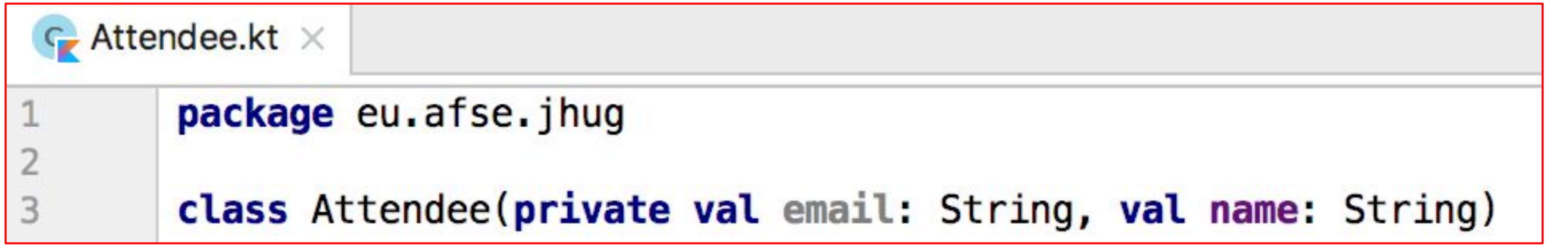
- You can call Java code from you Kotlin files transparently


HelloJhug.kt

```kotlin
package eu.afse.jhug

fun main(args: Array<String>) {

    val attendee = Attendee( email: "antonis.lilis@gmail.com",  name: "Antonis")

    println("Hello ${attendee.name}")
}
```

Convert Java to Kotlin


Search Everywhere:
Include non-project items (Double ⇧)
Convert Java
Actions (⇧⌘A)
Convert Java File to Kotlin File ⌥⇧⌘K



```
package eu.afse.jhug

class Attendee(private val email: String, val name: String)
```

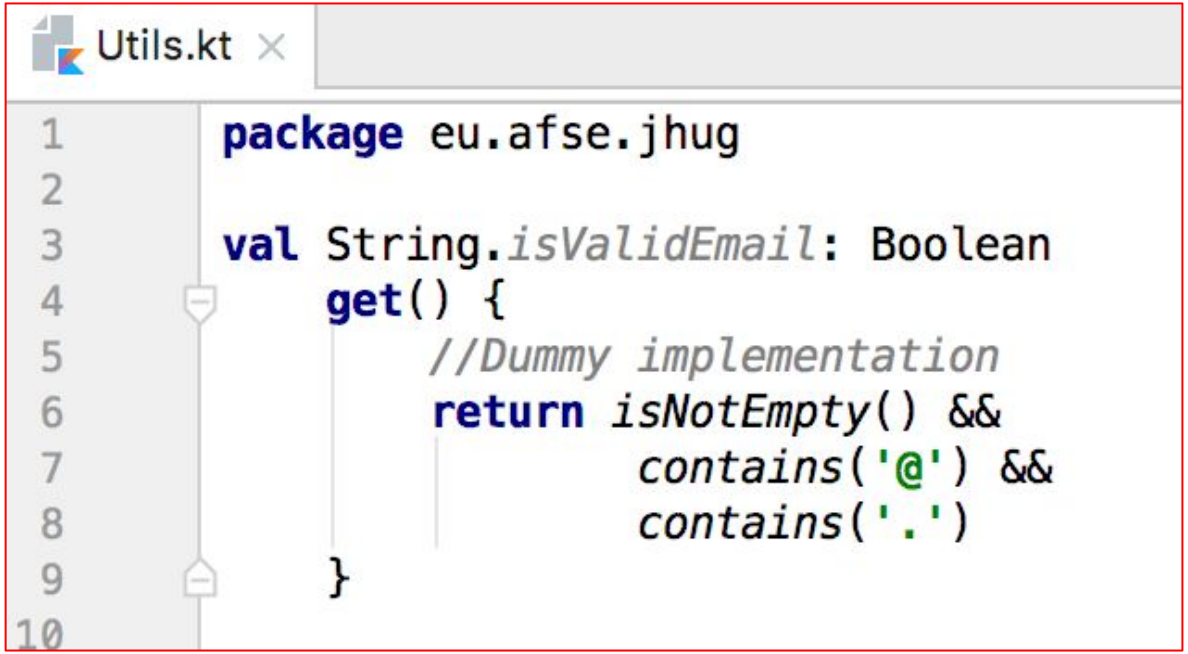
Code size

- According to Jetbrains converting a Java application to Kotlin is expected to **reduce the line count by 40%**

  - **More concise** language (eg. Kotlin data classes can replace 50 line classes with a single line)
  - The Kotlin standard library enhances existing Java classes with **extensions** that trivialize common usage (eg. collections)
  - Kotlin allows you to extract **more re-usable** patterns than what Java allows

Utility

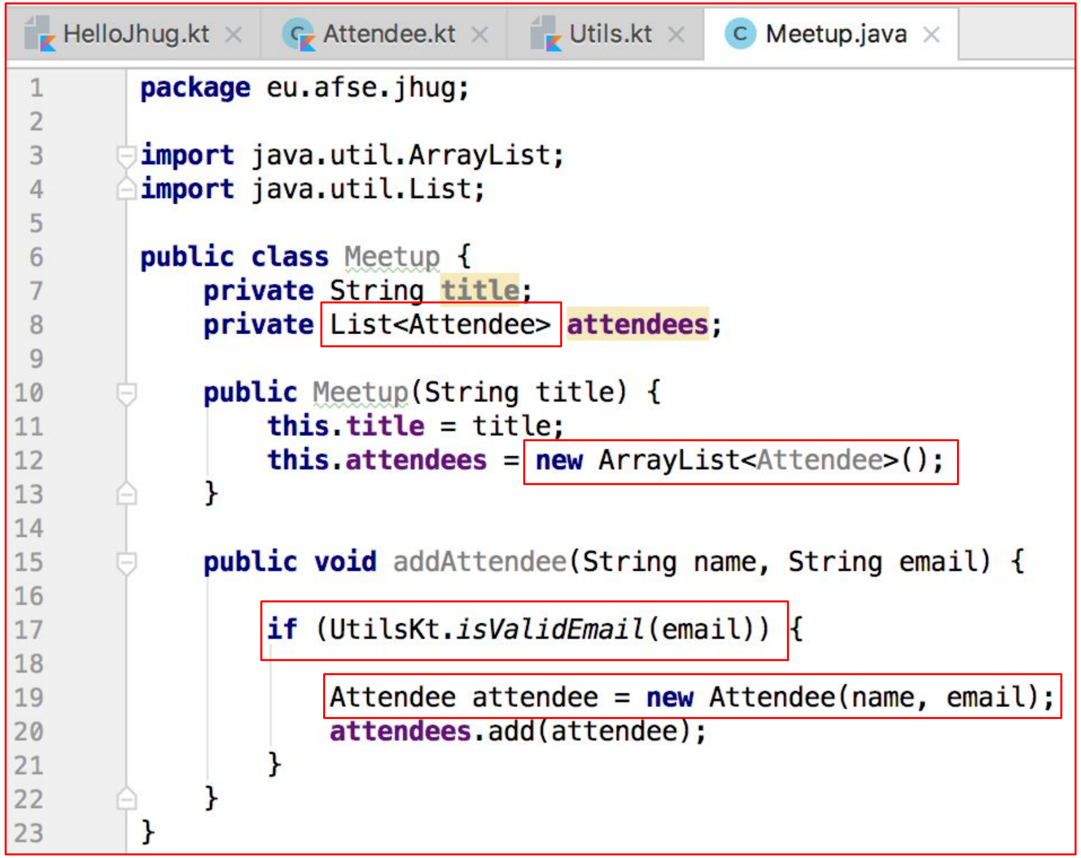
- **Top-level** computed **property**
- String **extension**


Utils.kt

```kotlin
package eu.afse.jhug

val String.isValidEmail: Boolean
    get() {
        //Dummy implementation
        return isNotEmpty() &&
                contains('@') &&
                contains('.')
    }

```

Kotlin from Java

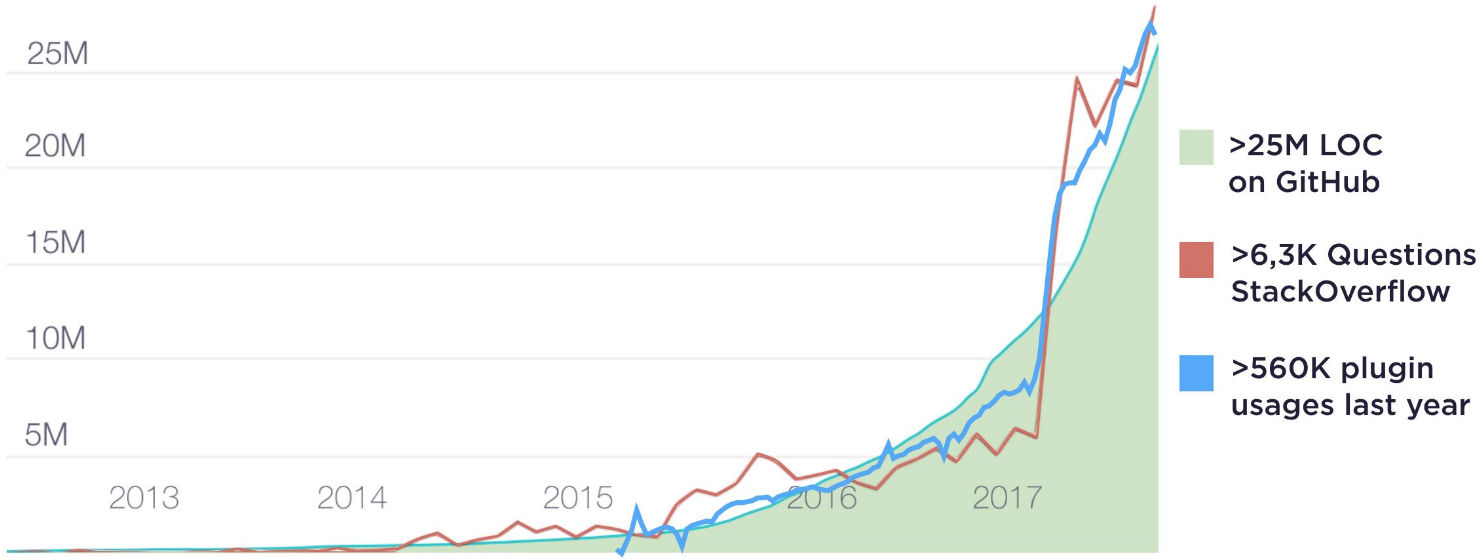
- In most cases the integration is **Seamless**
- Some **Kotlin features do not exist in Java** (e.g. top level functions or properties)
- In such cases **conventions** are used
- In our case a **static class is generated** for the **top-level** declarations



```java
package eu.afse.jhug;

import java.util.ArrayList;
import java.util.List;

public class Meetup {
    private String title;
    private List<Attendee> attendees;

    public Meetup(String title) {
        this.title = title;
        this.attendees = new ArrayList<Attendee>();
    }

    public void addAttendee(String name, String email) {

        if (UtilsKt.isValidEmail(email)) {

            Attendee attendee = new Attendee(name, email);
            attendees.add(attendee);
        }
    }
}
```

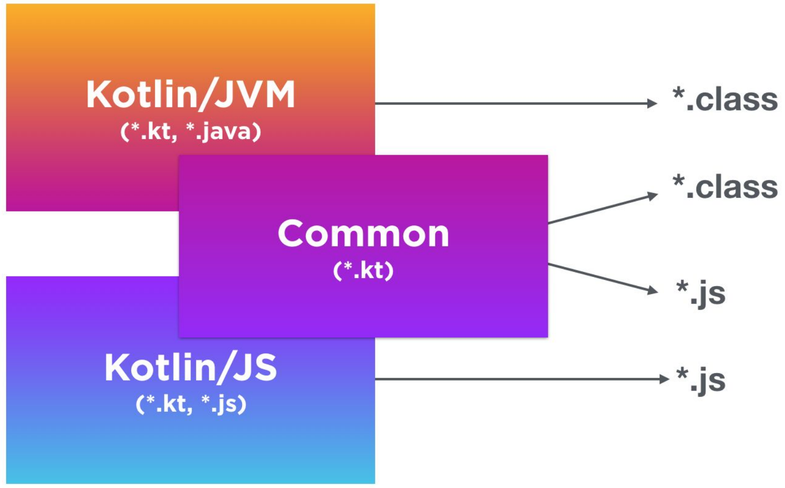
Libraries & Resources

- You can use **Any Java Library** since Java and Kotlin are 100% interoperable
- **Kotlin libraries:** a nice curated list at https://kotlin.link
- Kotlin **popularity is growing** and resources become more abundant



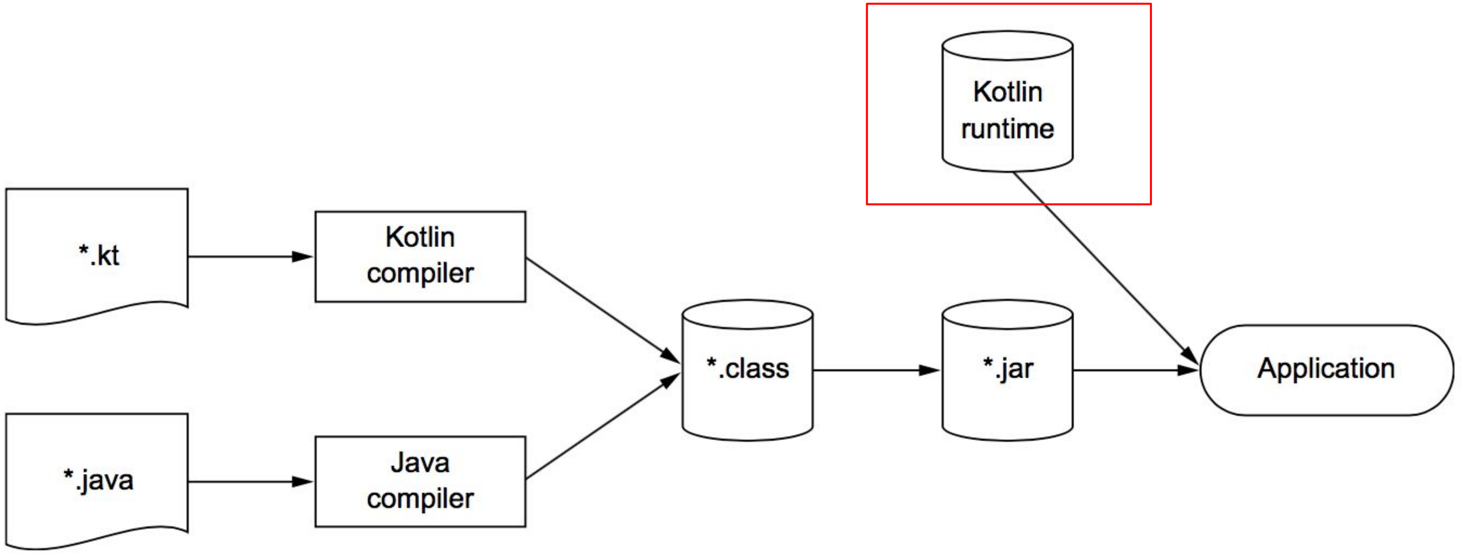
Kotlin also lives outside the JVM

- Kotlin 1.1 (March 2017): officially released the **JavaScript** target, allowing you to compile Kotlin code to JS and to run it in your browser
- Kotlin 1.2 (November 2017): adding the possibility to **reuse code** between the JVM and JavaScript
- Kotlin/**Native** v0.6 (Valentine's Day release 2018): Better support for native targets (e.g. **iOS**, **WebAssembly**, **Windows**)



Any Disadvantages?

- An app built with Kotlin will likely result in a **larger file** package size than one built purely in Java
- The **build time** for Kotlin is a little slower

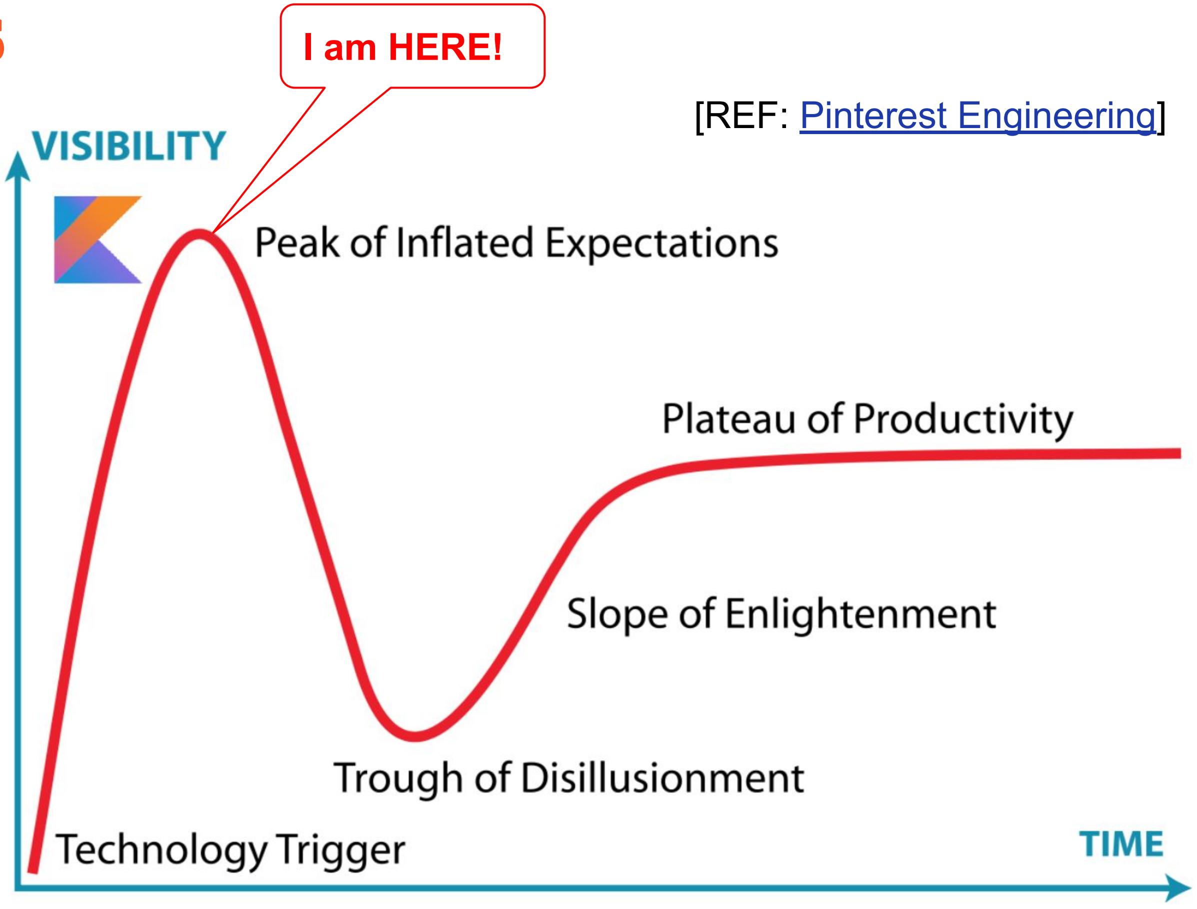


Final Thoughts

- The **learning curve**
  - IMHO comparatively small
- **Not so popular** yet
  - **44th** in TIOBE Index for February but competed with C for language of the year 2017
- Development **Stability**
  - Tools still in **Beta**
  - **Static Analysis** Tools
- **Reversibility**
  - Once you Go Kotlin...

IMHO Kotlin is here to stay

[REF: Pinterest Engineering]



Kotlin on the Hype Curve

Thank you!

Questions?

http://antonis.me/