

# Deep Dive Kotlin : du Hello World au ByteCode



**Emmanuel Vinas**

Expert Android & Java

 @emmanuelvinas

 [emmanuel@monkeypatch.io](mailto:emmanuel@monkeypatch.io)

---



**Igor Laborie**

Expert Java & Web

 @ilaborie

 [igor@monkeypatch.io](mailto:igor@monkeypatch.io)

---

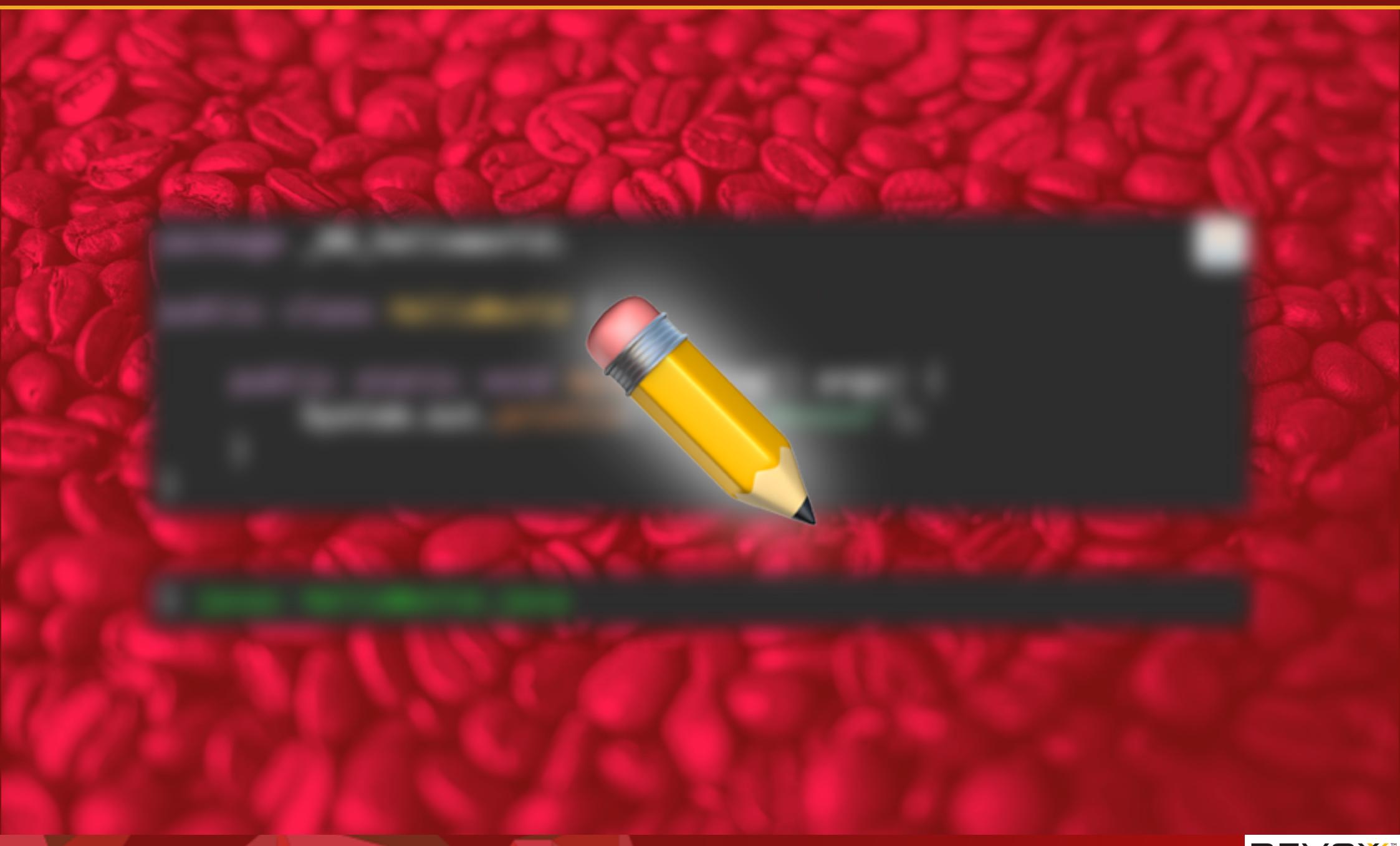


- I. ByteCode Java ?
- II. Introduction Kotlin
- III. Les bases
- IV. null-safety
- V. Les types
- VI. Les fonctions
- VII. Les lambdas
- VIII. Les classes
- IX. Extensions de fonctions
- X. Pause
- XI. ByteCode Android
- XII. Autres structures
- XIII. Les collections
- XIV. Les delegates
- XV. Un peu plus sur les fonctions
- XVI. Conclusion

# ByteCode Java ?

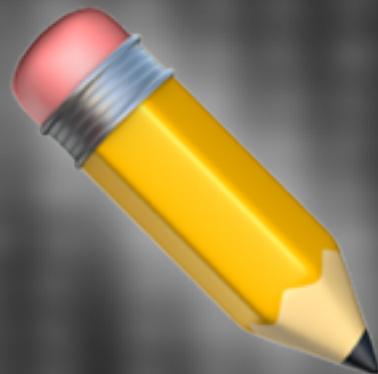
# HelloWorld.java

# 3



# Java ByteCode binary

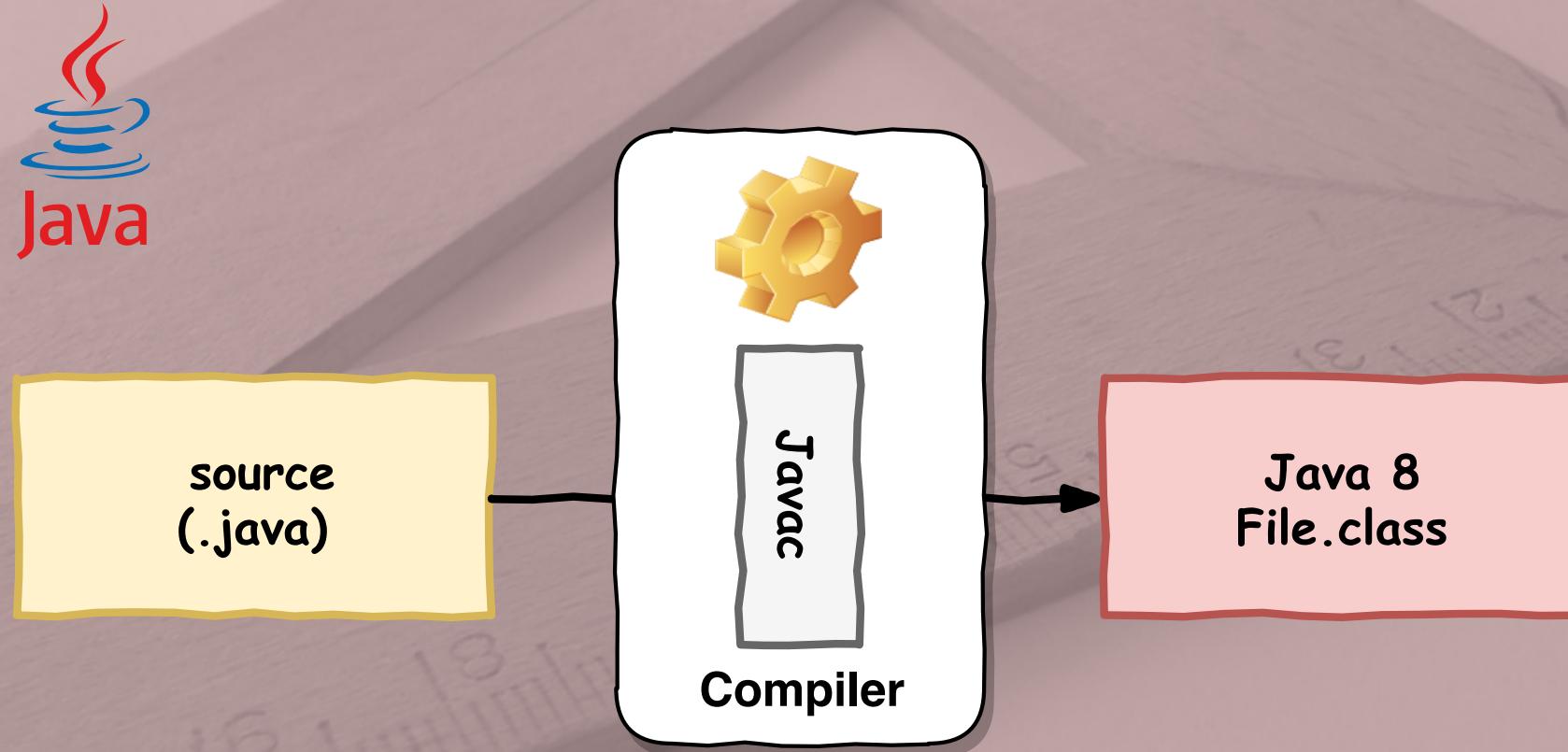
# 4



# Explorons le ByteCode

# 5

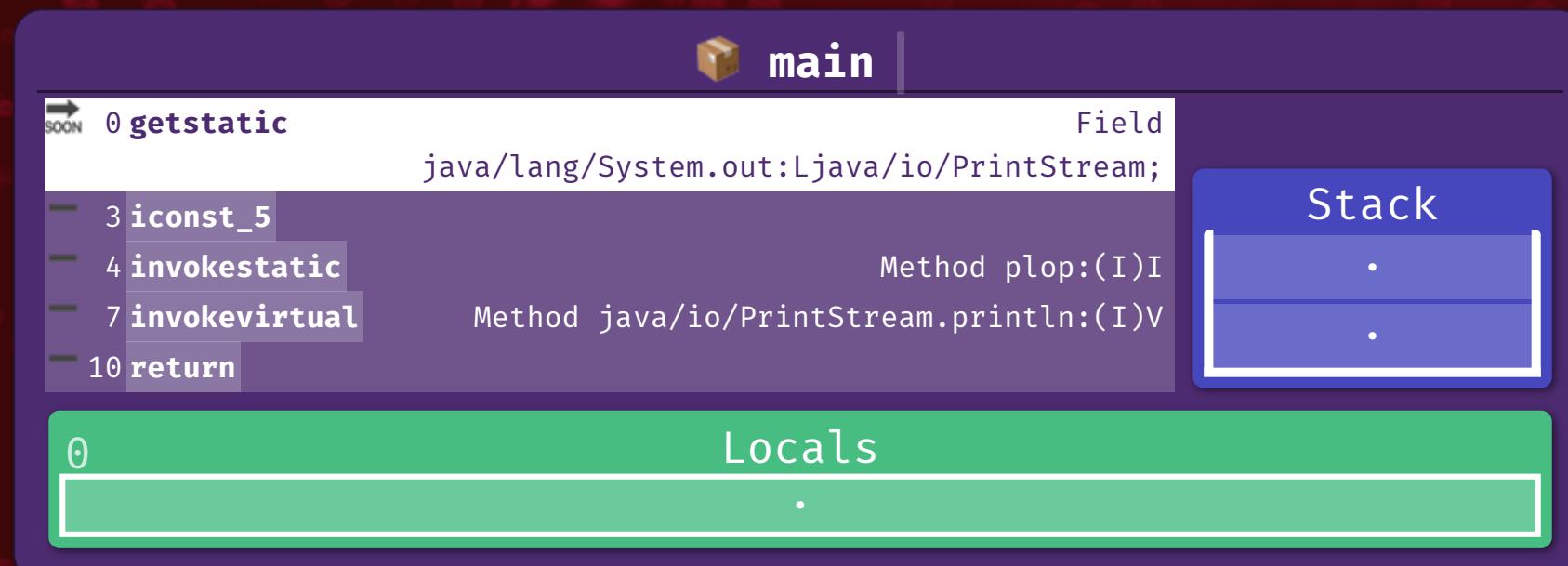




- environ 200 opérations possibles (maxi. 256 opcodes)
- préfix pour le type d'opérations (`i` pour entier, `d` pour double, ...)
- manipulation de la pile, des variables locales (`iconst_0`, `istore`, `iload`, ...)
- contrôle du flux des instructions (`if_icmpgt`, `goto`, ...)
- manipulation d'objets (`invokevirtual`, `invokedynamic`, ...)
- arithmétiques et conversion de type (`iadd`, `iinc`, `i2d`, ...)
- autres (`athrow`, ...)

- ▶ Constant Pool
- ▼ Frames

 Next



- ➔ Mastering Java Bytecode at the Core of the JVM
- ➔ Introduction to Java Bytecode
- ➔ The Java® Virtual Machine Specification
- ➔ The Java Virtual Machine Instruction Set
- ➔ Byte Buddy
- ➔ asm



Soyez curieux: regardez comment ça marche  
avec `javap -c`

# Introduction Kotlin

- 2011

Dévoilé par JetBrains

- 2016:

v1.0

Supporté par Spring Framework

- 2017:

v1.1: coroutines, ...

Officiellement supportée par Google

v1.2: multiplatform

- 2018:

Kotlin Native (external) 0.6



**JVM et Android**



**JavaScript**

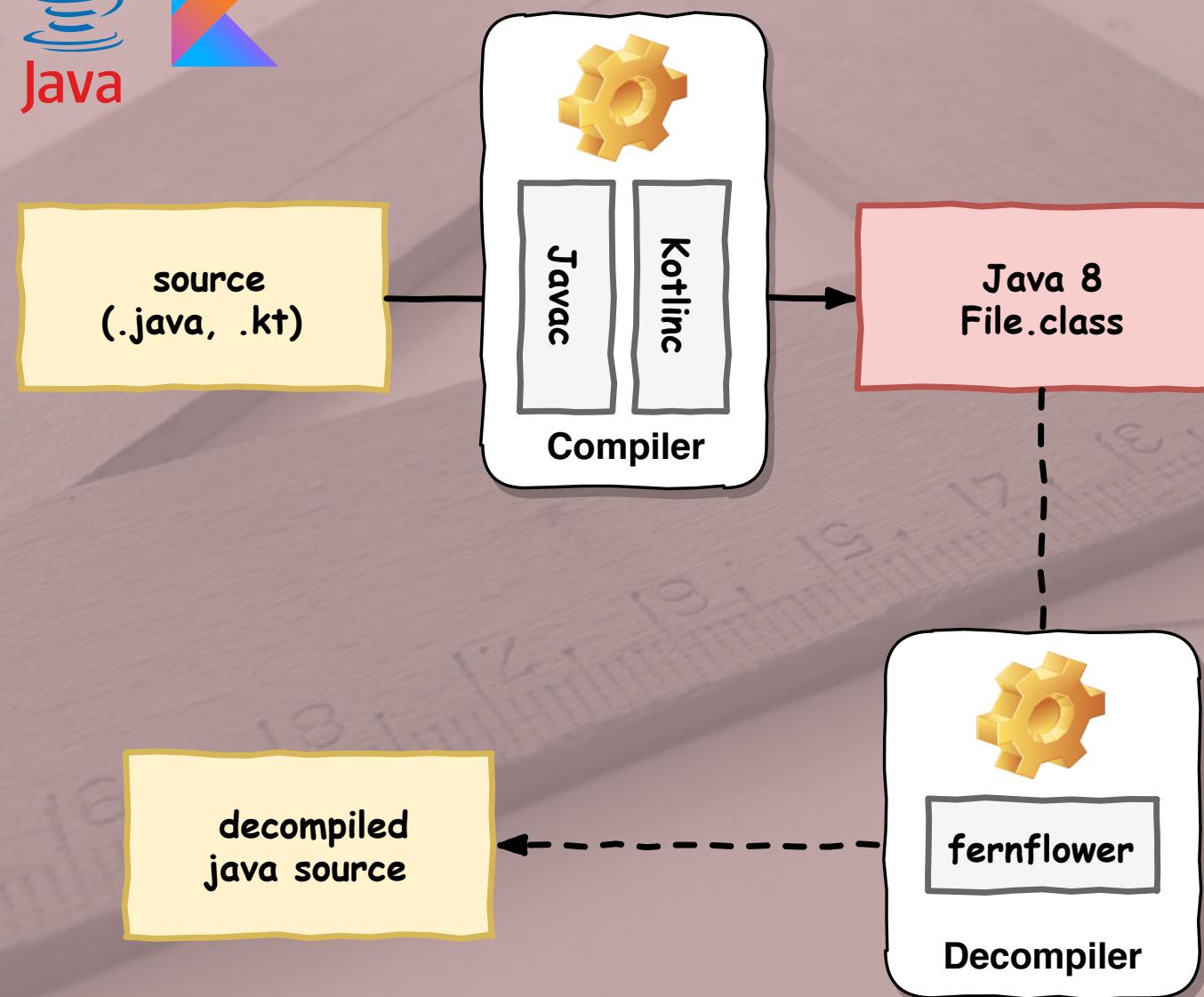


**Native avec  
LLVM**

# HelloWorld.kt

#13





00000000	ca fe ba be 00 00 00 00 32 00 33 01 00 1b 5f 30 30	.....2.3..._00
00000010	5f 68 65 6c 6c 6f 77 6f 72 6c 64 2f 48 65 6c 6c	_helloworld/Hell
00000020	6f 57 6f 72 6c 64 4b 74 07 00 01 01 00 10 6a 61	oWorldKt.....ja
00000030	76 61 2f 6c 61 6e 67 2f 4f 62 6a 65 63 74 07 00	va/lang/Object ..
00000040	03 01 00 04 6d 61 69 6e 01 00 16 28 5b 4c 6a 61	....main ... ([Lja
00000050	76 61 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b 29	va/lang/String;)
00000060	56 01 00 23 4c 6f 72 67 2f 6a 65 74 62 72 61 69	V..#Lorg/jetbrai
00000070	6e 73 2f 61 6e 6e 6f 74 61 74 69 6f 6e 73 2f 4e	ns/annotations/M
00000080	6f 74 4e 75 6c 6c 3b 01 00 04 61 72 67 73 08 00	otNull; ... args ..
00000090	08 01 00 1e 6b 6f 74 6c 69 6e 2f 6a 76 6d 2f 69	....kotlin/jvm/i
000000a0	6e 74 65 72 6e 61 6c 2f 49 6e 74 72 69 6e 73 69	nternal/Intrinsi
000000b0	63 73 07 00 0a 01 00 17 63 68 65 63 6b 50 61 72	cs.....checkPar
000000c0	61 6d 65 74 65 72 49 73 4e 6f 74 4e 75 6c 6c 01	ameterIsNotNull.
000000d0	00 27 28 4c 6a 61 76 61 2f 6c 61 6e 67 2f 4f 62	.'(Ljava/lang/Ob
000000e0	6a 65 63 74 3b 4c 6a 61 76 61 2f 6c 61 6e 67 2f	ject;Ljava/lang/
000000f0	53 74 72 69 6e 67 3b 29 56 0c 00 0c 00 0d 0a 00	String;)V.....
00000100	0b 00 0e 01 00 0c 48 65 6c 6c 6f 20 44 65 76 6f	.....Hello Devc
00000110	78 78 08 00 10 01 00 10 6a 61 76 61 2f 6c 61 6e	xx.....java/lar
00000120	67 2f 53 79 73 74 65 6d 07 00 12 01 00 03 6f 75	g/System.....ou
00000130	74 01 00 15 4c 6a 61 76 61 2f 69 6f 2f 50 72 69	t ... Ljava/io/Pri
00000140	6e 74 53 74 72 65 61 6d 3b 0c 00 14 00 15 09 00	ntStream;.....
00000150	13 00 16 01 00 13 6a 61 76 61 2f 69 6f 2f 50 72	.....java/io/Pr
00000160	69 6e 74 53 74 72 65 61 6d 07 00 18 01 00 07 70	intStream.....p
00000170	72 69 6e 74 6c 6e 01 00 15 28 4c 6a 61 76 61 2f	rintln ... (Ljava/
00000180	6c 61 6e 67 2f 4f 62 6a 65 63 74 3b 29 56 0c 00	lang/Object;)V ..
00000190	1a 00 1b 0a 00 19 00 1c 01 00 13 5b 4c 6a 61 76	.....[Ljav
000001a0	61 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b 01 00	a/lang/String; ..
000001b0	11 4c 6b 6f 74 6c 69 6e 2f 4d 65 74 61 64 61 74	.Lkotlin/Metadat

```
Compiled from "HelloWorld.kt"
public final class _00_helloworld.HelloWorldKt {
    public static final void main(java.lang.String[]);
        Code:
    0:  aload_0
    1:  ldc          #9           // String args
    3:  invokestatic #15          // Method kotlin/j
    6:  ldc          #17          // String Hello De
    8:  astore_1
    9:  getstatic    #23          // Field java/lang
   12:  aload_1
   13:  invokevirtual #29          // Method java/io/
   16:  return
}
```



- Kotlin ajoute des contrôles
- du coup on a besoin de JARs en plus

jar	taille
kotlin-stdlib-1.2.31.jar	919K
kotlin-stdlib-jdk7-1.2.31.jar	3.1K
kotlin-stdlib-jdk8-1.2.31.jar	13K
kotlin-reflect-1.2.31.jar	2.5M
guava-18.0.jar	2.2M
lombok-1.16.18.jar	1.4M
spring-core-5.0.5.RELEASE.jar	1.2M
jackson-databind-2.9.5.jar	1.3M

- Performances ?

II Ne croyez pas les benchmarks, faites les vous-même !

-  <https://github.com/JetBrains/kotlin-benchmarks>
-  <https://github.com/MonkeyPatchIo/kotlin-perf>

Benchmark	Mode	Cnt	Score	Error	Units
testJava	thrpt	200	66490.271	± 879.996	ops/s
testKotlin	thrpt	200	72393.914	± 935.962	ops/s

# Les bases

```
var x: Int = 10
val y: Int = 3
x += 4
// y += 4  Compilation Error

println(x * y) // 42
```



```
fun greeting(who: Someone) {  
    println("Hello $who!")  
    println("Hello ${who.firstName} ${who.lastName}!")  
}
```



# string-template.java



# ByteCode de string-template

#24

```
Compiled from "string-templates.kt"
public final class _01_basic.String_templatesKt {
    public static final void greeting(_01_basic.Someone);
        Code:
            0:  aload_0
            1:  ldc          #9           // String who
            3:  invokesstatic #15          // Method kotlin/j
            6:  new          #17          // class java/lang
            9:  dup
            10: invokespecial #21         // Method java/la
            13: ldc          #23          // String Hello
            15: invokevirtual #27         // Method java/la
            18:  aload_0
            19:  invokevirtual #30         // Method java/la
            22:  bipush        33
            24:  invokevirtual #33          // Method java/la
            27:  invokevirtual #37          // Method java/la
            30:  astore_1
            31:  getstatic     #43          // Field java/lang
            34:  aload_1
            35:  invokevirtual #49         // Method java/io
```

```
val anInt = 42 // type inference: Int
val aLong = 42L // type inference: Long
var aDouble: Double? = null
```





```
Compiled from "numeric.kt"
public final class _01_basic.NumericKt {
    public static final void tryNumeric();
        Code:
            0: bipush      42
            2: istore_0
            3: ldc2_w      #7                      // long 42l
            6: lstore_1
            7: aconst_null
            8: checkcast    #10                     // class java/lang/
            11: astore_3
            12: return
}
```

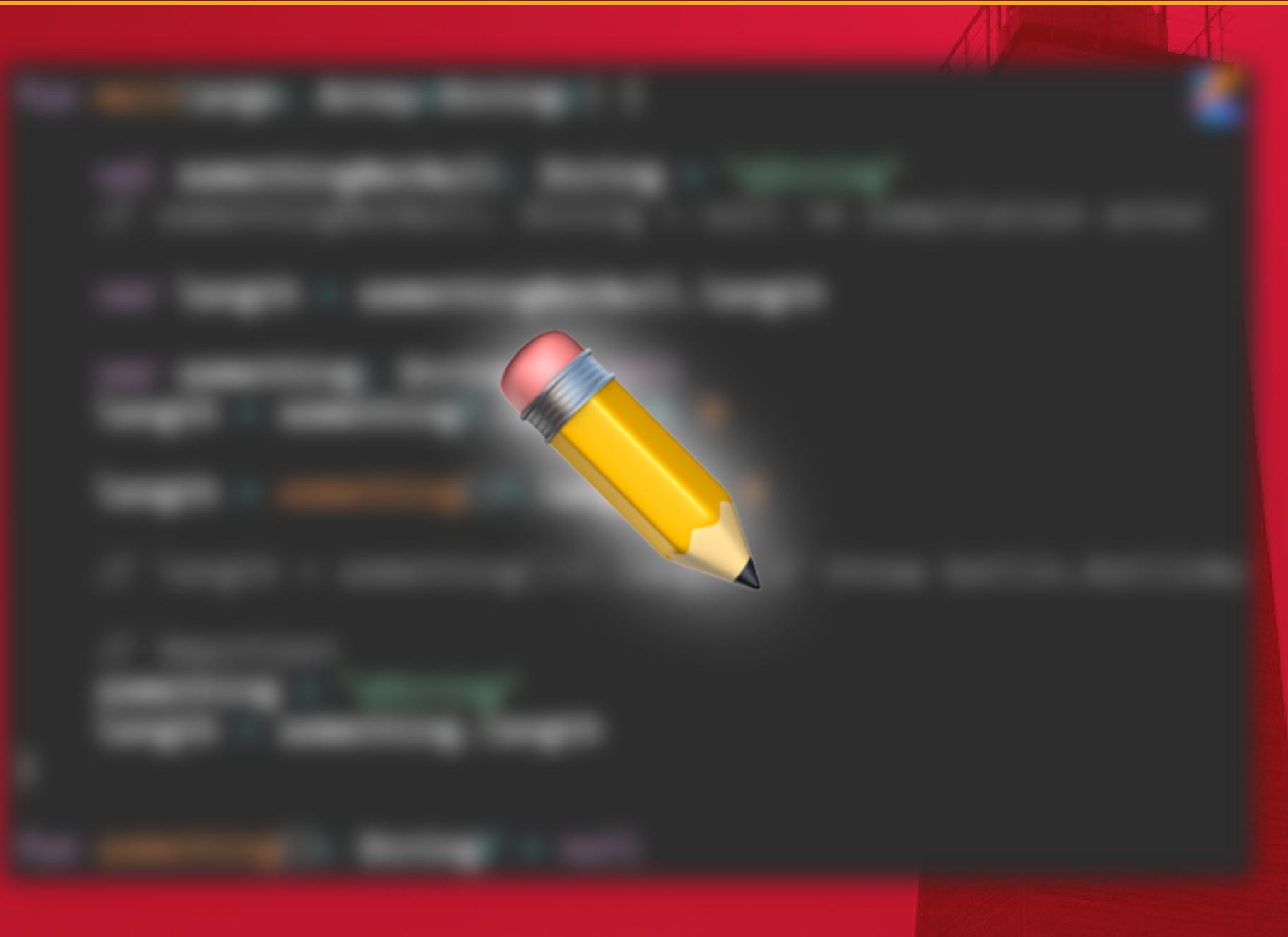
- plus de ; \*
- 😍 String templating
- 😊 plus de types primitifs (avant la compilation)
- 😊 inférence de types
- on peut mélanger du code Java et Kotlin

# null-safety

“ I call it my billion-dollar mistake. It was the invention of the `null` reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a `null` reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

--Tony Hoare (C.A.R. Hoare)

➡ Null References: The Billion Dollar Mistake



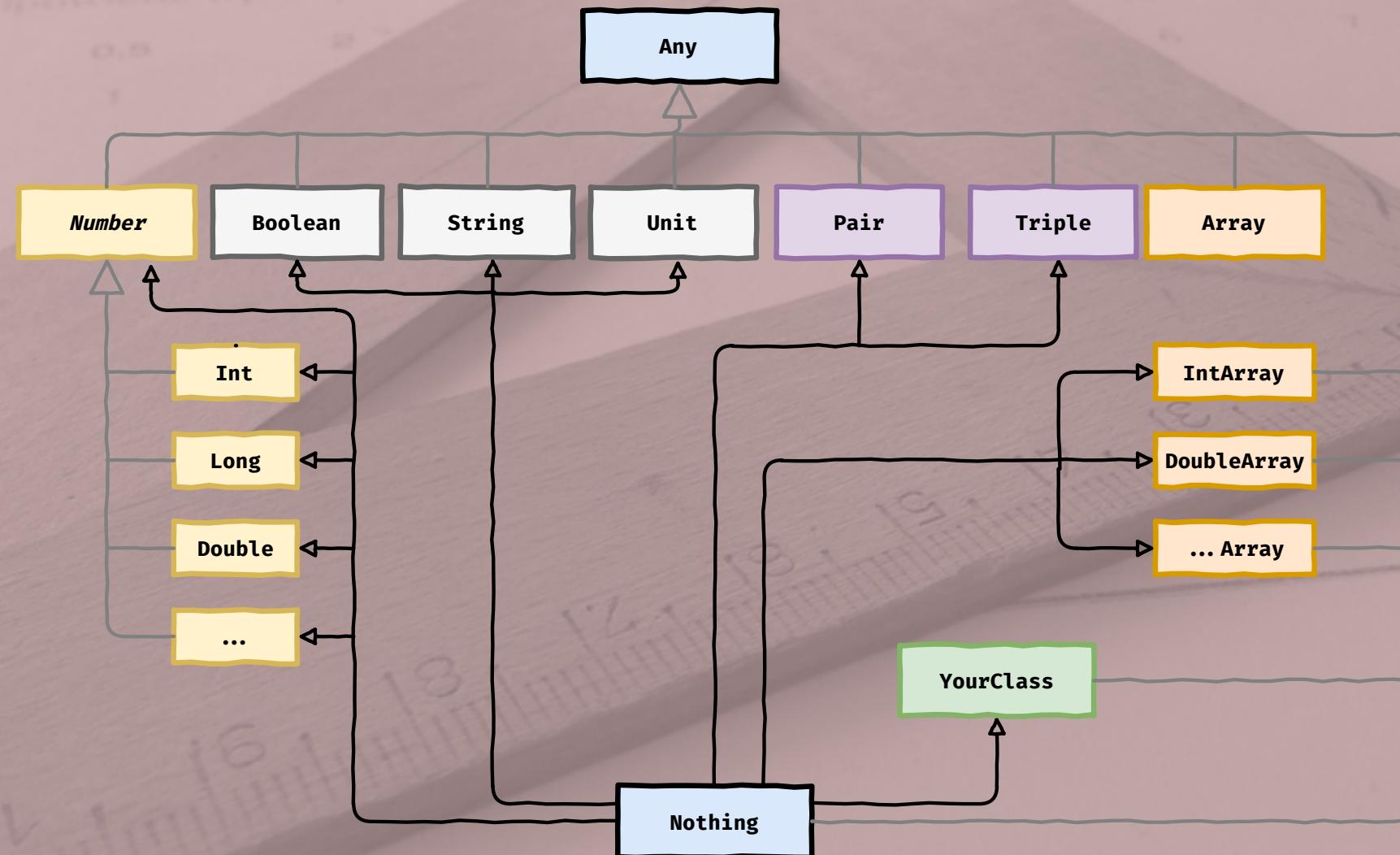


- plus de `NullPointerException`
- ⚠ quand on appelle du Java

# Les types

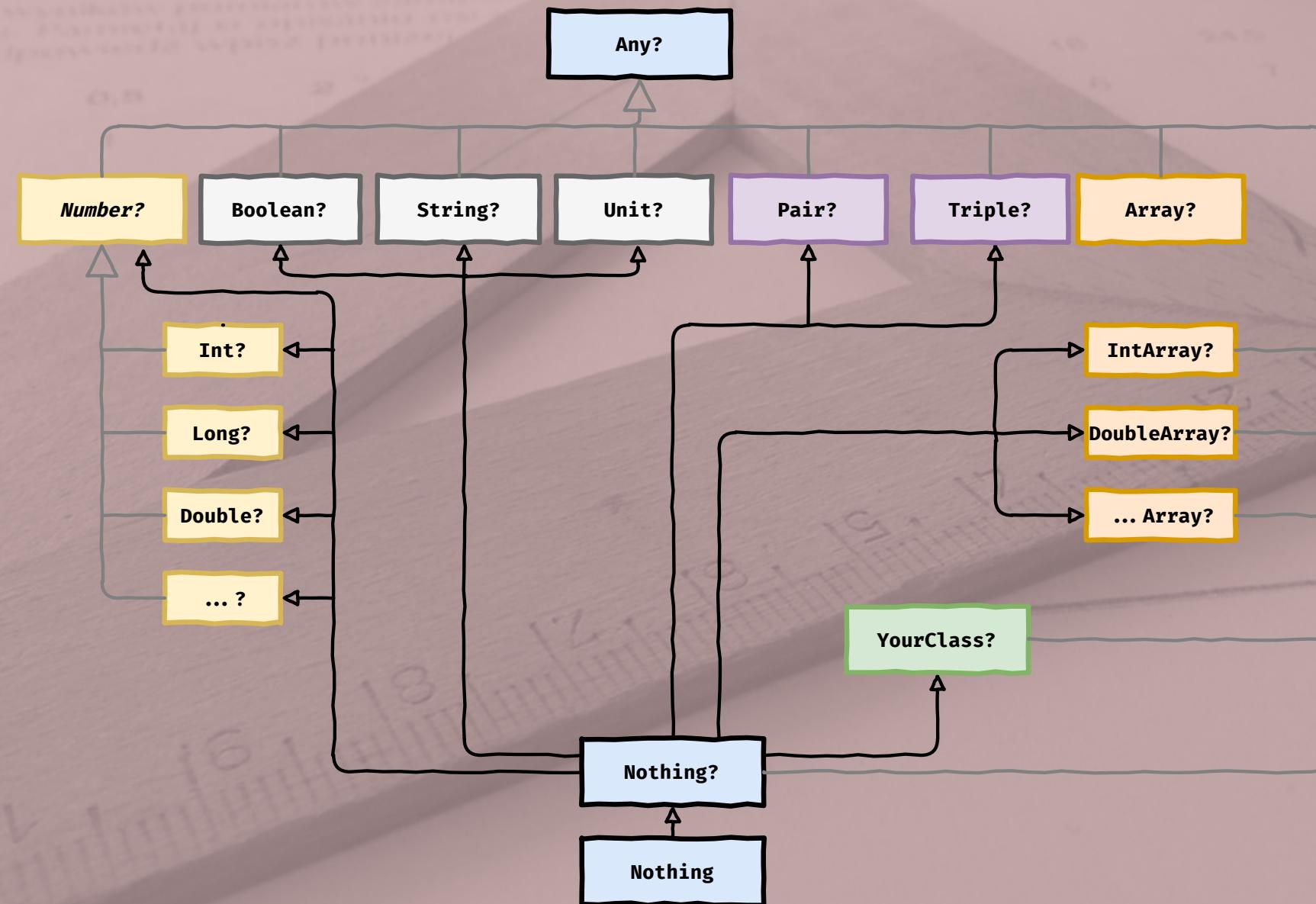
# Types basiques

#35



# Types basiques nullable

#36





-  le TODO( ) est l'ami du TDD

# Les fonctions

```
fun buildString(prefix: String,  
               who: String,  
               enhanced: Boolean): String {  
    var msg = "$prefix $who"  
    if (enhanced) {  
        msg += '!'  
    }  
    return msg  
}  
  
fun greetings(): String =  
    buildString(enhanced = true, who = "Devoxx", prefix = "He")
```





```
fun buildString2(prefix: String = "Hello",  
                 who: String,  
                 enhanced: Boolean = true): String {  
    var msg = "$prefix $who"  
    if (enhanced) {  
        msg += '!'  
    }  
    return msg  
}  
  
fun greetings2(): String =  
    buildString2(who = "Devoxx")
```



# default-value.java



# ByteCode de default-value

# 44

## ✨ Conseils

- Toujours typer le retour de vos fonctions (sauf si c'est évident et une surcharge comme le `toString`)
- Kotlin est plus expressif que Java => évitez de faire des fonctions trop longues
- Sautez une ligne après le `=`
- Utilisez le passage des arguments par nom quand ça lève des ambiguïtés



## Notes

- Le passage des arguments par nom, ne marche pas sur les appels de code Java

# Les lambdas

```
// Declare apply function with function as parameter
fun apply(x: Int, y: Int, operation: (Int, Int) → Int): Int =
    operation(x, y)

// Declare function
fun sumf(x: Int, y: Int) =
    x + y

// call apply with function reference
val sum5 = apply(2,3, ::sumf)

// store function reference
val sumLam = ::sumf

// call apply with the function reference
val sum6 = apply(1,5, sumLam)
```









```
package _04_lambda;

import kotlin.Metadata;
import kotlin.jvm.functions.Function2;
import org.jetbrains.annotations.NotNull;

@Metadata(
    mv = {1, 1, 9},
    bv = {1, 0, 2},
    k = 2,
    d1 = {"\u0000\u0012\n\u0000\n\u0002\u0010\b\n\u0002\b\u0000!"},
    d2 = {"sum3", "", "getSum3", "()I", "sum4", "getSum4", "sum5", "()I!"},
)
public final class LambdaKt {
    @NotNull
    private static final Function2 sum1;
    private static final int sum3;
    private static final int sum4;

    @NotNull
    public static final Function2 getSum1() {
        return sum1;
    }
}
```

```
val other = sumf(1,2)
    .let { it + 1 }

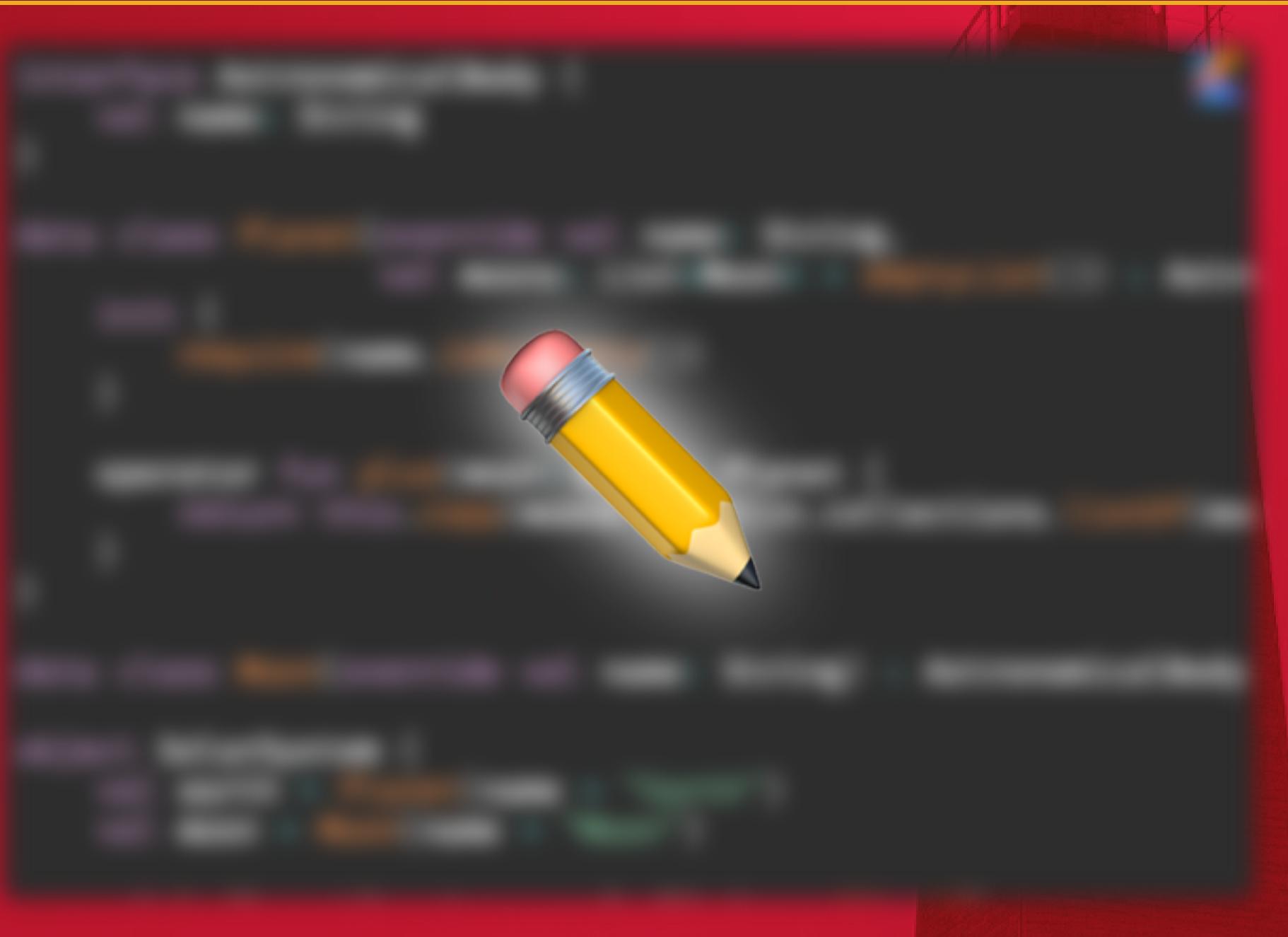
val nullable = maybeAnInt()
    ?.let { it + 1 }
```





- ! pas de `return`
- pensez à mettre vos lambda comme dernier argument
- voir aussi les `apply`, `also`, `run`, `use`, `with`

# Les classes





# Héritage en Kotlin

#57



- ! Les contrôles de types génériques ne sont fait qu'au moment de la compilation
- Covariant (consome): `out`, en java ? `extends T`
- Contravariant (produit): `in`, en java ? `super T`

## Borne supérieur

```
fun <T : Comparable<T>> sort(list: List<T>): List<T>
```

Les détails: 

<https://kotlinlang.org/docs/reference/generics.html>

```
interface Function<in T, out U>
```

```
Function<*, String> // correspond à Function<in Nothing, String>
```

```
Function<Int, *> // correspond à Function<Int, out Any?>
```

```
Function<*, *> // correspond à Function<in Nothing, out Any?>
```

```
sealed class JsonValue

data class JsonObject(val attributes: Map<String, JsonValue>)
data class JsonArray(val values: List<JsonValue>) : JsonValue()
data class JsonString(val value: String) : JsonValue()
data class JsonNumber(val value: Number) : JsonValue()
data class JsonBoolean(val value: Boolean) : JsonValue()
object JsonNull : JsonValue()
```



```
interface Entity

typealias Id = String
typealias Version = Int
typealias EntityKey = Pair<Id, Version>

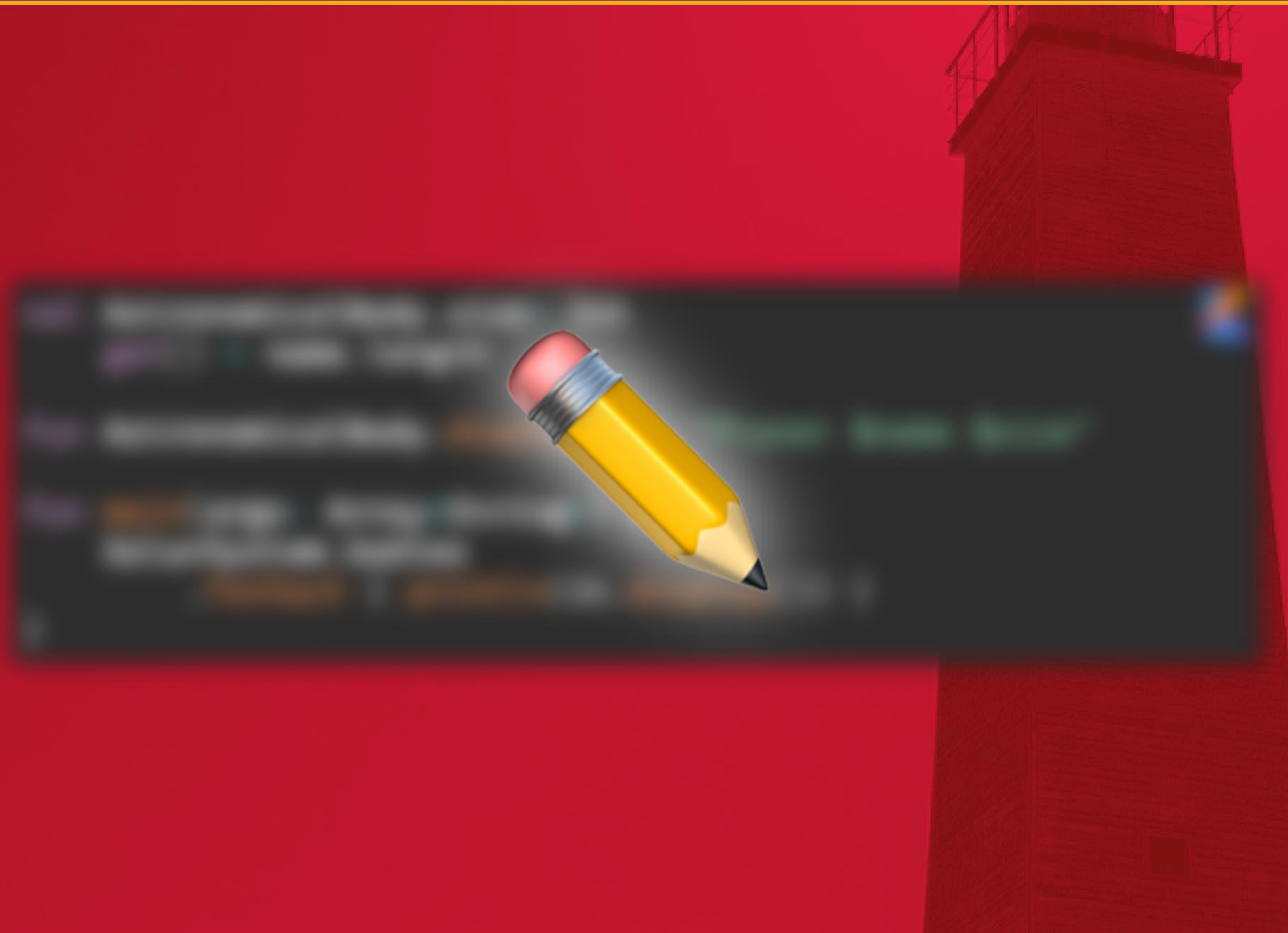
// fun getAllEntities(): Map<Pair<String, Int>, List<Entity>>
fun getAllEntities(): Map<EntityKey, List<Entity>> = emptyMap()
```



```
Compiled from "typealias.kt"
public final class _06_class_2.TypealiasKt {
    public static final java.util.Map<kotlin.Pair<java.lang.String, kotlin.String>> mapOf(
        Code:
            0: invokestatic  #12                      // Method kotlin/collection/MapKt.mapOf(Lkotlin/collection/Map;)
            3: areturn
    }
```

- 😂 **data class**
- 🤔 Mais pourquoi on n'a pas ça en Java ?
- Une seule classe par fichier n'est pas utile
- 😎 **sealed** permet de faire des types algébriques de données (Algebraic Data Type)

# Extensions de fonctions



```
package _08_extension;

import astronomy.AstronomicalBody;
import astronomy.SolarSystem;
import java.util.Iterator;
import kotlin.Metadata;
import kotlin.jvm.internal.Intrinsics;
import org.jetbrains.annotations.NotNull;

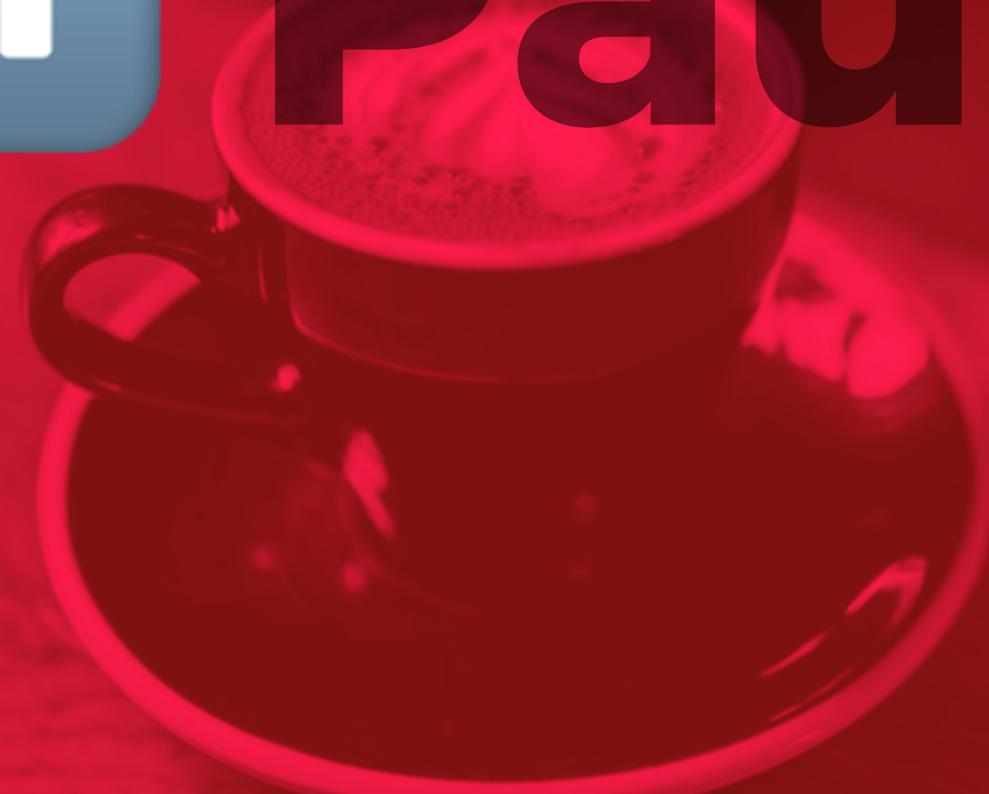
@Metadata(
    mv = {1, 1, 9},
    bv = {1, 0, 2},
    k = 2,
    d1 = {"\u0000 \n\u0000\n\u0002\u0010\b\n\u0002\u0018\u0002"},
    d2 = {"size", "", "Lastronomy/AstronomicalBody;", "getSize"}
)
public final class ExtensionKt {
    public static final int getSize(@NotNull AstronomicalBody $receiver) {
        Intrinsics.checkNotNullParameter($receiver, "$receiver");
        return $receiver.getName().length();
    }
}
```



- Permet d'enrichire les api Java
  -  Spring,  RxKotlin,  SparkJava
- Permet la SoC



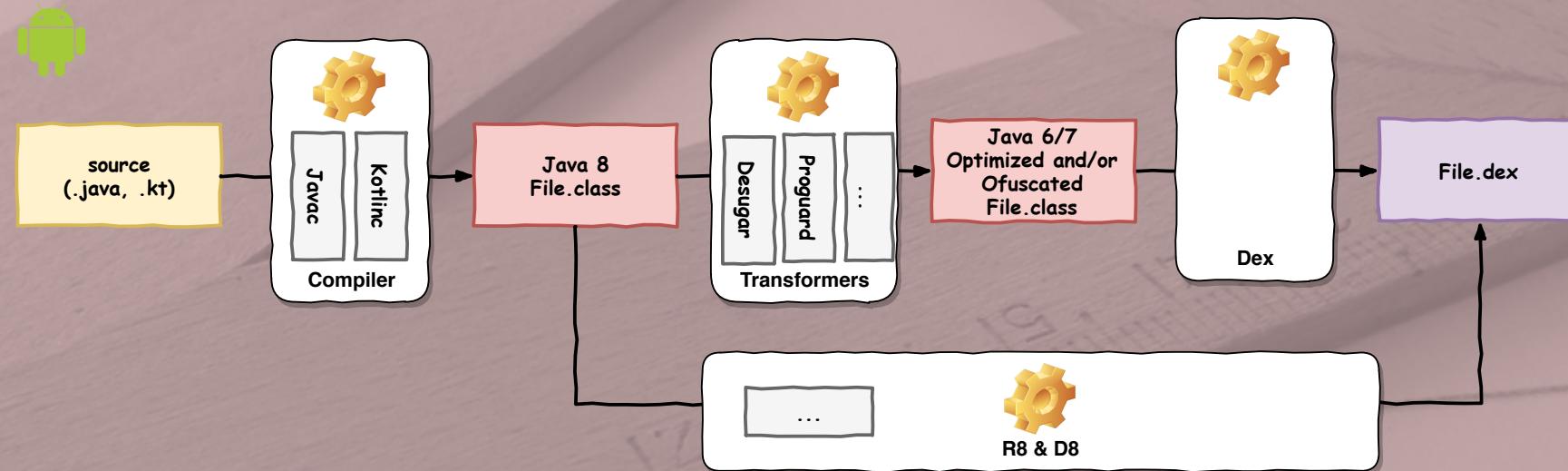
# Pause



# ByteCode Android

# Compilation pour Android

#70





## Dalvik Executable format

```
$ java -jar ./scripts/lib/d8.jar --release \
  --output ./target/android/dex \
  ./target/android/hello.jar
```

00000000	64	65	78	0a	30	33	35	00	06	50	f0	61	50	10	7c	c0	dex.035..P.aP. .
00000010	b2	f9	77	2d	54	df	60	f3	ac	dd	b0	10	eb	55	53	e1	..w-T.`.....US.
00000020	28	f6	10	00	70	00	00	00	78	56	34	12	00	00	00	00	(...p...xV4.....
00000030	00	00	00	00	4c	f5	10	00	12	17	00	00	70	00	00	00	....L.....p...
00000040	31	03	00	00	b8	5c	00	00	2a	08	00	00	7c	69	00	00	1....\..*... i..
00000050	30	03	00	00	74	cb	00	00	20	1a	00	00	f4	e4	00	00	0...t... ....
00000060	75	02	00	00	f4	b5	01	00	94	f1	0e	00	94	04	02	00	u.....
00000070	4a	f1	07	00	4c	f1	07	00	75	f1	07	00	a4	f1	07	00	J...L...u.....
00000080	c2	f1	07	00	e0	f1	07	00	00	f2	07	00	23	f2	07	00	.....#...
00000090	71	f2	07	00	b9	f2	07	00	07	f3	07	00	37	f3	07	00	q.....7...
000000a0	69	f3	07	00	90	f3	07	00	bc	f3	07	00	e3	f3	07	00	i.....
000000b0	27	f4	07	00	71	f4	07	00	8f	f4	07	00	ad	f4	07	00	'...q.....
000000c0	cb	f4	07	00	ed	f4	07	00	0f	f5	07	00	2c	f5	07	00	.....,...
000000d0	49	f5	07	00	79	f5	07	00	b1	f5	07	00	e9	f5	07	00	I...y.....
000000e0	1a	f6	07	00	51	f6	07	00	7b	f6	07	00	a6	f6	07	00	....Q...{....
000000f0	d1	f6	07	00	0a	f7	07	00	47	f7	07	00	84	f7	07	00	.....G....
00000100	c1	f7	07	00	02	f8	07	00	43	f8	07	00	84	f8	07	00	.....C....
00000110	f1	f8	07	00	12	f9	07	00	41	f9	07	00	6e	f9	07	00	.....A...n...
00000120	b0	f9	07	00	dd	f9	07	00	0a	fa	07	00	39	fa	07	00	.....9...
00000130	68	fa	07	00	ab	fa	07	00	e0	fa	07	00	25	fb	07	00	h%.....

```
$ ~/.android-sdk/build-tools/23.0.1/dexdump -d \
  ./target/android/dex/classes.dex | \
  > ./target/android/dex/classes.dex.dump
```

```
Processing './target/android/dex/classes.dex' ...
Opened './target/android/dex/classes.dex', DEX version '035'
Class #0
  Class descriptor : 'L_00_helloworld/HelloWorldKt;'
  Access flags     : 0x0011 (PUBLIC FINAL)
  Superclass       : 'Ljava/lang/Object;'
  Interfaces       :
  Static fields    :
  Instance fields  :
  Direct methods   :
    #0             : (in L_00_helloworld/HelloWorldKt;)
      name          : 'main'
      type          : '([Ljava/lang/String;)V'
      access        : 0x0019 (PUBLIC STATIC FINAL)
      code          :
      registers     : 2
      ins            : 1
      outs           : 2
      insns size    : 13 16-bit code units
020494:                                     | [020494] _00_helloworld.Hello
0204a4: 1a00 1310                           | 0000: const-string v0, "args"
0204a8: 7120 2811 0100                       | 0002: invoke static {v1, v0}
```



# Autres structures

```
fun handleAstronomicalBody(body: AstronomicalBody) {  
    val message =  
        if (body is Planet &&  
            body.name == "Earth"  
        ) "Welcome Earth"  
        else "Welcome martian"  
  
    println(message)  
}
```



```
fun main(args: Array<String>) {  
    for (body in SolarSystem.bodies) { // 🙁  
        print(body)  
    }  
}
```





```
Compiled from "for.kt"
public final class _09_structures.ForKt {
    public static final void main(java.lang.String[]);
    Code:
        0:  aload_0
        1:  ldc          #9           // String args
        3:  invokestatic #15          // Method kotlin/j
        6:  getstatic   #21          // Field astronomy
        9:  invokevirtual #25          // Method astronomy
        12: invokeinterface #31,  1      // InterfaceMethod
        17: astore_2
        18: aload_2
        19: invokeinterface #37,  1      // InterfaceMethod
        24: ifeq         47
        27:  aload_2
        28: invokeinterface #41,  1      // InterfaceMethod
        33: checkcast    #43          // class astronomy
        36: astore_1
        37: getstatic    #49          // Field java/lang
        40:  aload_1
        41: invokevirtual #55          // Method java/io
```

```
while (x > 0) {  
    x--  
}  
  
do {  
    val y = retrieveData()  
} while (y != null) // y is visible here!
```



```
for (body in SolarSystem.bodies) { // 😬\n\n    val message = when (body) {\n        is Planet → "Planet ${body.name}"\n        is Star   → "Star ${body.name}"\n        else       → null\n    }\n\n    if (message ≠ null) {\n        println(message)\n    }\n}
```



```
// Note: assert(n ≥ 0)
fun forFactorial(n: Int): Int { // 😊
    var acc = 1
    for (i in 1..n) {
        acc *= i
    }
    return acc
}
```



# ByteCode factoriel avec for

```
Compiled from "for-factorial.kt"
public final class _09_structures.recusion.For_factorialKt {
    public static final int forFactorial(int);
        Code:
            0:  iconst_1
            1:  istore_1
            2:  iconst_1
            3:  istore_2
            4:  iload_0
            5:  istore_3
            6:  iload_2
            7:  iload_3
            8:  if_icmpgt    26
            11: iload_1
            12: iload_2
            13: imul
            14: istore_1
            15: iload_2
            16: iload_3
            17: if_icmpeq    26
            20: iinc          2, 1
            23: iload_1
            24: ireturn
```

```
// Note: assert(n ≥ 0)
fun recFactorial(n: Int): Int =
    if (n < 1) 1 else n * recFactorial(n - 1)
```



# ByteCode factoriel avec recursivité

#84

```
Compiled from "rec-factorial.kt"
public final class _09_structures.recusion.Rec_factorialKt {
    public static final int recFactorial(int);
        Code:
            0: iload_0
            1: iconst_1
            2: if_icmpge    9
            5: iconst_1
            6: goto        17
            9: iload_0
            10: iload_0
            11: iconst_1
            12: isub
            13: invokestatic #8                  // Method recFacto
            16: imul
            17: ireturn
    }
```

```
// Note: assert(n ≥ 0)
fun tailRecFactorial(n: Int): Int {

    tailrec fun aux(n: Int, acc: Int): Int =
        if (n < 1) 1 else aux(n - 1, acc * n)

    return aux(n, 1)
}
```



# ByteCode factoriel avec recursivité terminal 1/2

#86

```
Compiled from "tailrec-factorial.kt"
public final class _09_structures.recusion.Tailrec_factorialKt {
    public static final int tailRecFactorial(int);
        Code:
            0: getstatic      #12           // Field _09_structu
            3: astore_1
            4: aload_1
            5: iload_0
            6:  iconst_1
            7: invokevirtual #16           // Method _09_structu
            10: ireturn
    }
```

# ByteCode factoriel avec recursivité

## terminal 2/2

# 87

```
Compiled from "tailrec-factorial.kt"
final class _09_structures.recusion.Tailrec_factorialKt$tailRe
    public static final _09_structures.recusion.Tailrec_factorial

public java.lang.Object invoke(java.lang.Object, java.lang.Object)
Code:
  0: aload_0
  1: aload_1
  2: checkcast      #11                      // class java/lang/Object
  5: invokevirtual #15                      // Method java/lang/Object.invoke:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object
  8: aload_2
  9: checkcast      #11                      // class java/lang/Object
 12: invokevirtual #15                      // Method java/lang/Object.invoke:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object
 15: invokevirtual #18                      // Method invoke:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object
 18: invokestatic   #24                      // Method java/lang/Object.invoke:(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object
 21: areturn

public final int invoke(int, int);
Code:
  0: iload_1
  1: iconst_1
  2: isub
  3: ireturn
```

 Ne croyez pas les benchmarks, faites les vous-même !



<https://github.com/MonkeyPatchlo/kotlin-perf>

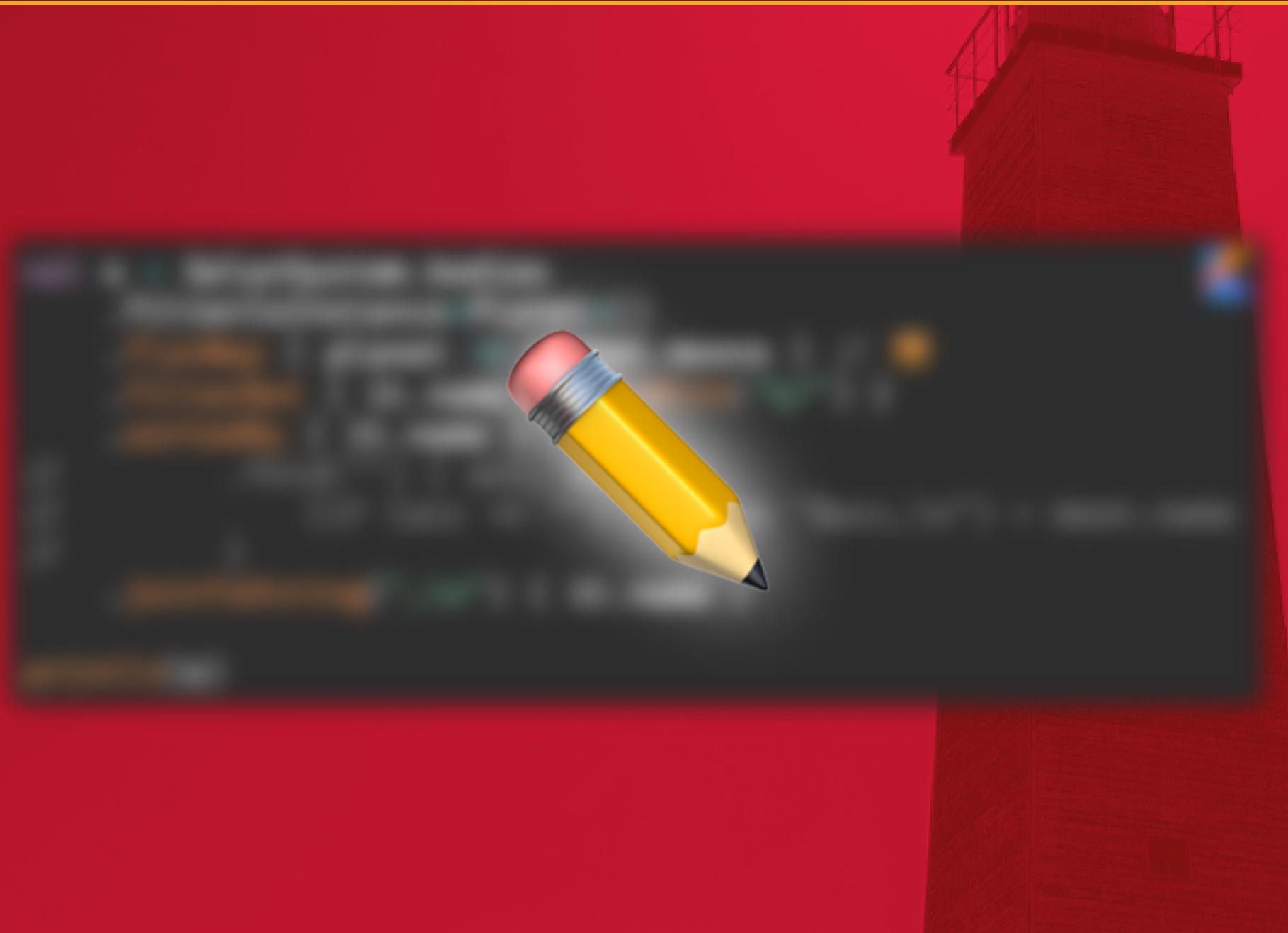
Benchmark	Mode	Cnt	Score	Error	Units
factorialJava	thrpt	200	274141213.561	± 28963758.069	ops/s
factorialKotlinFor	thrpt	200	267717955.205	± 8457315.205	ops/s
factorialKotlinRec	thrpt	200	56270660.700	± 2453418.383	ops/s
factorialKotlinTailRec	thrpt	200	341898899.761	± 11456349.191	ops/s

- Il y a aussi des `break` et `continue`, label pour les boucles
- `when` peut être utiliser avec
  - des constantes,
  - plusieurs valeurs séparées par `,`
  - une expression
  - avec `is` et un type (avec un 'smart cast')

## 💡 Tips

- privilégier les `when` si vous avez plus de 2 cas
- si vous faites des fonctions récursives, faites les `tailrec`

# Les collections



# immutable-mutable.kt

#92



# break-immutable.kt

#93





 Ne croyez pas les benchmarks, faites les vous-même !

Benchmark	Mode	Cnt	Score	Error	Units
collectionApiClassic	thrpt	200	44535.029	± 3550.944	ops/s
collectionApiSequence	thrpt	200	23652.238	± 1967.535	ops/s

```
val s = SolarSystem.bodies.asSequence()  
    .filterIsInstance<Planet>()  
    .flatMap { planet → planet.moons.asSequence() } // 😍  
    .filterNot { it.name.startsWith("S/") }  
    .map { it.name }  
    .first()  
  
println(s)
```



 Ne croyez pas les benchmarks, faites les vous-même !

Benchmark	Mode	Cnt	Score	Error	Units
collectionApiClassicFirst	thrpt	200	241752.062	± 5022.663	ops/s
collectionApiSequenceFirst	thrpt	200	3615451.391	± 454502.198	ops/s



```
for (i in 1..3) print(i) // prints 123  
  
for (i in 3 downTo 1) print(i) // prints 321  
  
for (i in 1..5 step 2) print(i) // prints 135
```





```
fun main(args: Array<String>) {  
    val aPair = Pair("Earth", "Moon")  
    val (planet, moon) = aPair  
  
    val aTriple = Triple("Voyager 1", 1977, listOf("Jupiter",  
    val (probeName, launchYear, flyOver) = aTriple  
}
```

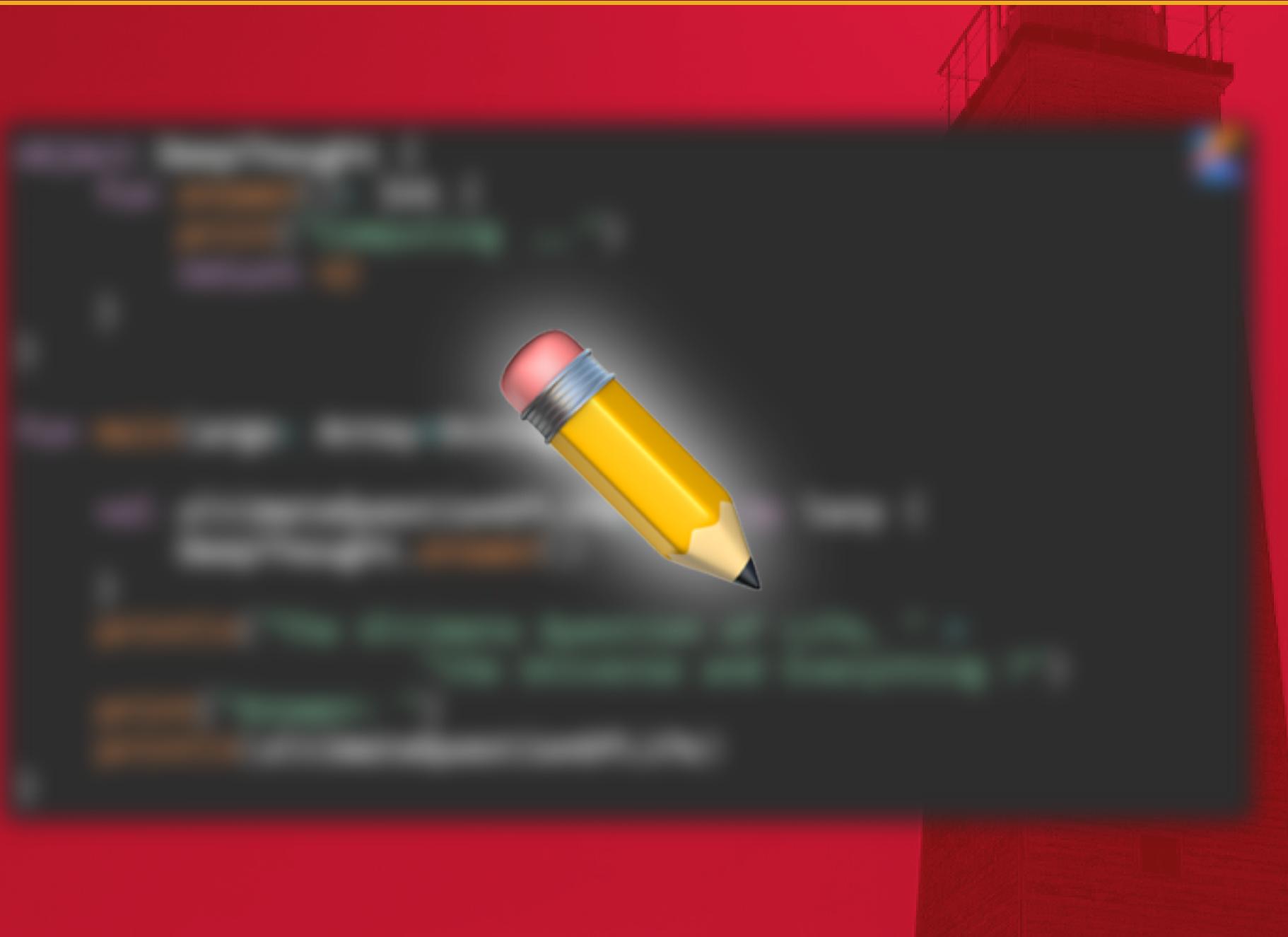




- Super on a de l'immutabilité, des `map`, `flatMap`, `fold`, `aggregate`...
- Mais ça reste des collections Java
- Avant d'utiliser les séquences, faites des mesures

# Les delegates







```
lateinit var str: String

fun main(args: Array<String>) {
    // println(str) kotlin.UninitializedPropertyAccessException
    str = "Hello Devoxx"
    println(str)
}
```



- lateinit : évite les null check pour les propriétés qui ne peuvent être initialisées immédiatement (ex référence de vues sur **Activity**, **Fragment**).
  - Ne peut pas être utilisé avec les types primitifs
- Lazy : utile pour les propriétés qui ne sont pas systématiquement utilisées.
  - ⚠ À manipuler avec précaution dans les activités Android ( avec le cycle de vie, cela peut référencer une ancienne instance)
- Delegate : Observable, Not null, ...

# Un peu plus sur les fonctions

```
import java.time.Instant

class Logger(private val name: String) {
    private enum class Level { TRACE, DEBUG, INFO, WARN, ERROR }
    private val level = Level.INFO

    fun info(message: () -> String) {
        log(Level.INFO, message)
    }

    private inline fun loglvl: Level, message: () -> String)
        if (level >= lvl) {
            println("[${level.name}] $name - ${message()}")
        }
    }

    fun main(args: Array<String>) {
        val logger = Logger("Main")

        logger.info { "Time: ${Instant.now()}" }
    }
}
```







## Cas d'utilisation du `reified`

- pour créer des extensions kotlin des fonctions Java qui utilisent des `Class<T>`

## Cas d'utilisation des `inline`, `noinline`

- quand on utilise `reified`
- quand on sait se qu'on fait,   
<https://kotlinlang.org/docs/reference/inline-functions.html>

# Conclusion

- Faible surcharge
- Support officiel par Google
-  [Using Project Kotlin for Android](#)
-  [Kotlin Guide](#)
-  [android-ktx](#)
-  [Kotlin Android Extensions](#)

- Supporter officiellement depuis  Spring 5,  Spring Boot 2
-  SparkJava,  javalin
-  Vert.x
-  KTor
- ...

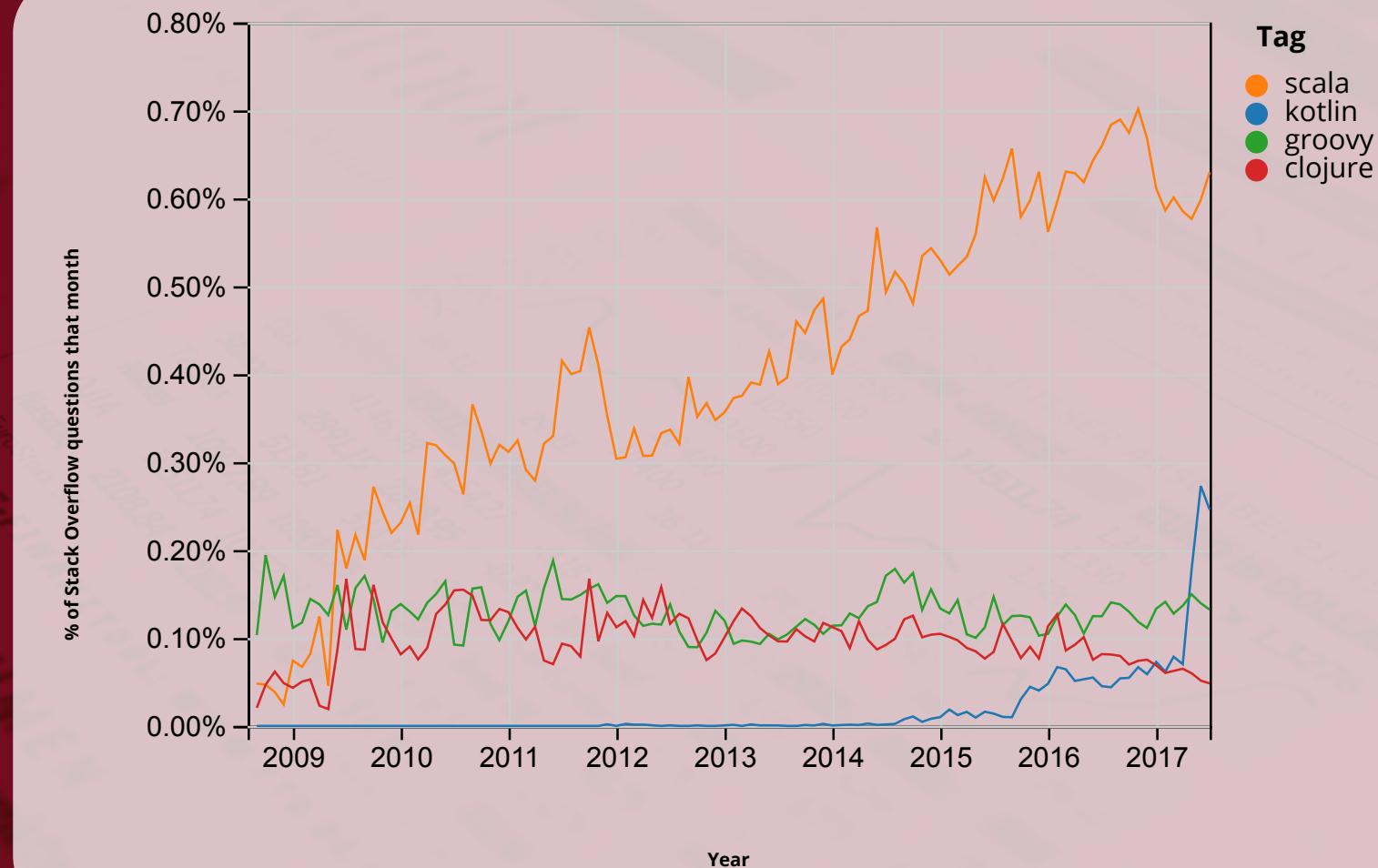
## Web

- Partager du code commun
-  Use Kotlin with npm, webpack and react

## Natif

- Faire des applications sans JVM
- Partager du code avec iOS
- WebAssembly

- 💎 JVM
- 😎 Le byte code c'est cool
- 🎯 Généralement, ça ne suffit pas pour prédire les performances
- ⚖️ Mesurez !



Stackoverflow insights

- C'est déjà mature
- 🤝 Code plus expressif, plus sûr, plus simple
- 🤝 Interopérable avec Java
- 🤝 Outilage (éditeur, gradle, maven)
- 🤝 Ecosystème et communauté
- 🚀 Évolution rapide
- 🐚 Code multiplatform

“ Kotlin réussit une belle alchimie entre pragmatisme, puissance, sûreté, accessibilité.

-  Référence
-  Blog
-  Forum
-  Slack
-  Koans
-  KEEP - Kotlin Evolution and Enhancement Process

- Slides en HTML: ➡ <http://bit.ly/KotlinDevoxxFR>
- Slides en PDF: <http://bit.ly/kotlinDevoxxFRpdf>
-  [kotlin-perf](#)
-  [Kotlin by example](#)

-  kotlinx.serialization
-  kotlinx.coroutines
-  KotlinTest
-  Javalin
-  RxKotlin
-  Arrow
- ...
-  Kotlin is Awesome

# Questions ?

*Pensez au votes et aux retours*