

Deep Dive Kotlin : du Hello World au ByteCode



Emmanuel Vinas

Expert Android & Java

 @emmanuelvinas

 emmanuel@monkeypatch.io



Igor Laborie

Expert Java & Web

 @ilaborie

 igor@monkeypatch.io





Roadmap

#1

- I. ByteCode Java ?
- II. Introduction Kotlin
- III. Les bases
- IV. null-safety
- V. Les fonctions
- VI. Les lambdas
- VII. Les classes
- VIII. Types
- IX. Extensions de fonction
- X. Structure
- XI. Pause
- XII. ByteCode Android
- XIII. Collection
- XIV. Delegate
- XV. Plus sur les fonctions
- XVI. Serialization
- XVII. Coroutines
- XVIII. DSL
- XIX. Conclusion

ByteCode Java ?

HelloWorld.java

#3

```
package _00_helloworld;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello Devoxx");
    }
}
```

```
$ javac HelloWorld.java
```

Java ByteCode binary

4

```
$ hexdump -C HelloWorld.class
```

00000000	ca fe ba be 00 00 00 00 34 00 1d 0a 00 06 00 0f 09 4.....
00000010	00 10 00 11 08 00 12 0a 00 13 00 14 07 00 15 07
00000020	00 16 01 00 06 3c 69 6e 69 74 3e 01 00 03 28 29 ()
00000030	56 01 00 04 43 6f 64 65 01 00 0f 4c 69 6e 65 4e V ... Code ... LineN
00000040	75 6d 62 65 72 54 61 62 6c 65 01 00 04 6d 61 69 umberTable ... mai
00000050	6e 01 00 16 28 5b 4c 6a 61 76 61 2f 6c 61 6e 67 n ... ([Ljava/lang
00000060	2f 53 74 72 69 6e 67 3b 29 56 01 00 0a 53 6f 75 /String;)V ... Sou
00000070	72 63 65 46 69 6c 65 01 00 0f 48 65 6c 6c 6f 57 rceFile ... HelloW
00000080	6f 72 6c 64 2e 6a 61 76 61 0c 00 07 00 08 07 00 orld.java.....
00000090	17 0c 00 18 00 19 01 00 0c 48 65 6c 6c 6f 20 44 Hello D
000000a0	65 76 6f 78 78 07 00 1a 0c 00 1b 00 1c 01 00 19 evoxx.....
000000b0	5f 30 30 5f 68 65 6c 6c 6f 77 6f 72 6c 64 2f 48 _00_helloworld/H
000000c0	65 6c 6c 6f 57 6f 72 6c 64 01 00 10 6a 61 76 61 elloWorld ... java
000000d0	2f 6c 61 6e 67 2f 4f 62 6a 65 63 74 01 00 10 6a /lang/Object ... j
000000e0	61 76 61 2f 6c 61 6e 67 2f 53 79 73 74 65 6d 01 ava/lang/System.
000000f0	00 03 6f 75 74 01 00 15 4c 6a 61 76 61 2f 69 6f .. out ... Ljava/io
00000100	2f 50 72 69 6e 74 53 74 72 65 61 6d 3b 01 00 13 /PrintStream; ...
00000110	6a 61 76 61 2f 69 6f 2f 50 72 69 6e 74 53 74 72 java/io/PrintStr
00000120	65 61 6d 01 00 07 70 72 69 6e 74 6c 6e 01 00 15 eam ... println ...
00000130	28 4c 6a 61 76 61 2f 6c 61 6e 67 2f 53 74 72 69 (Ljava/lang/Stri
00000140	6e 67 3b 29 56 00 21 00 05 00 06 00 00 00 00 00 ng;)V.!.....
00000150	02 00 01 00 07 00 08 00 01 00 09 00 00 00 1d 00

Explorons le ByteCode

5

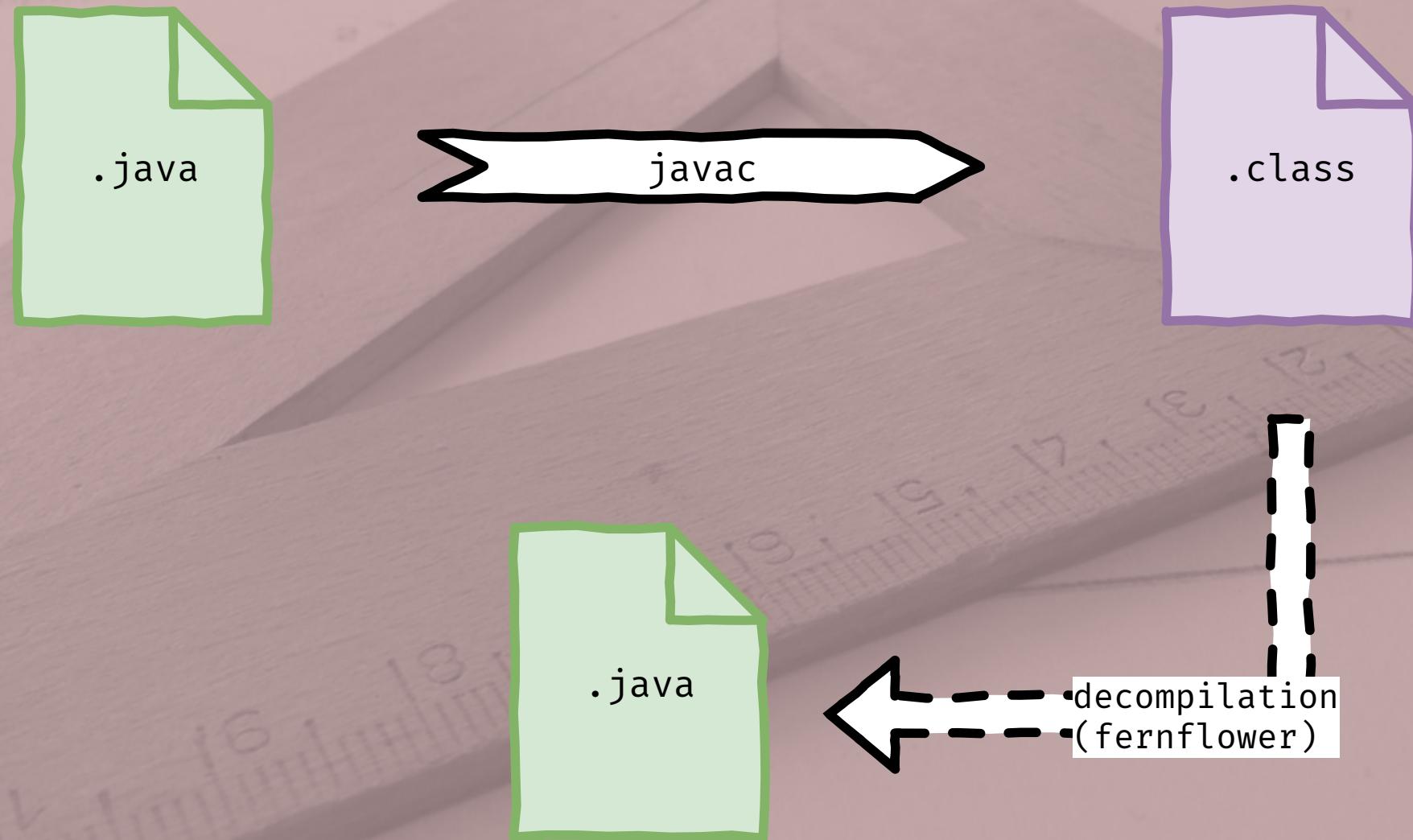
```
$ javap -c HelloWorld.class
```

```
Compiled from "HelloWorld.java"
public class _00_helloworld.HelloWorld {
    public _00_helloworld.HelloWorld();
        Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<":()V
            4: return

    public static void main(java.lang.String[]);
        Code:
            0: getstatic      #2                  // Field java/lang/System.out:Ljav
            3: ldc           #3                  // String Hello Devoxx
            5: invokevirtual #4                  // Method java/io/PrintStream.prin
            8: return
}
```

Transpile

#6



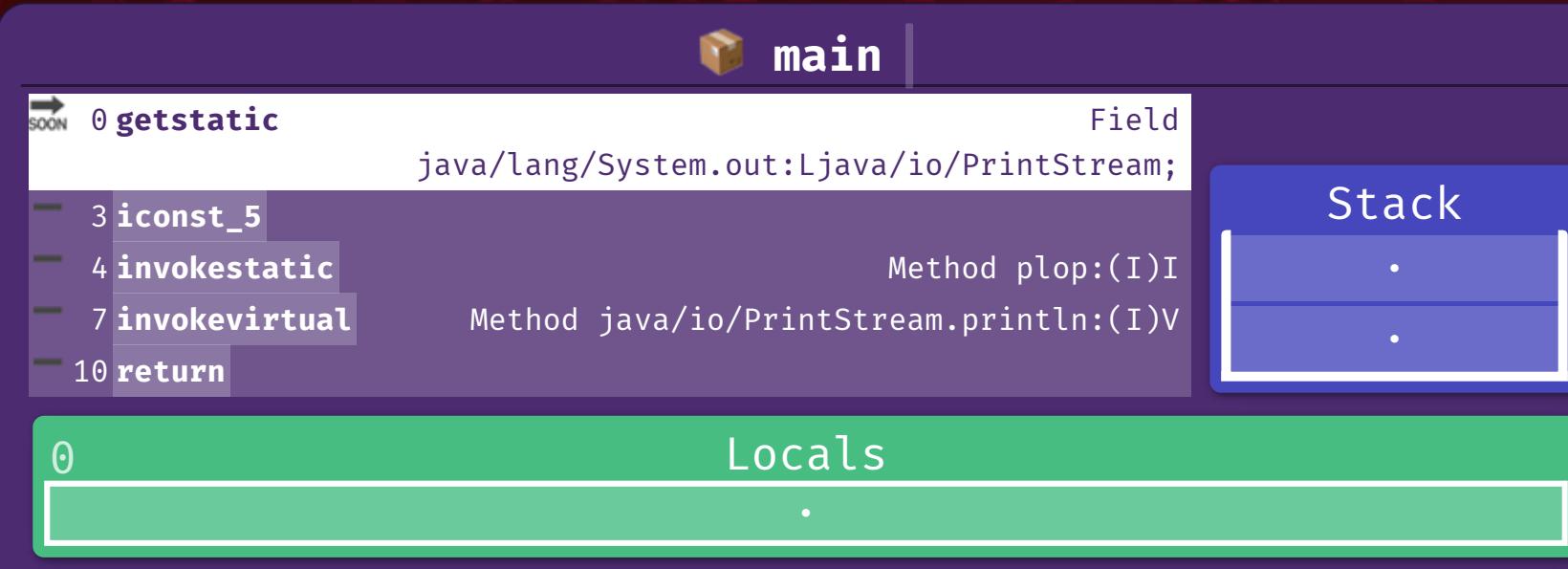
- environ 200 opérations possibles (maxi. 256 opcodes)
- préfix pour le type d'opérations (`i` pour entier, `d` pour double, ...)
- manipulation de la pile, des variables locales
(`iconst_0`, `istore`, `iload`, ...)
- contrôle du flux des instructions (`if_icmpgt`, `goto`, ...)
- manipulation d'objets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

Jouons un peu

8

- ▶ 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

```
package _00_helloworld

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

```
$ kotlinc HelloWorld.kt
```

00000000	ca fe ba be 00 00 00 32	00 33 01 00 1b 5f 30 302.3 ... _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 61main ... ([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/N
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/Intrinsic
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 6fHello Devo
00000110	78 78 08 00 10 01 00 10	6a 61 76 61 2f 6c 61 6e	xx.....java/lan
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 72java/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	aLang/String.

```
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/jvm/internal/Intr
            6: ldc           #17                 // String Hello Devoxx
            8: astore_1
            9: getstatic     #23                 // Field java/lang/System.out:Ljav
            12: aload_1
            13: invokevirtual #29                 // Method java/io/PrintStream.prin
            16: return
    }
```

```
package _00_helloworld;

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\u0012\n\u0000\n\u0002\u0010\u0002\n\u0000\
    d2 = {"main", "", "args", "", "", "([Ljava/lang/String;)
)
public final class HelloWorldKt {
    public static final void main(@NotNull String[] args) {
        Intrinsics.checkNotNull(args, "args");
        String var1 = "Hello Devoxx";
        System.out.println(var1);
```

- 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
spring-core-5.0.5.RELEASE.jar	1.2M
jackson-databind-2.9.5.jar	1.3M
logback-classic-1.2.3.jar	284K

- Performances ?

-  <https://github.com/JetBrains/kotlin-benchmarks>
-  Kotlin Hidden Costs Benchmark
 -  kotlin 1.1.3
 -  Part 1
 -  Part 2
 -  Part 3
 - Kotlin Hidden Costs - Benchmarks
 -  MàJ versions Kotlin, JMH

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



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

Benchmark	Mode	Cnt	Score	Error	Units
MyBenchmark.testJava	thrpt	200	66490.271	± 879.996	ops/s
MyBenchmark.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}!")  
}
```

```
package _01_basic;

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\f\n\u0000\n\u0002\u0010\u0002\n\u0000\n\u0000"},
    d2 = {"greeting", "", "who", "L_01_basic/Someone;"}
)
public final class String_templatesKt {
    public static final void greeting(@NotNull Someone who) {
        Intrinsics.checkNotNullParameter(who, "who");
        String var1 = "Hello " + who + '!';
        System.out.println(var1);
    }
}
```

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: invokestatic  #15          // Method kotlin/jvm/internal/Intr
            6: new           #17          // class java/lang/StringBuilder
            9: dup
            10: invokespecial #21         // Method java/lang/StringBuilder.
            13: ldc           #23          // String Hello
            15: invokevirtual #27         // Method java/lang/StringBuilder.
            18: aload_0
            19: invokevirtual #30         // Method java/lang/StringBuilder.
            22: bipush        33
            24: invokevirtual #33         // Method java/lang/StringBuilder.
            27: invokevirtual #37         // Method java/lang/StringBuilder.
            30: astore_1
            31: getstatic     #43          // Field java/lang/System.out:Ljav
            34: aload_1
            35: invokevirtual #49         // Method java/io/PrintStream.prin
            38: new           #17          // class java/lang/StringBuilder
            41: dup
            42: invokespecial #21         // Method java/lang/StringBuilder.
            45: ldc           #23          // String Hello
            47: invokevirtual #27         // Method java/lang/StringBuilder.
            50: aload_0
```

```
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/Double
           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

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 = "
```

```
package _03_fun;

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\u0010\n\u0000\n\u0002\u0010\u000e\n\u0002\
    d2 = {"buildString", "", "prefix", "who", "enhanced", ""}
)
public final class NamedKt {
    @NotNull
    public static final String buildString(@NotNull String p
        Intrinsics.checkNotNull(prefix, "prefix");
        Intrinsics.checkNotNull(who, "who");
```

```
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")
```

```
package _03_fun;

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\u0010\n\u0000\n\u0002\u0010\u000e\n\u0002\
    d2 = {"buildString2", "", "prefix", "who", "enhanced", "
)
public final class Default_valueKt {
    @NotNull
    public static final String buildString2(@NotNull String
        Intrinsics.checkNotNull(prefix, "prefix");
        Intrinsics.checkNotNull(who, "who");
```

ByteCode de default-value

#36

```
Compiled from "default-value.kt"
public final class _03_fun.Default_valueKt {
    public static final java.lang.String buildString2(java.lang.String, java.lang.String)
        Code:
            0: aload_0
            1: ldc           #9                  // String prefix
            3: invokestatic  #15                 // Method kotlin/jvm/internal/InternalKt.appendStringToString
            6: aload_1
            7: ldc           #17                 // String who
            9: invokestatic  #15                 // Method kotlin/jvm/internal/InternalKt.appendStringToString
           12: new           #19                 // class java/lang/StringBuilder
           15: dup
           16: invokespecial #23                 // Method java/lang/StringBuilder.<init>
           19: ldc           #25                 // String
           21: invokevirtual #29                 // Method java/lang/StringBuilder.append
           24: aload_0
           25: invokevirtual #29                 // Method java/lang/StringBuilder.append
           28: bipush        32
           30: invokevirtual #32                 // Method java/lang/StringBuilder.append
           33: aload_1
           34: invokevirtual #29                 // Method java/lang/StringBuilder.append
           37: invokevirtual #36                 // Method java/lang/StringBuilder.append
           40: astore_3
           41: iload_2
           42: ifeq          66
           45: aload_3
           46: new           #19                 // class java/lang/StringBuilder
```

✨ 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

Les classes

```
class Point(x: Int, y: Int) {  
    val x = x  
    var y = y  
}  
  
// val p1 = Point(2, 4)
```



```
class Point2(val x: Int, var y: Int)  
  
// val p2 = Point2(2, 4)
```



```
class Point3(val x: Int, val y: Int) {  
    constructor(pair: Pair<Int, Int>) : this(pair.first, pa  
}  
  
// val p3 = Point3(Pair(2, 4))
```



```
class Point4(val x: Int, val y: Int) {  
    constructor(pair: Pair<Int, Int>) : this(pair.first, pa  
  
    init {  
        println("( $x, $y )")  
    }  
}  
  
// val p4 = Point4(Pair(2, 4))
```



```
open class Animal {  
    open fun talk(): String =  
        "???"  
}  
  
data class Cat(val name: String) : Animal() {  
    override fun talk(): String =  
        "Meow"  
}  
  
data class Dog(val name: String) : Animal() {  
    override fun talk(): String =  
        "Woof"  
}  
  
fun main(args: Array<String>) {  
    val pets: List<Animal> = listOf(Cat("Felix"), Dog("Rex"))  
}
```

- Covariant (consome): `out`
- Contravariant (produit): `in`
- Borne supérieur :

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

⚠ Les contrôles de types génériques ne sont fait qu'au moment de la compilation

- 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>> = emptyM
```

```
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>> = emptyM
```

- 🤔 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)

Types

Hierarchie des types

#57

Extensions de fonction

Structure

Structures

#61

Pause

ByteCode Android

Collection

Collection

#67

#DevoxxFR #Kotlin @emmanuelVinas @ilaborie

DEVOXX
France™

Delegate

Plus sur les fonctions

Plus sur les fonctions

#71

Serialization

Coroutines

DSL

Conclusion

- Faible surcharge
- Support officiel par Google
-  Using Project Kotlin for Android
-  Kotlin Guide
-  Kotlin extensions for Android

- Supporter officiellement depuis  Spring 5,  Spring Boot 2
 -  SparkJava, 
 -  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

