Kotlin

De la isla de Java a la isla de Kotlin

Adrián Arroyo Calle

Abril 2023 GDG Valladolid

Kotlin

$ whoami

- Adrián Arroyo Calle
- En Telefónica desde 2019
  - Ahora mismo proyecto Agente Único HaaC
  - 90% Kotlin
- También he dado clases en la UVa
- Contributor de Scryer Prolog
- Tengo una web con proyectos y un blog donde intento escribir todos los meses: https://adrianistan.eu.

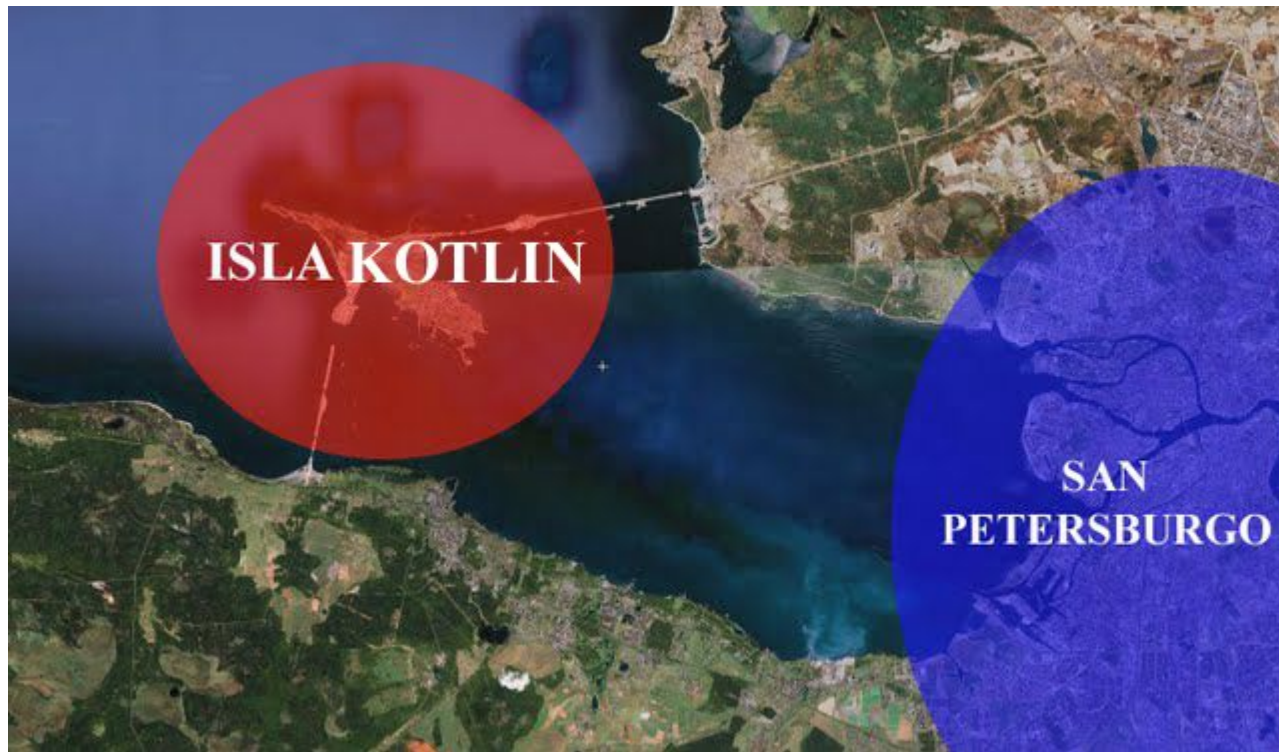
Kotlin

La historia de Kotlin

2010

2017

Kotlin



Kotlin

¿Por qué Kotlin?

Kotlin se define como un lenguaje **pragmático** y **conciso**.

En aquella época el desarrollo de Java se encontraba paralizado. Compra de Sun por parte de Oracle.

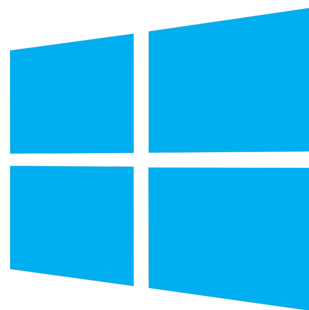
Muchos lenguajes alternativos surgieron en la JVM: **Scala**, **Clojure**, **Groovy**, …

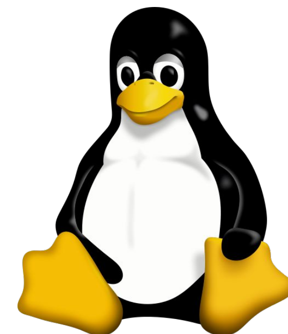
Pero ninguno se aproximaba a una versión de Java *mejorada* sino que tenían ideas propias.

¿Si Java se diseñase en el siglo XXI como sería?

Kotlin

¿Dónde puedo usar Kotlin?





Kotlin

¡Hola Mundo!

Es recomendable usar IntelliJ para comenzar con Kotlin. Desde ahí podemos crear un proyecto nuevo de Kotlin/JVM.

```kotlin
fun main() {
  println("Hello World!")
}
```

KotlinTalk – Main.kt

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

KotlinTalk 〉 src 〉 main 〉 kotlin 〉 Main.kt Current File

Project

Main.kt Notifications

```
1 fun main() {
2     println("Hello World!")
3 }
```

Run: MainKt

Structure

```
/usr/lib/jvm/java-17-openjdk/bin/java -javaagent:/usr/share/idea/lib/idea_rt.jar=40457:/usr/share/idea/bin -Dfile.encodin
Hello World!

Process finished with exit code 0
```

Bookmarks

make

Version Control Run TODO Problems Terminal Services Build

Build completed successfully in 22 sec, 869 ms (a minute ago) LF UTF-8 4 spaces

Kotlin

Fibonacci recursivo básico

```
fun main() {
    val fib = fib(15)
    println("Fibonacci number 15: $fib")
}

fun fib(n: Int): Int {
    if(n == 0) {
        return 0
    } else if (n == 1) {
        return 1
    } else {
        return fib(n-1) + fib(n-2)
    }
}
```

Kotlin

Usa when!

```kotlin
fun main() {
    val fib = fib(15)
    println("Fibonacci number 15: $fib")
}

fun fib(n: Int): Int {
    return when(n) {
        0 -> 0
        1 -> 1
        else -> fib(n-1) + fib(n-2)
    }
}
```

Kotlin

Usa notación funcional!

```kotlin
fun main() {
    val fib = fib(15)
    println("Fibonacci number 15: $fib")
}

fun fib(n: Int): Int =
    when(n) {
        0 -> 0
        1 -> 1
        else -> fib(n-1) + fib(n-2)
    }
```

Kotlin

Lanza (custom) exceptions!

```kotlin
fun main() {
    try {
        val fib = fib(15)
        println("Fibonacci number 15: $fib")
    } catch (e: NegativeNumberException) {
        println("Number was negative")
    }
}

class NegativeNumberException : RuntimeException("Negative number detected")

fun fib(n: Int): Int =
    if(n < 0) {
        throw NegativeNumberException()
    } else {
        when (n) {
            0 -> 0
            1 -> 1
            else -> fib(n - 1) + fib(n - 2)
        }
    }
```

Kotlin

Usa estilo funcional!

```
fun main() {
    try {
        val fib = fib(15)
        println("Fibonacci number 15: $fib")
    } catch (e: NegativeNumberException) {
        println("Number was negative")
    }
}

class NegativeNumberException : RuntimeException("Negative number detected")

fun fib(n: Int): Int =
    if(n < 0) {
        throw NegativeNumberException()
    } else {
        (0..n).fold(Pair(0, 0)) { acc, i ->
            when (i) {
                1 -> Pair(0, 1)
                else -> {
                    val minusTwo = acc.first
                    val minusOne = acc.second
                    Pair(minusOne, minusOne + minusTwo)
                }
            }
        }.second
    }
```

Kotlin

O usa estilo imperativo!

```kotlin
fun main() {
    try {
        val fib = fib(15)
        println("Fibonacci number 15: $fib")
    } catch (e: NegativeNumberException) {
        println("Number was negative")
    }
}

class NegativeNumberException : RuntimeException("Negative number detected")

fun fib(n: Int): Int =
    if(n < 0) {
        throw NegativeNumberException()
    } else {
        var a = 0
        var b = 0
        for(i in 0 .. n){
            if(i == 1) {
                a = 0
                b = 1
            } else {
                val sum = a + b
                a = b
                b = sum
            }
        }
        b
    }
```

Kotlin

Usa clases de Java!

```kotlin
import java.util.Scanner

fun main(args: Array<String>) {
    try {
        val scanner = Scanner(System.`in`)
        val fib = fib(scanner.nextInt())
        println("Fibonacci number 15: $fib")
    } catch (e: NegativeNumberException) {
        println("Number was negative")
    }
}

class NegativeNumberException : RuntimeException("Negative number detected")

fun fib(n: Int): Int =
    if(n < 0) {
        throw NegativeNumberException()
    } else {
        var a = 0
        var b = 0
        for(i in 0 .. n) {
            if(i == 1) {
                a = 0
                b = 1
            } else {
                val sum = a + b
                a = b
                b = sum
            }
        }
        b
    }
```

Kotlin

Null-safety

Quizá la diferencia más relevante entre Java y Kotlin sea el tratamiento de NULL. En Java cualquier tipo (salvo primitivos) puede tomar el valor NULL. En Kotlin solo los tipos nulables pueden tomar el valor NULL. El compilador además comprueba que nunca accedemos a algo con NULL.

¡No existen los NullPointerException!

```kotlin
fun main(args: Array<String>) {
    var list: NodeList? = NodeList(
        value = "hola",
        next = NodeList("gente", NodeList("de", NodeList("GDG", null)))
    )

    // Dentro del while list pasa a ser de tipo NodeList
    while(list != null) {
        print("${list.value} ")
        list = list.next
    }
}


class NodeList(value: String, next: NodeList?) {
    val value = value
    val next = next
}
```

Kotlin

```kotlin
fun main(args: Array<String>) {
    val msg1 = "Hola"
    val msg2 = "Adios"
    val msg3 = null

    val list = listOf(msg1, msg2, msg3)

    list.forEach {
        val n = stringLength(it)
        // ERROR
        // println(10 + n)
        println(10 + (n ?: 0))
    }
}

// ERROR
//fun stringLength(str: String?): Int? =
//    str.length

fun stringLength(str: String?): Int? =
    if(str != null) {
        str.length
    } else {
        null
    }

fun stringLength2(str: String?): Int? =
    str?.length

fun stringLength3(str: String?): Int? =
    str?.let {
        println("Doing stuff!")
        it.length
    }
```

Kotlin

Inmutable por defecto

val es inmutable. var es mutable. Kotlin nos anima a usar inmutabilidad. Menos cambios de estado impredecibles → Software con menos bugs.

Tanto val como var tienen inferencia de tipos pero el tipado sigue siendo estático como Java

Kotlin

```kotlin
fun main(args: Array<String>) {
    val n: Int = 5
    // ERROR
    // n = n + 1
    var m: Int = 5
    m = m + 1

    val list = listOf(n, m)
    // ERROR
    // list.add(7)
    val mList = mutableListOf(n, m)
    mList.add(7)
}
```

Kotlin

Lambdas

Podemos crear funciones anónimas con llaves.
Hay dos mejoras ergonómicas:

- Si la función necesita una función en el último argumento, se puede abrir el lambda después del paréntesis
- Si la función lambda tiene un solo argumento (caso muy común) se crea una variable predefinida llamada **it**.

Kotlin

```kotlin
fun main(args: Array<String>) {
    val add = { a: Int, b: Int ->
        a + b
    }
    val result = add(12, 40)
    println(result)

    val nums = listOf(1,2,3,4,5)
    val result2 = nums.map { it*2 }
    println(result2)
}
```

Kotlin

Mejoras en OOP

Kotlin no abandona el paradigma de Java de clases pero añade pequeñas mejoras y ligeros cambios. A destacar:

- No existe la relación 1 clase = 1 archivo
- Constructores automáticos
- Distintos defaults (clases por defecto son final, el acceso público no se marca, …)
- Data class (implementa por nosotros métodos como equals o toString)

```
fun main(args: Array<String>) {
    val car = Car(
        manufacturer = "Renault",
        model = "21",
        year = 1991
    )
    car.getYear()
}

class Car(manufacturer: String, model: String, year: Int) {
    private val manufacturer = manufacturer
    private val model = model
    private val year = year

    fun getYear() = year
}

class Car2(
    private val manufacturer: String,
    private val model: String,
    private val year: Int,
)
```

Kotlin

```kotlin
class Car3(
    private val manufacturer: String,
    private val model: String,
    year: Int
) {
    private val year = year + 1000
}

class Car4(
    private val manufacturer: String,
    private val model: String,
    private val year: Int,
) {
    init {
        println("Created class for car $model")
    }
}
```

```kotlin
data class Car5(
    val manufacturer: String,
    val model: String,
    val year: Int
) {
    init {
        println("Created data class for car $model")
    }
}

data class Car6(
    val manufacturer: String,
    val model: String,
    val year: Int,
) {
    constructor() : this("", "", 0)
}
```

```kotlin
sealed class Vehicle(private val year: Int) {
    open fun saySomething() {
        println("BRUM BRUM")
    }
}
class Car7(private val manufacturer: String, private val model: String, year: Int) : Vehicle(year)
class Bike(private val manufacturer: String, year: Int) : Vehicle(year) {
    override fun saySomething() {
        println("<SILENCE> <SILENCE>")
    }
}

fun brumVehicles() {
    val vehicles: List<Vehicle> = listOf(
        Car7("Renault", "21", 1991),
        Bike("Orbea", 2014),
        // Vehicle(1800) -> ERROR
    )

    vehicles.forEach {
        it.saySomething()
    }
}
```

Kotlin

Extension Methods y Companion

```kotlin
fun Car5.printCar() {
    println(this.manufacturer + this.model + this.year)
}
```

```kotlin
data class Point(
    val x: Int,
    val y: Int,
) {
    companion object {
        const val AUTHOR = "Point Inc."
    }
}
```

Kotlin

Corutinas

Una de las grandes mejoras de Kotlin, aunque más avanzadas, es el soporte a la concurrencia estructurada mediante corutinas.

Las corutinas son “hilos virtuales” que pueden ser parados y ejecutados por un hilo real indistintamente.

Un caso común: las corutinas nos permiten aprovechar mejor los recursos de la CPU cuando tenemos bloqueos por I/O (disco, red, …)

Kotlin

```kotlin
import kotlinx.coroutines.*

fun main(args: Array<String>) = runBlocking {
    launch {
        delay(1000L)
        println("World")
    }
    println("Hello")
}
```

```kotlin
import kotlinx.coroutines.*

fun main(args: Array<String>) = runBlocking {
    launch {
        doSomething()
    }
    println("Hello")
}

suspend fun doSomething() {
    delay(1000L)
    println("World")
}
```

Kotlin

```kotlin
import kotlinx.coroutines.*

fun main(args: Array<String>) = runBlocking {
    println("START")
    val a = async {
        doSomething1()
    }
    val b = async {
        doSomething2()
    }

    val nums = awaitAll(a, b)
    println(nums.sum())
}

suspend fun doSomething1(): Int {
    delay(5000L)
    return 50
}

suspend fun doSomething2(): Int {
    delay(5000L)
    return 100
}
```

Kotlin

```kotlin
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*

fun simple(): Flow<Int> = flow {
    for (i in 1..3) {
        delay(100)
        emit(i)
    }
}

fun main() = runBlocking {
    launch {
        for (k in 1..3) {
            println("I'm not blocked $k")
            delay(100)
        }
    }
    simple().collect { value -> println(value) }
}
```

Kotlin

```kotlin
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.Channel

fun main() = runBlocking {
    val channel = Channel<Int>()
    launch {
        for (x in 1..5) {
            delay(1000L)
            channel.send(x * x)
        }
    }

    repeat(5) { println(channel.receive()) }
    println("Done!")
}
```

Kotlin

FIN