Kotlin

http://bit.ly/KotlinCalculoGitHub

https://play.kotlinlang.org/

Daniel Illescas Romero

15 de noviembre de 2018

Calculo S.A

Índice

Introducción

Qué es

- **Kotlin** es un lenguaje de propósito general, estáticamente tipado, que combina orientación a objetos con programación funcional.
- Funciona donde **Java** funcione: apps móviles, aplicaciones de escritorio, apps del lado del servidor.
- Tiene **soporte** oficial para:
  - **IntelliJ IDEA** y Android Studio
  - Eclipse
  - Maven, Gradle, Ant
  - Spring Boot
  - etc [1]

Quién está detrás

- Es un lenguaje **open-source**, disponible en GitHub y con más de 250 contribuyentes.
- **JetBrains** es el desarrollador y principal patrocinador de Kotlin.

Características interesantes

- No tiene **;**
- Null-safety
- Interoperabilidad con Java
- Claro y conciso [2]

Sintaxis y ejemplos

Hello World

- Hello world!

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

[3]

Variables y constantes

- **var** para variables, **val** para constantes
- Inferencia de tipo

```
val age = 22

// age = 100 ⇒ Error (age is constant)
// age = "Hola" ⇒ Error (type mismatch)
```

- Especificar el tipo

```
val age: Int = 22
val height: Double = 170.2
```

Variables y constantes

```
var age = 22
age = 90 // OK, age variable & Int

val name: String

if (age == 22) {
    name = "Daniel"
} else {
    name = " Name here ".trim().toLowerCase()
}
// or
val name: String =
    if (age == 22) {
        "Daniel"
    } else {
        " Name here 33 ".trim().toLowerCase()
    }
```

Variables y constantes

- Instanciar clases

```
1 val daniel = Person(name = "Daniel", age = 22)
2 daniel.run(distance = 100.k + 890)
```

Todo es un objeto

- Los literales, sean numéricos o texto, son objetos de clases que pueden directamente invocar métodos.

```
1.rangeTo(30) // or: 1..30 // [1, 30]
30.6.hashCode()
"TesT".toLowerCase()
9.div(2.0) // 4.5
9 / 2 // 4
```

Funciones

- Declaración habitual

```
fun factorial(n: Long): Long {
   if (n ⩽ 1) {
      return 1
   }
   return n * factorial(n-1)
}
```

- Declaración acortada

```
fun factorial(n: Long): Long =
   if (n ⩽ 1) 1
   else n * factorial(n-1)
```

Clases

Declaración de clase con:

- Constructor
- Propiedades
- `toString()`
- `equals(), hashCode()`
- `copy()`, etc

```
1 data class Person(val name: String, var age: Int) {
2    fun run(distance: Long) = println("Running $distance m")
3 }
```

Clases - Java

```java
import kotlin.Metadata;
import kotlin.jvm.internal.Intrinsics;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

@Metadata(
   /* ... */
)
public final class Person {
   @NotNull
   private final String name;
   private int age;

   public final void run(long distance) {
      String var3 = "Running " + distance + " m";
      System.out.println(var3);
   }

   @NotNull
   public final String getName() {
      return this.name;
   }

   public final int getAge() {
      return this.age;
   }
```

Clases - Java

```java
public final void setAge(int var1) {
    this.age = var1;
}

public Person(@NotNull String name, int age) {
    super();
    Intrinsics.checkParameterIsNotNull(name, "name");
    this.name = name;
    this.age = age;
}

@NotNull
public final String component1() {
    return this.name;
}

public final int component2() {
    return this.age;
}

@NotNull
public final Person copy(@NotNull String name, int age) {
    Intrinsics.checkParameterIsNotNull(name, "name");
    return new Person(name, age);
}
```

Clases - Java

```
52    @NotNull
      public static Person copy$default(Person var0, String var1, int var2, int var3,
            Object var4) {
54       if ((var3 & 1) ≠ 0) {
            var1 = var0.name;
56       }
         if ((var3 & 2) ≠ 0) {
58          var2 = var0.age;
         }
60       return var0.copy(var1, var2);
      }
62
      @NotNull
64    public String toString() {
         return "Person(name=" + this.name + ", age=" + this.age + ")";
66    }

68    public int hashCode() {
         return (this.name ≠ null ? this.name.hashCode() : 0) * 31 + this.age;
70    }
```

Clases - Java

```java
    public boolean equals(@Nullable Object var1) {
        if (this ≠ var1) {
            if (var1 instanceof Person) {
                Person var2 = (Person)var1;
                if (Intrinsics.areEqual(this.name, var2.name) && this.age == var2.age) {
                    return true;
                }
            }
            return false;
        } else {
            return true;
        }
    }
}
```

Clases - Polimorfismo

```
1  open class Shape
2
3  open class Polygon(val sidesAndLength: Map<Int, Int>): Shape() {
4     val sides = this.sidesAndLength.size
5  }
6
7  open class Rectangle(
8     val horizontalLength: Int,
9     val verticalLength: Int
10 ): Polygon(sidesAndLength =
11    (1..4).map {
12       it to if (it % 2 == 0) horizontalLength else
             verticalLength
13    }.toMap()
14 )
15
16 data class Square(val sidesLength: Int):
17    Rectangle(sidesLength, sidesLength)
```

Clases - Polimorfismo

```
1 val square = Square(sidesLength = 90)
2 println(square.sides)
3
4 val rectangle = square as Rectangle
5 println(rectangle.horizontalLength)
6
7 println("First side: ${square.sidesAndLength[1]}")
8 println(square.sidesAndLength) // {1=90, 2=90, 3=90, 4=90}
```

Enum class & when

```
enum class Color(val value: String) {
    RED("FF0000"), GREEN("00FF00"), BLUE("0000FF")
}

fun main(args: Array<String>) {

    val myColor = Color.RED

    when (myColor) {
        Color.RED → println(Color.RED.value)
        Color.GREEN, Color.BLUE → println("Green or blue")
    }
}
```

when con rangos

```
val number = 100

when (number) {
    in 0..50 → println(" ... ")
    !in 80..200 → println(" ... ")
    in 100..200 → println("yes")
    else → {
        println("nope")
    }
}
```

For, Map, Range, List

```kotlin
val names = listOf("Daniel", "John", "Jack")
for (name in names) {
    println(name)
}

val nameAgeMap = mapOf("Daniel" to 22, "Michael" to 44)
println(nameAgeMap) // {Daniel=22, Michael=44}

for (i in 0 until 100) { /* ... */ }
for (i in 0..100) { /* ... */ }
for (i in 1..15 step 2) { /* ... */ }
for (i in 10 downTo 0) { /* ... */ }
//
val (name, age) = Person(name = "Daniel", age = 22)
Random.nextInt(0, 10) // kotlin.random
```

Interoperabilidad

Kotlin en Java

```java
import org.jetbrains.annotations.NotNull;
import java.util.List;

public class People {

    private final List<Person> list;

    public List<Person> getList() {
        return this.list;
    }

    People(@NotNull List<Person> people) {
        this.list = people;
        this.list.get(0).getName();
    }
}
```

Java en Kotlin

```kotlin
val danielAndJohn = listOf(
    Person(name = "Daniel", age = 22),
    Person(name = "John", age = 30)
)
val people = People(danielAndJohn) // people = ... NOT allowed
println(people.list[0].name)
```

Avanzado

Optional

- Para tipos que pueden ser nulos, usamos tipos opcionales, representados con el símbolo ?

```
var abc: String = "abc"
abc = null // compilation error

val daniel = Person(name = "Daniel", age = 22)
val john = Person(name = "John", age = 44)

val idPerson = mapOf(0 to daniel, 10 to john)
println(idPerson[0]?.name ?: "unknown") // Elvis operator

// optional value
var maybeDaniel: Person? = idPerson[0]
maybeDaniel = null // oK
```

Extensions

- Cuando queramos añadir funcionalidades a clases existentes sin herencia.

```
enum class Color(val rgb: Int) {
   RED(0xFF0000), GREEN(0x00FF00), BLUE(0x0000FF);
   val hexString = this.rgb.toHex()
}

fun Int.toHex(): String {
   return java.lang.Integer.toHexString(this)
}

fun String.toColor() = Color.valueOf(this.toUpperCase())

val Int.M: Long
   get() {
      return this.toLong() * 1_000_000
   }
val Int.k get() = this.toLong() * 1_000
```

Sobrecarga de operadores

- Para añadir operadores

```
operator fun String.times(n: Int): String {
    return (0 until n).joinToString(separator = "") { this }
}

operator fun String?.not(): Boolean {
    return this == null || this.isEmpty()
}

fun main(args: Array<String>) {
    println("Hello" * 3) // HelloHelloHello
    val name = " ".trim()
    if (!name) {
        /* ... */
    }
}
```

Lambdas & programación funcional

```
fun calculate(a: Int, b: Int, operation: (Int, Int) → Int): Int {
    return operation(a, b)
}

val result = calculate(10, 20) { a, b → a * b }
```

```
val sumOfLargeNamesLength =
    listOf("Daniel", "John", "Juan!")
        .filter { it.length > 4 }
        .map { it.length }
        .reduce { accumulated, next → accumulated + next}

println(sumOfLargeNamesLength) // 11
```

Coroutines

- Operaciones en paralelo de forma asíncrona.

```
1  import kotlinx.coroutines.*
2  import kotlin.system.measureTimeMillis
3
4  suspend fun doingSomethingHeavy(): Int {
5      delay(1000L)
6      return 42
7  }
8  suspend fun doingSomethingHeavy2(): Int { /* */ }
9
10 fun main() = runBlocking {
11     val time = measureTimeMillis {
12         val result1 = async { doingSomethingHeavy() }
13         val result2 = async { doingSomethingHeavy2() }
14         println("Results: ${result1.await() +
               result2.await()}") // 462
15     }
16     println("Completed in: ${time.div(1000)}s") // 1s
17 }
```

Bibliografía

Bibliografía i

- Kotin 1.0 release - jetbrains blog.
  URL: `https://bit.ly/2dXuodq`.
- Kotlin - github.
  URL: `https://github.com/JetBrains/kotlin`.
- Kotlin - reference.
  URL: `https://kotlinlang.org`.

Bibliografía ii

Recomendaciones:

- https://kotlinlang.org
- http://devdocs.io
- https://zealdocs.org
- https://kotlinlang.org/community/
- https://github.com/illescasDaniel/Kotlin-Slides