Kotlin: getting started

Dario Pellegrini
iOS & Android developer @s4win
info@dariopellegrini.com

Student Branch IEEE University of Brescia
Università degli Studi di Brescia
S4WIN strategie digitali

Di cosa si tratta

- Multiparadigma
- Fortemente tipizzato
- 100% interoperabilità Java
- Multipiattaforma
- Open source (https://github.com/JetBrains/kotlin)

Multiplatform

- Android (Kotlin JVM)
- Backend (Kotlin JVM)
- JavaScript (Kotlin JS)
- iOS, macOS, Windows, Linux (Kotlin Native beta)

Control flow

- if
- while
- for
- when

Strings

```
val name = "John"
println("Hello $name")
println("Name length ${name.length}”)
```

```
println("""
    <html>
    <head>
    <title>Hello</title>
    </head>
    <body><h1>Hello</h1></body>
    </html>
“"".trimIndent())
```

```
println("%d, %s, %.6f : %.3f".format(1, "Hello", 3.57897987987, 3.9879879879879879879))
```

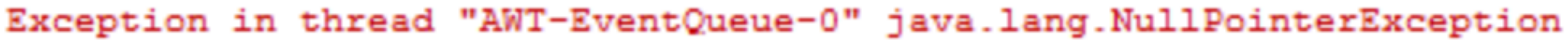
Range

```
(1..10).forEach { print(it) }

(10 downTo 0).forEach { print(it) }

(1..10 step 2).forEach { print(it) }

(1 until 10).forEach { print(it) }
```

Type inference

```
val a = "abc"                                       // Tipo String dedotto
val b = 4                                           // Tipo Int dedotto

val c: Double = 0.7                                 // Tipo dichiarato esplicitamente
```

Null Safety



```
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
```

Null Safety

```kotlin
var name: String = "Hello"
name = null // null cannot be a value of non-null variable

var name: String? = "Hello"
name = null // OK


println(name?.hashCode()) // Prints null if name is null


val l = name?.length ?: -1 // If name is null return name.length else -1
print(l)


// Safe cast
val ageString: String? = age as? String


// Meglio evitarlo
print(name!!.length)
```

Smart cast

```
if (obj is String) {
    print(obj.toUpperCase())     // obj qui è riconosciuto come String
}


fun printIfNotNull(message: String?) {
    if (message != null) {
        print(message.length)     // message qui è riconosciuto come not null
    }
}
```

Collections

```kotlin
val numbers = listOf(1, 2, 3)

val strings = setOf("a", "b", "c", "c")

val map = mapOf("key1" to 1, "key2" to 2)
```

Classi

- Creazione di classi e costruttori meno verbose che in Java
- Niente metodi statici (sostituiti da package functions o companion object)
- Valori di default nel costruttore
- Arguments naming
- init function

Classi

```
class Spaceship(id: String,
                val name: String,
                val cFactor: Float,
                var description: String,
                var notes: String? = null) {
     init {
         . . .
     }
}
```

```
val spaceship = Spaceship("12345","Enterprise", 10.5f, "Exploration ship")


val spaceship = Spaceship(
        id = "12345",
        cFactor = 10.5f,
        name = "Enterprise",
        notes = "Various notes",
        description = "Exploration ship")
```

Data class

```
data class Spaceship(val id: String,
                     val name: String,
                     val cFactor: Float,
                     val description: String,
                     val notes: String? = null)

// equals()/hashCode()
// toString()
// Deconstructive declarations support
// copy()
```

Equals

```
val john1 = Person("John")
val john2 = Person("John")
john1 == john2    // true uguaglianza strutturale
john1 === john2   // false uguaglianza di reference
```

Deconstrictive declarations

```
val spaceship = Spaceship(“12345",
                          “Enterprise",
                          10.5f,
                          “Exploration ship. Captain: James T: Kirk")
val (_, name, cFactor, description) = spaceship
println(name)
println(cFactor)
println(description)
```

```
map.entries.forEach { (key, value) ->
    print("$key: $value")
}
```

STUDENT BRANCH IEEE UNIVERSITY OF BRESCIA
UNIVERSITÀ DEGLI STUDI DI BRESCIA
S4WIN strategie digitali

Properties

```kotlin
val fasterThanLight: Boolean
    get() = cFactor > 1


var captain: String? = null
    set(value) {
        println("Captain has changed from $field to $value")
        field = value
    }
```

Extensions

```kotlin
fun Spaceship.canDock(): Boolean {
    return allowedIds.contains(this.id)
}

if (spaceship.canDock()) {
    print("Clear to proceed")
} else {
    print("Spaceship not allowed")
}


val String.date: Date?
    get() {
        val format = SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'", Locale.ENGLISH)
        return try {
            format.parse(this)
        } catch (e: Exception) {
            null
        }
    }

val date = "2018-12-05T10:30:00.000Z".date
```

Funzioni

```
fun double(x: Int): Int {
    return 2 * x
}

fun double(x: Int): Int = x * 2



val handler: (String?) -> Boolean = { message ->
    message != null
}
```

Funzioni

```
fun waitAndDo(millis: Long, closure: () -> Unit) {
    Thread {
        Thread.sleep(millis)
        closure()
    }.start()
}



waitAndDo(2000, {
    print("Hello")
})



waitAndDo(2000) {
    print("Hello")
}
```

Funzioni

```
fun getFromNetwork(closure: (Int, String?) -> Unit) {
    Thread {
       try {
              val results = get(“http://baseurl/endpoint")
              closure(200, results)
       } catch(e: Exception) {
              closure(400, null)
       }
    }.start()
}


 getFromNetwork { statusCode, result ->
     when(statusCode) {
         in (200..299) -> println("Success: ${result}")
         else -> println("Error")
     }
 }
```

Funzioni

```kotlin
val list = listOf(1, 2, 3, 4, 5, null, 6, null, null, null, 7, 8, 9, 10)
val finalString = list.filterNotNull()
                      .filter { it % 2 == 0 }
                      .map { "$it" }
                      .reduce { acc, s -> "$acc - $s" }
print(finalString) // 2 - 4 - 6 - 8 - 10
```

Classi e Funzioni

```kotlin
class SpaceGun(init: SpaceGun.() -> Unit) {
    var id: String = ""
    var name: String? = null
    var type: String? = null
}


val gun = SpaceGun {
    id = "123"
    name = "Boom"
    type = "Rifle"
}
```

Infix functions

```
infix fun List<String>.merge(list: List<String>): List<String> {
    val mutableList = this.toMutableList()
    mutableList.addAll(list)
    return mutableList
}

val l1 = listOf("A", "B", "C")
val l2 = listOf("1", "2", "3")
val lm = l1 merge l2
print(lm) // [A, B, C, 1, 2, 3]
```

Sealed class

```kotlin
sealed class Status
data class Approaching(val velocity: Double): Status()
data class Leaving(val spaceship: Spaceship): Status()
data class Docking(val spaceship: Spaceship, val dockNumber: Int): Status()
data class NoMovement(): Status()


. . .
// status from stream service

when(status) {
    is Approaching -> if (status.velocity > MAX_VELOCITY) print(“Alarm!”)

    is Leaving     -> print("${status.spaceship.name} is leaving”)

    is Docking     -> {
        if (isFriend(status.spaceship)) {
            print("${status.spaceship.name} can proceed to dock number ${status.dockNumber}")
        } else {
            print("${status.spaceship.name} in an enemy. Attack")
        }
    }

    is NoMovement -> print("Nothing in sight")
}
```

Singleton Pattern - Java

```java
public class Singleton {

  private final static Singleton instance= new Singleton();

  private Singleton() {}

  public static Singleton getInstance() {
    return instance;
  }
}
```

Singleton pattern - Kotlin

```kotlin
object Singleton {
    var value: String? = null
}
```

Observer pattern - Java

```java
public class TypeChangeListener {
    public void onValueChanged(String newValue) {
        System.out.println("Type has changed to " + newValue);
    }
}

public class ObservableObject {
    private TypeChangeListener typeChangeListener;
    private String name;
    private String type;

    public ObservableObject(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public TypeChangeListener getTypeChangeListener() {
        return typeChangeListener;
    }

    public void setTypeChangeListener(TypeChangeListener typeChangeListener) {
        this.typeChangeListener = typeChangeListener;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
        typeChangeListener.onValueChanged(type);
    }
}
```

Observer pattern - Kotlin

```kotlin
class Planet(val name: String, val capital: String ) {

    private var observerTypeFunction: ((String) -> Unit)? = null

    var type: String by Delegates.observable("") { prop, old, new ->
        observerTypeFunction?.invoke(new)
    }
}


val planet = Planet("Mars")
planet.observerTypeFunction = {
    print("Type changed to $it")
}
planet.type = "Red planet"
```

Strategy pattern - Java

```java
public interface Strategy {
   public int doOperation(String text);
}
```

```java
public class OperationUpperCase implements Strategy{
   @Override
   public int doOperation(String text) {
      return text.toUpperCase();
   }
}
```

```java
public class OperationLowerCase implements Strategy{
   @Override
   public int doOperation(String text) {
      return text.toLowerCase();
   }
}
```

```java
public class Context {
   private Strategy strategy;

   public Context(Strategy strategy){
      this.strategy = strategy;
   }

   public int executeStrategy(String text){
      return strategy.doOperation(text);
   }
}
```

```java
Context context = new Context(new OperationUpperCase());
System.out.println(context.executeStrategy(“Hello”));

context = new Context(new OperationLowerCase());
System.out.println(context.executeStrategy(“Hello”));
```

Strategy pattern - Kotlin

```
class Printer(val printStrategy: (String) -> (String)) {
    fun printString(toBePrint: String) {
        println(printStrategy(toBePrint))
    }
}



val lowerCasePrinter = Printer {
    it.toLowerCase()
}
lowerCasePrinter.printString("Hello") // hello

val upperCasePrinter = Printer {
    it.toUpperCase()
}
lowerCasePrinter.printString("my friend") // MY FRIEND
```

Builder pattern - Java

```java
class Pizza
{
    private String dough = "";
    private String sauce = "";
    private String topping = "";

    public void setDough(String dough)
    { this.dough = dough; }
    public void setSauce(String sauce)
    { this.sauce = sauce; }
    public void setTopping(String topping)
    { this.topping = topping; }
}
```

```java
/** "Abstract Builder" */
abstract class PizzaBuilder
{
    protected Pizza pizza;

    public Pizza getPizza()
    {
        return pizza;
    }
    public void createNewPizzaProduct()
    {
        pizza = new Pizza();
    }

    public abstract void buildDough();
    public abstract void buildSauce();
    public abstract void buildTopping();
}

/** "ConcreteBuilder" */
class HawaiianPizzaBuilder extends PizzaBuilder
{
    public void buildDough()
    {
        pizza.setDough("cross");
    }
    public void buildSauce()
    {
        pizza.setSauce("mild");
    }
    public void buildTopping()
    {
        pizza.setTopping("ham+pineapple");
    }
}

/** "ConcreteBuilder" */
class SpicyPizzaBuilder extends PizzaBuilder
{
    public void buildDough()
    {
        pizza.setDough("pan baked");
    }
    public void buildSauce()
    {
        pizza.setSauce("hot");
    }
    public void buildTopping()
    {
        pizza.setTopping("pepperoni+salami");
    }
}
```

```java
pizzaBuilder.createNewPizzaProduct();
pizzaBuilder.buildDough();
pizzaBuilder.buildSauce();
pizzaBuilder.buildTopping();
```

Build Pattern - Kotlin

```kotlin
class Pizza(var dough: String? = null,
            var sauce: String? = null,
            var toppings: String? = null)

fun buildPizza(closure: Pizza.() -> Unit): Pizza {
    val pizza = Pizza()
    pizza.closure()
    return pizza
}


val pizza = buildPizza {
    dough = "Baked"
    sauce = "Normal"
    toppings = "Ham"
}

val bigPizza = buildPizza {
    dough = "Double baked"
    sauce = "Double tomato"
    toppings = "Ham, Mushrooms"
}
```

Builder pattern - Kotlin

```kotlin
class SpaceGun() {
    var id: String = ""
    var name: String? = null
    var type: Type? = null
}

class Type() {
    var name: String? = null
    var number: Int? = null
}
```

```kotlin
fun spaceGun(init: SpaceGun.() -> Unit): SpaceGun {
    val spaceGun = SpaceGun()
    spaceGun.init()
    return spaceGun
}

fun type(init: Type.() -> Unit): Type {
    val type = Type()
    type.init()
    return type
}
```

```kotlin
val spaceGun = spaceGun {
    id = "123"
    name = "Boom"
    type {
        name = "Rifle"
        number = 1
    }
}
```

Altro

- Coroutine
- Observable
- Delegation
- Generics
- Operator overloading
- DSL
- Type alias
- Enum

Grazie per l'attenzione

Codice sorgente app Android e slide disponibili

https://github.com/dariopellegrini/CISB2018

STUDENT BRANCH IEEE UNIVERSITY OF BRESCIA

UNIVERSITÀ DEGLI STUDI DI BRESCIA

S4WIN strategie digitali