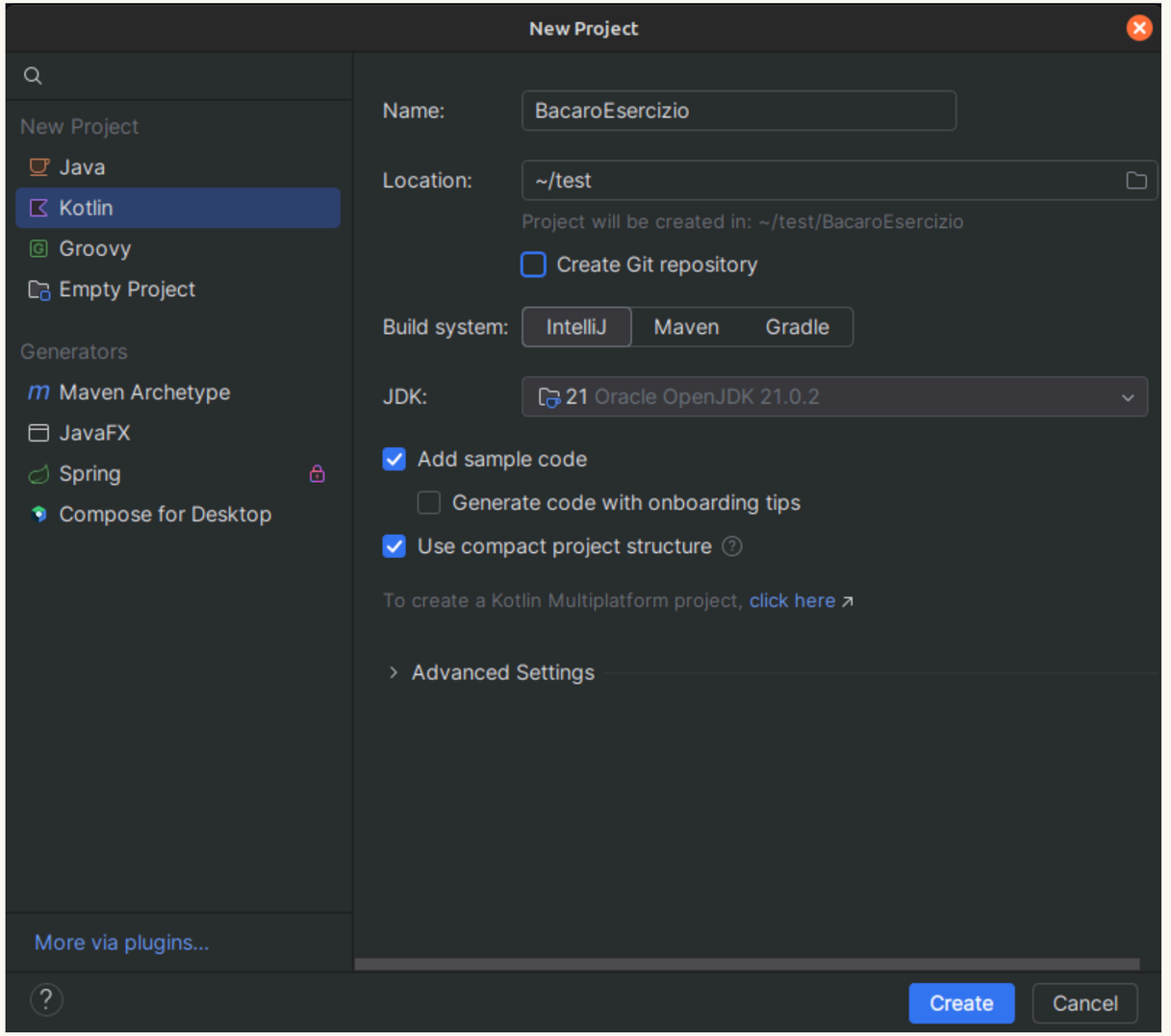
Presentata da Moreno Frigo Turco
Supportata dal team di Bacaro Tech

Uno sguardo a Kotlin!

Programma

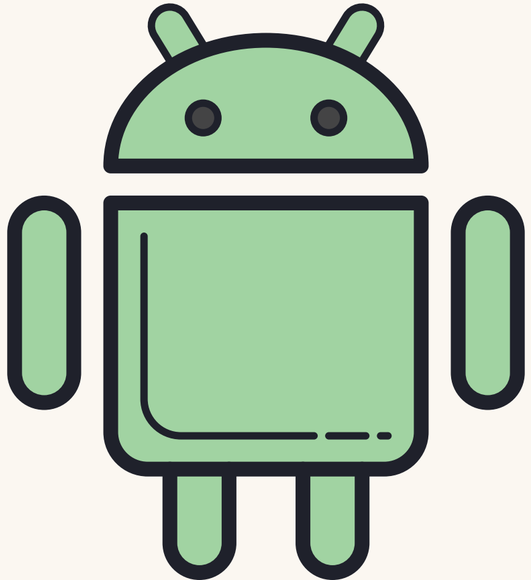
- Introduzione del linguaggio
- Versatilità del linguaggio
- Sintassi di base
- Esercizio pratico (richiede IntelliJ)

Configurazione esercizio



Cos’è Kotlin?

Conosciuto principalmente per lo sviluppo Android, Kotlin è un linguaggio moderno e versatile che sfrutta al massimo la potenza della Java Virtual Machine (JVM).



Kotlin o Java?

Sicuramente, Java è il linguaggio più noto che sfrutta la JVM, ma è altrettanto conosciuto per alcuni suoi punti deboli.

Confrontiamo dunque i due linguaggi per valutarne le differenze.

- Sintassi moderna
- Flessibilità
- Versatilità
- Affidabilità

- Maturità
- Rigidità
- Community
- Sistemi Legacy

Esempio

```kotlin
fun main() {
    val world = "World"
    println("Hello $world!")
}
```

```java
public static void main(String[] args){
    String world = "World";
    System.out.println("Hello " + world + "!");
}
```

Kotlin farà morire Java?

Come abbiamo visto, Kotlin sembra avere tutte le carte in regola per sostituire Java. Dobbiamo quindi abbandonare quest'ultimo?

Assolutamente NO!

Nonostante Kotlin offra un'esperienza di sviluppo notevolmente più comoda e potente, è fondamentale riconoscere l'importanza storica di Java e il suo ampio ecosistema (es. COBOL).

Versatilità

Come accennato in precedenza, Kotlin è un linguaggio estremamente versatile. Ma cosa lo rende tale?

La sua flessibilità e modernità rappresentano sicuramente un grande vantaggio, rendendo il codice più chiaro e accessibile anche ai meno esperti.

Scopriamo insieme in quali contesti Kotlin può essere utilizzato.

Contesti

- Android Development
- Backend
- Cross-platform
- Kotlin/JS
- Kotlin/WASM

Android development

Dal 2019, Kotlin è diventato il linguaggio ufficiale per lo sviluppo Android, sostituendo Java.

Questo dato, confermato anche sul sito ufficiale di Kotlin, dimostra l'evoluzione di questo linguaggio e come, secondo il 70% degli sviluppatori che lo utilizzano, esso aumenti la loro produttività.

Backend

Sebbene non sia ancora così diffuso, Kotlin sta guadagnando terreno anche nello sviluppo backend.

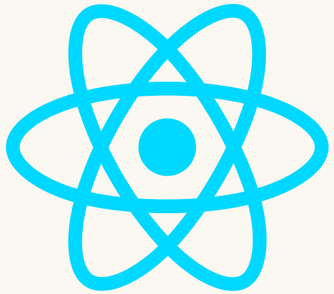
La sua interoperabilità con Java gli consente di sfruttare framework consolidati come Spring Boot.

Tuttavia, per chi preferisce rimanere nel mondo Kotlin puro, c'è Ktor, un framework più leggero e meno automatico rispetto a Spring Boot, che offre maggiore controllo e flessibilità.

Cross-platform

Dalla fine del 2023, Kotlin ha fatto il suo ingresso nel mondo cross-platform con il rilascio della prima versione stabile di Kotlin Multiplatform (KMM).

Questo consentirà agli sviluppatori di utilizzare lo stesso codice su diverse piattaforme, creando così un ecosistema completamente basato su Kotlin e migliorando l'efficienza nello sviluppo di applicazioni multipiattaforma.



Kotlin/JS

Kotlin/JS è uno strumento che permette agli sviluppatori di convertire il codice Kotlin in JavaScript.

Grazie a questo supporto, è possibile creare applicazioni frontend utilizzando Kotlin, con la possibilità di integrare framework come React e Angular.
Inoltre, è possibile sviluppare anche il backend sfruttando Node.js e importare framework come Express, offrendo così un'ampia flessibilità nello sviluppo full-stack.

Kotlin/WASM

Kotlin/WASM è uno strumento ancora in fase alpha, ma con un grande potenziale.

Attraverso l'integrazione con Jetpack Compose Multiplatform, permette agli sviluppatori di riutilizzare i componenti UI su tutte le piattaforme.

Sebbene non sia ancora consigliato per applicazioni reali, dare un'occhiata preliminare può essere molto utile per prepararsi al rilascio della prima versione stabile.

Sintassi base

Dopo aver introdotto brevemente il mondo di Kotlin e la sua versatilità, che consente di creare un intero ecosistema basato su questo linguaggio, è arrivato il momento di esplorare insieme alcune delle sue basi sintattiche.

Vediamo quindi la sintassi fondamentale, per fornire a tutti una panoramica generale del suo funzionamento.

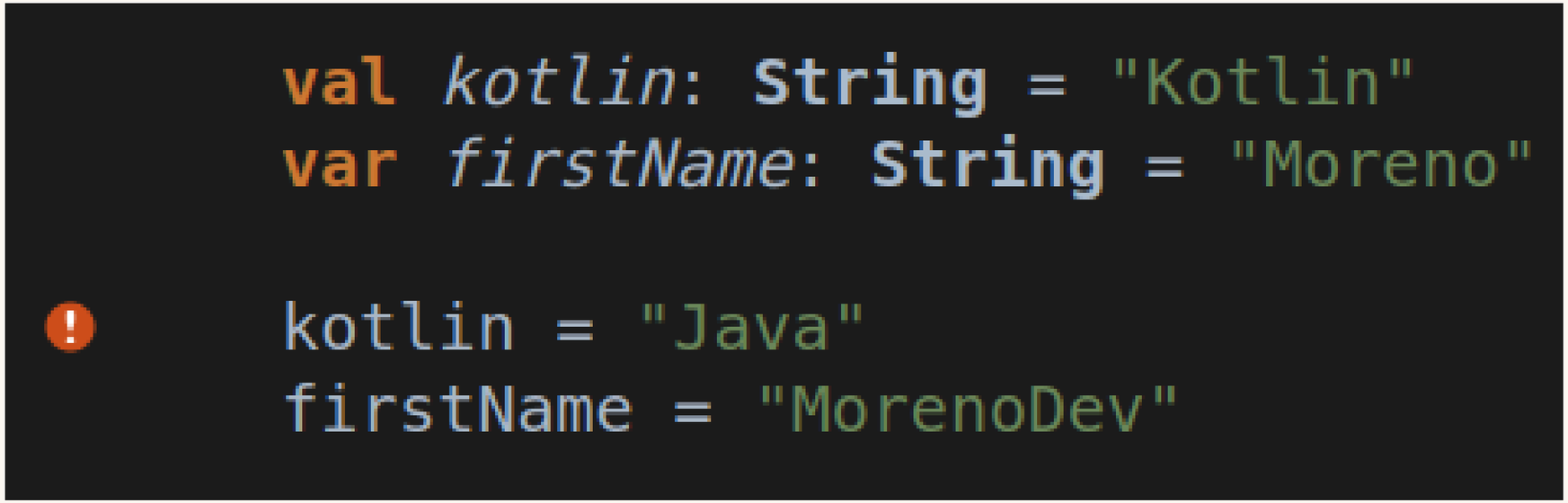
Per testare il codice, possiamo usare il compilatore online.

Tipi principali

- **Int: numeri interi**
- **Float: numeri decimali**
- **String: testo**
- **Boolean: booelani (vero/falso)**
- **List: lista di elementi**

Variabili

- **var: variabili che cambiano valore**
- **val: variabili che non cambiano valore**



Condizioni

- if (else if/else)
- when

If

L'if in Kotlin funziona allo stesso modo degli altri linguaggi di programmazione: valuta una condizione e, in base al risultato, esegue il codice corrispondente.

```
val n1: Int = 10
val n2: Int = 10

if (n1 == n2) println ("n1 ($n1) and n2 ($n2) are equal")
else println ("n1 ($n1) and n2 ($n2) are not equal")
```

When

Il when di Kotlin è simile allo switch di altri linguaggi, ma è molto più potente e permette di gestire più condizioni in modo più compatto, leggibile ed elegante rispetto a una serie di else if.

```kotlin
val x: Int = 20

when (x) {
    10 -> println("x ($x) value is 10")
    20 -> println("x ($x) value is 20")
    else -> println("x ($x) value is not present in this condition block")
}
```

Cicli

- for
- (do) while
- repeat

For

Come in altri linguaggi, il for di Kotlin itera su un range di valori.

```kotlin
for(i in 1..10){
    println(i)
}
```

```kotlin
for (i in 10 downTo 1){
    println(i)
}
```

(Do) While

Come in altri linguaggi, il ciclo while di Kotlin esegue il suo corpo finché la condizione è vera, (con il dogarantisce che il corpo venga eseguito almeno una volta, indipendentemente dalla condizione).

```kotlin
var text: String = ""
while(!text.equals("exit", ignoreCase = true)){
    print("Text something: ")
    text = readln()
    println("Inserted text: $text")
}
println("App terminated.")
```

```kotlin
var text: String
do {
    print("Text something: ")
    text = readln()
    println("Inserted text: $text")
} while(!text.equals("exit", ignoreCase = true))
println("App terminated.")
```

Repeat

Analogamente al ciclo for, la funzione repeat in Kotlin esegue il suo corpo un numero specificato di volte.

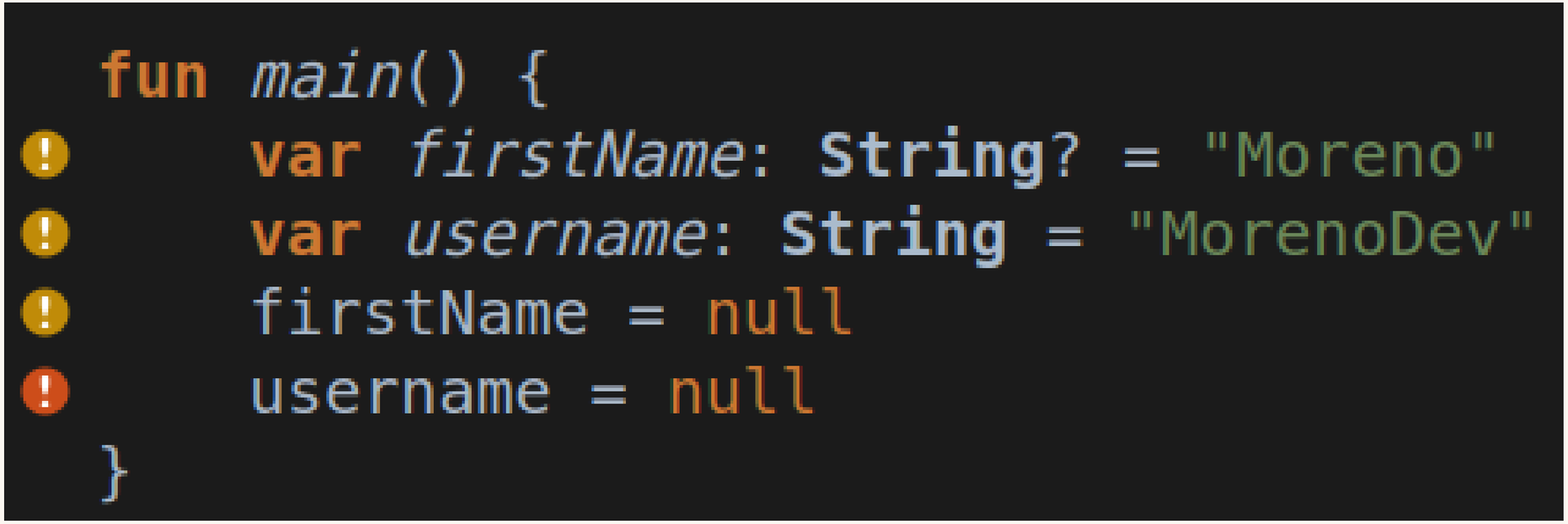
```
repeat(5) {
    println(it)
}
```

Funzionalità avanzate

- Null Safety
- Coroutines
- Sealed Classes/Interfaces
- Data Classes
- Extension functions
- Smart casts

Null Safety

Uno degli strumenti più potenti in Kotlin è la null safety: basta aggiungere l'operatore ? a una variabile per indicare che può essere null, prevenendo i NullPointerException.



Elvis operator

In aggiunta alla null safety, l'Elvis operator (?:) permette di definire un valore predefinito o un'azione da eseguire nel caso in cui il valore sia null.

```
fun main() {
    greet()
    greet("Moreno")
}

fun greet(name: String? = null) {
    name?.let{
        println("Hello, $it")
    } ?: println("Hello")
}
```

Esercizio

Siamo giunti alla fase finale, ed è il momento di mettere in pratica ciò che abbiamo visto nelle slide precedenti.

L'esercizio consiste nel creare una mini applicazione in cui l'utente inserisce un valore numerico tramite la console e tenta, entro 10 tentativi, di indovinare un numero casuale generato tra 1 e 100.

La sfida principale è provare a usare Kotlin, ma sentitevi liberi di utilizzare il linguaggio che preferite!
(Soluzione su Github)

Ringraziamenti

Volevo ringraziare ancora il team del Bacaro Tech per lo spazio concesso a queste interessanti chicche su Kotlin.

Se i contenuti proposti vi hanno suscitato interesse, vi invitiamo a farcelo sapere, così da poter organizzare un altro incontro insieme.

Buon codice a tutti! 🍷

Grazie dell’attenzione!

Per ulteriori contenuti come questo, seguitemi su