Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

KOTLIN E FIREBASE
UTILIZZI E FUNZIONALITÁ
DI UN APPLICAZIONE ANDROID

Relatore:
Chiar.mo Prof.
LUCA BEDOGNI

Presentata da:
ALESSIO KOCI

Sessione III
Anno Accademico 2017/18

Questa è la Dedica*:*
ognuno può scrivere quello che vuole,
anche nulla . . .

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Questa è l'introduzione.

Capitolo 2

Kotlin

2.1 Storia

Kotlin é un linguaggio di programmazione open-source [1], con la caratteristica di essere orientato agli oggetti, staticamente tipizzato e basato sulla JVM (Java Virtual Machine).
Lo sviluppo di Kotlin é iniziato nel 2010 dall'azienda JetBrains, conosciuta nel modo dello sviluppo software per la realizzazione di diversi IDE (Integrated development environment), tra cui: Intellij IDEA,, sul quale si basa l'attuale IDE ufficiale di Google dedicato alla programmazione Android, chiamato: Android Studio.Java é sempre stato, negli ultimi anni, uno dei linguaggii piú usati e conosciuti ma presenta diverse imperfezioni e problemi che spinse il team di JetBrains a iniziare lo sviluppo di un suo linguaggio di programmazione che prendesse in considerazione alcuni spunti e idee introdotte da linguaggi preesistenti come CSharp, Scala, Groovy, ECMAScript, Go, Python.
Nel 2015 Google prese in considerazione l'utilizzo di Kotlin come plugin per Android-Studio, e dopo vari test nel 2017 durante la conferenza Google IO (2017), arrivó l'annuncio che uficcializzava Kotlin come nuovo linguaggio di

[1] https://github.com/JetBrains/kotlin

programmazione per lo sviluppo di applicazioni Android, senza escludere e rinunciare a Java, su cui si basa attualmente l'SDK di Android.

2.2 Caratteristiche

Il linguaggio Kotlin é stato sviluppato per risolvere molti problemi tipici che riscontrano spesso programmatori Java, infatto é stato progettato e progressivamente testato per 7 anni come versione "beta" dagli stessi programmatori che lavoravano presso JetBrains, fino a raggiungere nel 2017 la versione stabile, é nato quindi all'interno di un team di sviluppatori che hanno cercato di risolvere i loro stessi problemi e non in ambito di ricerca come spesso accade.
Kotlin prendendo spunto dalle problematiche di Java e da buone regole introdotte da alcuni linguaggi imperativi e funzionali é stato modellato in modo tale da aggiungere funzionalitá utili sia a livello sintattico che a livello prestazionale, offrendo quindi al programmatore strumenti, caratteristiche e implementazioni semplici e utili ma molto potenti.
Un altro aspetto importante su cui il team di Kotlin ha prestato molta attenzione Ã¨ stata la cura e la buona integrazione del suo plugin con Android Studio. Il supporto dato dal plugin é quello di aiutare in ogni momento lo sviluppatore, consigliando tecniche di buona programmazione, abbreviazione del codice per rendere il tutto piú coinciso, conversione automatica del codice Java in Kotlin e ove possibile cercherá di allertare lo sviluppatore su possibili errori e problemi di prestazione e sintattici. Le caratteristiche piú importanti offerte da Kotlin sono l'interoperabilitá con Java, permettendo l'utilizzo di librerie Java e Kotlin simultaneamente, l'introduzione di alcune caratteristiche dei linguaggi di ordine superiore, la tipizzazione statica delle variabili, l'inferenza di tipo e soprattutto il null-safety permettendo di differenziare il tipo nullabile e il tipo non-nullabile, prevenendo quindi errori di null pointer expetions.
Il codice prodotto in Kotlin è inoltre piú compatto, coninciso e meno verboso

grazie alle dataclass, il supporto delle lambda function e altri costrutti.

2.2.1 Interoperabilitá

I linguaggi Kotlin e Java sono fortemente intercompatibili permettendo quindi a entrambi i linguaggi di coesistere all'interno dello stesso codice e di richimare funzioni e parti di codice in Java da Kotlin e viceversa, poiché entrambi i linguaggi producono Java Bytecode.
Prendiamo in considerazione il classico esempio "Hello word" scritto in Kotlin e in Java

Listing 2.1: Hello.kt in Kotlin

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

Listing 2.2: Hello.java in Java

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Il compilatore di Kotlin prendendo come input il file "Hello.kt" produrrá un JAR eseguibile da Java "Hello.jar"

Listing 2.3: Compilatore kotlin

```
$ kotlinc Hello.kt -include-runtime -d Hello.jar //
$ java -jar Hello.jar
$ Hello, world!
```

-includeruntime é un opzione del compilatore per produrre un eseguibile jar, includendo le runtime di Kotlin (800Kb)

2.2.2 Variabili

Java pone due differenze quando si parla di variabili, mette a disposizione i principali tipi primitivi (int, boolean, byte, long, short, float, double, char) e i le loro corrispondenti classi (Int, Boolean, Byte, Long, Short, Float, Double, Char). Uno dei principali cambiamenti introdotti da Kotlin é stato quello di rendere accessibile allo sviluppatore tutte le varibili come se fossero oggetti.
La differenza fra i tipi primitivi e gli oggetti sta nel fatto che i primi indicano solamente il tipo di una variabile, mentre gli oggetti incapsulano il tipo e ne aggiungono funzionalitá e metodi aggiuntivi, inoltre il tipo primitivo non puó assumere valore nullo.
Kotlin operando ad alto livello, rimuove e astrae le due distinzioni poiché di default quando viene inizializzata una nuova variabile, Kotlin la identifica come un oggetto, consentendo allo sviluppatore di utilizzare i metodi aggiuntivi ad esso associati, e solo in fase di compilazione, il compilatore di Kotlin controlla se l'oggetto é strettamente necessario o puó essere sostituito dal suo corrispondente tipo primivio.

Tipo	Oggetto	Dimensione
int	Int	32 bits
boolean	Boolean	1 bits
byte	Byte	8 bits
long	Long	64 bits
short	Short	16 bits
float	Float	32 bits
double	Double	64 bits
char	Char	16 bits

Le variabili in Kotlin sono pressocché le stesse che sono presenti in Java, con la particolariá che Kotlin cerca di evitare alcuni problemi dovuti a referenze a puntatori nulli (NullPointerException).
Kotlin richiede che una variabile a cui assegnamo un valore nullo sia dichiarata con l'operatore "?", in caso contrario mostrera un errore in fase di compilazione

Listing 2.4: Esempio

```
var esempio1: String? = null //corretto
var esempio2: String = null //errore
```

Il safe call operator "?" serve ad indicare che la variabile puó assumere in qualsiasi momento un valore nullo, e lascia al programmatore la responsabilitá e la possibilitá di accederci ugualmente per leggerne il valore, con l'utilizzo dell operatore "!!"",

Listing 2.5: Esempio !!

```
val nome = getName()!!
```

in altrnativa attraverso il Smart Casting offerto da Kotlin l'operatore "!!" si puó omettere, poiché il compilatore capisce automaticamente che la variabile non potrá essere essere nulla.

Listing 2.6: Smart Casting

```
fun getName(): String? {..}

val name = getName()
if (name != null) {
  println(name.length)
}

//forma contratta
println(name?.length)
```

Un'ultima caratteristica introdotta da Kotlin nell'utilizzo e gestione delle variabili sono "Lazy Initialization " e "Late Initialization", due nuovi modi per inizializzre una variabile.

- Lazy consente di delegare ad una funzione l'inizializzazione della variabile, il risultato della funzione verrá assegnato alla variabile, in seguito quando verrá effettuato l'accesso alla variabile la funzione non sará rieseguita ma verrá solamente passato il valore

- Late permette di posticipare l'inizializzazione di una variabile, se si tenterá di acceddere alla variabile prima che essa venga inizializzata si riceverá un errore. Late é stato principalmente introdotto per supportare la "dependency injection", ma puó essere comunque utilizzato dal programmatore per scrivere codice efficiente

2.2.3 Funzioni

Le funzioni sono definite utilizzando la parola "fun" e il tipo di protezione di default associata alla funzione Ã¨ quello "public", successivamente al nome della funzione verranno indicati i parametri opzionali e il valore di ritorno, qualora ci fosse.

Listing 2.7: Esempio Kotlin

```
fun saluta(nome: String): String {
  return "Ciao $nome"
}
```

Gli argomenti delle funzioni in Kotlin possono assumere il valore passato dal chiamante della funzione oppure avere un valore di default. Questa caratteristica oltre ad essere utile al programmatore che eviterá di inserire controlli all'interno di funzioni, o addirittura creare un'altra funzione con parametri diversi, aumenta la leggibilitá del codice, rendendolo piú diretto e comprensivo.

Listing 2.8: Esempio Kotlin

```
fun buyItem(id:String, status: color = true){...}
buyItem(23)
```

Listing 2.9: Esempio Java

```
void buyItem(String id, Boolean color){...}
buyItem(23,true);
```

Kotlin prendendo spuntoo dalla programmazione funzionale introduce anche le seguenti caratteristiche:

- Funzioni alto ordine: sono funzioni che possono accettare come parametri altre funzioni, e restituire a loro volta funzioni
- Assegnamento: é possibile assegnare ad una variale TODO()
- TODO()

Listing 2.10: Esempio Kotlin

```
fun esempio(str: String, fn: (String) -> String): Unit {
  val prova = fn(str)
  println(prova)
}
```

2.2.4 Data Classes

Nella programmazione Android molto frequentemente vengono create classi per rappresentare modelli che verranno usati dall'applicazione, questi modelli devono contenere i metodi get e set per leggere e settare i valori di un oggetto. Kotlin come Java é un linguaggio orientato agli oggetto, e per rendere meno verbosa la creazione di classi, introducente il marcatore "data" permettendo al programmatore di scrivere solamente il costruttore senza

dover pensare alla creazione dei metodi get,set, equals,toString,hashCode tipicamente utilizzati nelle stesura di classi Java. Questo rende il codice delle classi in Kotlin molto piú coinciso e leggibile.

Listing 2.11: Esempio class Kotlin

```
data class User(val name: String, var password: String)
```

Listing 2.12: Esempio class Java

```
public class User {
 private String name;
 private String password;

 public User(String name, String password) {
  this.name = name;
  this.password = password;
 }
 public String getName() {
  return this.name;
  }
  public String getPassword() {
   return this.password;
   }

public void setName(String name) {
 this.name = name;
 }

 public void setPassword(String password) {
  this.password = password;
  }
```

2.2.5 Altro

balblalbalba

2.3 Section3

Capitolo 3

Firebase

3.1 Storia

Capitolo 4

Primo Capitolo

Questo è il primo capitolo.

4.1 Prima Sezione

Questa è la prima sezione.
Ora vediamo un elenco numerato:

1. primo oggetto
2. secondo oggetto
3. terzo oggetto
4. quarto oggetto

Figura 4.1: legenda sotto la figura

4.2 Seconda Sezione

Ora vediamo un elenco puntato:

- primo oggetto
- secondo oggetto

4.3 Altra Sezione

Vediamo un elenco descrittivo:

OGGETTO1 prima descrizione;

OGGETTO2 seconda descrizione;

OGGETTO3 terza descrizione.

4.3.1 Altra SottoSezione

SottoSottoSezione

Questa sottosottosezione non viene numerata, ma è solo scritta in grassetto.

4.4 Altra Sezione

Vediamo la creazione di una tabella; la tabella 4.1 (richiamo il nome della tabella utilizzando la label che ho messo sotto): la facciamo di tre righe e tre colonne, la prima colonna "incolonnata" a destra (r) e le altre centrate (c):

$(1,1)$	$(1,2)$	$(1,3)$
$(2,1)$	$(2,2)$	$(2,3)$
$(3,1)$	$(3,2)$	$(3,3)$

Tabella 4.1: legenda tabella

4.5 Altra Sezione

4.5.1 Listati dei programmi

Primo Listato

```
        In questo ambiente     posso scrivere      come voglio,
lasciare gli spazi che voglio e non % commentare quando voglio
e ci sarÃ  scritto tutto.
Quando lo uso Ã¨ meglio che disattivi il Wrap del WinEdt
```

Appendice A

Prima Appendice

In questa Appendice non si è utilizzato il comando: `\clearpage{\pagestyle{empty}\cleardoublepage}`, ed infatti l'ultima pagina 8 ha l'intestazione con il numero di pagina in alto.

Conclusioni

Kotlin[1]. Conclusione bla bla.
blabla blablablabla blabla

[1]https://kotlinlang.org/

Ringraziamenti

Ringraziamenti bla bla