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 attualmente uno dei linguaggii piú usato e conosciuto al mondo 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, Caratteristiche di diversi linguaggi di programmazione preesistenti come CSharp, Scala, Groovy, Java e altri. Nel 2015 Google In 2015 Google prese in considerazione l'utilizzo di Kotlin come plugin per Android-Studio, e dopo vari test durante la conferenza Google IO (2017), venne annunciato Kotlin come linguaggio di programmazione ufficiale per lo sviluppo di applicazioni Android, senza escludere e rinunciare a Java.

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

2.2 Caratteristiche

Kotlin prendendo spunto da diversi linguaggi di programmazione imperativi e funzionali ha aggiunto diverse caratteristiche utili sia a livello sintattico che a livello prestazionale.
Le caratteristiche piú importanti offerte da Kotlin sono l'interoperabilitá con Java, permettendo l'utilizzo di librerie Java e Kotlin simultaneamente, introduzione di alcune caratteristiche di linguaggi di ordine superiore, la tipizzazione statica delle variabili,il null-safety permettendo di differenziare il tipo nullabile e il tipo non-nullabile, prevenendo errori di null pointer expetions, Il codice prodotto in Kotlin è inoltre piú compatto, coninciso, permette di scrivere data-class senza dover indicare getters e setters, supporto delle lambda function Coroutines estensioni delle funzioni e molto altro

2.2.1 Interoperabilitá

I linguaggi Kotlin e Java sono intercompatibili permettendo quindi a entrambi i linguaggi 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 dando 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)

Il compilatore di Kotlin permette inoltre di generare il java bytecode

Listing 2.4: Compilatore kotlin per generare Hello.class

```
$ kotlinc HelloWorld.kt
$ javap -c Hello.class
```

Oltre ad alcune aggiunte di Kotlin il bytecode generato dal compilatore Kotlin é molto simile a quello generato da Java, successivamente verranno analizzate funzioni e parti di codice piú complesse per evidenziare alcune modifiche apportate da kotlin nella compilazione del jar per ottimizzare o gestire nuove funzionalitá.
Un altro esempio utile per capire meglio l'interoperabilitá di Kotlin é il seguente

Listing 2.5: Esempio

```
import java.util.Calendar

fun calendarDemo() {
    val calendar = Calendar.getInstance()
```

```
    if (calendar.firstDayOfWeek == Calendar.SUNDAY) { // call
        getFirstDayOfWeek()
        calendar.firstDayOfWeek = Calendar.MONDAY  // call
            setFirstDayOfWeek()
    }
}
```

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 definire tutte le varibili come se fossero un oggetto.
La differenza fra i tipi primitivi e gli offetti sta nel fatto che i primi indicano solamente il tipo di una variabile, mentre gli offetti incapsulano il tipo e ne aggiungono funzionalitá e metodi aggiuntivi, inoltre il tipo primitivo non puó assumere valore nullo.
Kotlin rimuove le due distinzioni, ma solo ad alto livello, poiché di default quando viene inizializzata una nuova variabile utilizza gli oggetti, e in fase di compilazione 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 pressoccé le stesse che sono presenti in Java, con la particolariá che Kotlin cerca di evitare alcuni problemi dovuti a referenze a puntatori nullabile (NullPointerException).
Kotlin richiede che una variabile a cui assegnamo un valore nullo sia dichiarata con il simbolo "?", in caso contrario mostrera un errore in fase di compilazione

Listing 2.6: Esempio

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

Il simbolo "?" serve ad indicare che la variabile puó assumere un valore nullo, e lascia al programmatore la possibilitá di accederci ugualmente per saperne il valore, con l'utilizzo il simbolo "!!"",

Listing 2.7: Esempio !!

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

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

Listing 2.8: 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 modi per inizializzre una variabile.

- Lazy consente di delegare ad una funzione l'inizializzazione della variabile, e 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 Ã¨ 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 associato alla funzione, quello di default Ã¨ public; successivamente verranno indicati i parametri opzionali e il valore di ritorno

Listing 2.9: 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 evita di inserire controlli all'interno di funzioni, o addirittura creare un'altra funzione con parametri diversi, aumenta la leggibilitá del codice.

Listing 2.10: Esempio Kotlin

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

Listing 2.11: Esempio Java

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

Kotlin prendendo spuntoo dalla programmazione funzionale introduce le seguenti caratteristiche:

- Funzioni alto ordine: sono funzioni che possono accettare come parametri altre funzioni, e restituire a loro volta funzioni
- Assegnamento: Ã¨ possibile assegnare ad una

Listing 2.12: Esempio Kotlin

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

2.2.4 Data Classes

Kotlin come Java é un linguaggio orientato agli oggetto.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 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. Questa semplice introduzione, il codice delle classi in Kotlin risulta essere molto piú coinciso e leggibile.

Listing 2.13: Esempio class Kotlin

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

Listing 2.14: 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 Data Classes

Capitolo 3

Primo Capitolo

Questo è il primo capitolo.

3.1 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 3.1: legenda sotto la figura

3.2 Seconda Sezione

Ora vediamo un elenco puntato:

- primo oggetto
- secondo oggetto

3.3 Altra Sezione

Vediamo un elenco descrittivo:

OGGETTO1 prima descrizione;

OGGETTO2 seconda descrizione;

OGGETTO3 terza descrizione.

3.3.1 Altra SottoSezione

SottoSottoSezione

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

3.4 Altra Sezione

Vediamo la creazione di una tabella; la tabella **??** (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 3.1: legenda tabella

3.5 Altra Sezione

3.5.1 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