

Dipartimento di Ingegneria

dell’Informazione

Corso di Laurea in

Ingegneria Informatica

Sistemi Embedded A.A. 2017/2018

**TITOLO**

|  |  |
| --- | --- |
| **A cura di:**  Stefano Ivancich  Cristian Lazzarin  Federico Mantovan | **Docente:**  Carlo Fantozzi |

**INDICE**

Sommario

[**1.** **???Titolo Capitolo 1???** 1](#_Toc510638976)

[**Bibliografia** 2](#_Toc510638977)

1. **???Titolo Capitolo 1???**

**KOTLIN**

Kotlin è un linguaggio di programmazione ad oggetti open-source sviluppato dall’azienda Jet-Brains, la stessa che ha prodotto IDE rinomati come Android Studio. Questo linguaggio nasce dalla necessità di offrire ai programmatori uno strumento avanzato, che superi vincoli e convenzioni che i linguaggi più diffusi si portano normalmente dietro. Non c’è da stupirsi per esempio se in Java, utilizzato dal 1995, dobbiamo seguire delle regole che al giorno d’oggi ci sembrano banali, se non superate, ma che magari al tempo avevano una certa rilevanza e/o un certo fondamento. Kotlin si predispone come un passo avanti nella programmazione consentendo però un graduale approccio a tutti gli sviluppatori Java: esso infatti è pensato per compilare sulla Java Virtual Machine, ed è possibile programmare in Kotlin sfruttando classi Java (e viceversa). Essendo Android Studio sviluppato in Java e pensato inizialmente per uno sviluppo Java, Kotlin si adatta benissimo alla programmazione per applicazioni Android, ed è per questo motivo che dal 2017 Google ha dichiarato Kotlin come linguaggio ufficiale per le sue app al pari di Java.

NULL POINTER EXCEPTION

Uno dei più temuti errori a cui vanno incontro i programmatori Java è fortemente contrastato in Kotlin. Già in fase di compilazione infatti viene segnalato un errore in caso di assegnazioni di valori nulli o tentate operazioni su elementi nulli, a meno che non venga esplicitata la volontà di utilizzare un oggetto nullo mediante la sintassi *tipoDato?* (es String?). Quest’ultima sintassi viene fortemente utilizzata anche in fase di controllo, per esempio il comando

box?.setText(“testo”);

Modifica il testo dell’oggetto box solamente se non nullo, ignorando l’istruzione altrimenti.

val size=box?.length ?: -1

specifica invece cosa fare nel caso di oggetto nullo, assegnando -1 alla variabile size.

GESTIONE DELLE ECCEZIONI

In java esistono eccezioni Checked ( tutte quelle che estondono la classe *Exception* tranne RunTimeException) ed eccezioni Unchecked( tutte quelle che estendono *RuntTimeException*). Le prime differiscono dalle seconde per:

Dichiarazione dell’eccezione nella firma del metodo, mediante parola chiave *Throws*

Obbligo per il programmatore di gestire le eccezioni ogni qualvolta si usa un metodo che possa lanciarne, mediante il costrutto *Try/Catch*

Il compilatore segna errore nel caso il punto precedente non venga rispettato

L’utilizzo di eccezioni Checked assicura una maggior protezione dagli errori, tuttavia per progetti di notevoli dimensioni influiscono pesantemente sulle prestazioni; inoltre costituiscono un pesante ed inutile vincolo per le situazioni in cui si ha la certezza matematica che l’errore non possa verificarsi.

A fronte di ciò in Kotlin le eccezioni sono sempre Unchecked, permettendo una maggiore versatilità a chi scrive codice. I blocchi *Try/Catch* sono comunque ammessi ed utilizzati, ma non rappresentano un obbligo e, di conseguenza, il compilatore non genera errore se non utilizzati. Nella firma del metodo non va specificata l’eccezione, che verrà invece lanciata nei punti opportuni con l’espressione *throw*.

TIPI DI DATO

In Kotlin vige la seguente regola generale: qualsiasi tipo di dato è considerato oggetto (in fase di esecuzione in realtà alcuni dati che in Java sono primitivi (int, char..) restano così anche in Kotlin, ma al programmatore appaiono tutti sotto forma di classi, con relative funzioni. Questo permette una maggior libertà d’azione, ma inevitabilmente certe ovvietà del mondo Java non sono più valide: per esempio la conversione da un tipo di dato numerico ad un altro più grande non è automatica, ma avviene solo mediante specifiche funzioni delle relative classi.

VARIABILI/COSTANTI

Per dichiarare una variabile utilizzando Kotlin, basterà semplicemente utilizzare la keyword “*var”* all’inizio della dichiarazione stessa.



Inoltre, è possibile omettere il “TipoVariabile” se l’istanziazione e la dichiarazione avvengono sulla stessa riga.



Nel caso in cui vengano dichiarate delle variabili ed il loro valore non venga mai modificato, la compilazione segnalerà un “warning” che ci avviserà di ciò.

Per dichiarare una costante sarà sufficiente inserire la keyword *“val”* al posto della keyword *“var”.*

**PROPRIETA’ DELLE CLASSI**

Come in Java, in Kotlin si possono assegnare variabili e costanti ad una classe, che ne definiscono la struttura e per tal ragione vengono chiamate “proprietà”. Nella programmazione è molto frequente l’accesso in lettura e/o scrittura a queste variabili, mediante metodi getter e setter; per agevolare ciò, alcuni ambienti di sviluppo come Android Studio offrono strumenti che generano in modalità automatica questi metodi. In Kotlin si è fatto un ulteriore passo avanti, rendendo automatico il tutto e, allo stesso tempo, evitando di appesantire il codice. Il programmtore può accedere alla sintassi dell’oggetto semplicemente utilizzando la seguente sintassi

nomeOggetto.proprietàX=”valore”; // equivalente di una chiamata Set

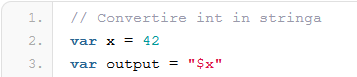
var a=nomeOggetto.proprietàX // equivalente di una chiamata Get

Questa funzionalità ancora una volta rende impossibile (se non forzata) la creazione di oggetti null, proprio per evitare che questi metodi “di background” possano generare errori.

CONVERSIONE DA UN TIPO DI DATO NUMERICO AD UNA STRINGA

Per poter convertire un tipo di dato numerico in una stringa, in kotlin è stato definito il simbolo “*$*” per compiere ciò. Infatti, basterà inserire all’interno dell’inizializzazione della stringa il carattere *“$”* affiancato dal nome della variabile numerica per eseguire la conversione.

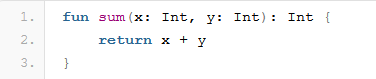
Esempio:



DICHIARAZIONE DI UNA FUNZIONE

La dichiarazione di una funzione avviene utilizzando la parola riservata *“fun”*, seguita da nome della funzione, eventuali parametri racchiusi tra parentesi tonde (indicando per ognuno il “NomeTipo” e il “NomeParametro”), ed infine il tipo di dato resistito preceduto dal simbolo “:”.

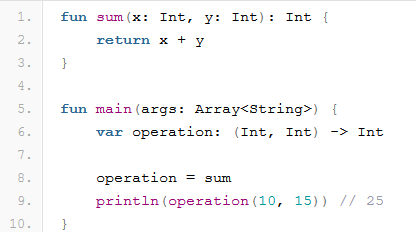
Esempio



Le funzioni definite in Kotlin, aggiungono oltre al classico funzionamento di java, la possibilità di poter essere assegnate a variabili, essere passate come argomento ad altre funzioni ed allocarne istanze a runtime.

Esempio

Una variabile di nome “operations” dichiara come tipo una funzione (definita precedentemente con il nome di “sum”), andando ad indicare fra parentesi i parametri accettati (di tipo “int”) e specificando il tipo di dato restituito (indicato con una freccia).



**EREDITARIETA’ E OVERRIDING**

In kotlin tutte le classi derivano implicitamente da una superclasse chiamata *Any*, similarmente a quanto avviene in Java con la classe *Object*. Tuttavia in *Any* non ci sono altri metodi al di fuori di *equals(), hashCode(), toString().*

Le classi possono derivare anche da altre superclassi, ma ciò è possibile solo se, nella dichiarazione della superclasse, è presente la parola chiave **open.** In caso contrario, ciascuna classe è di default **final,** non ammette cioè classi derivate. Un metodo di una classe open ammette overriding solo se, nella dichiarazione, viene a sua volta firmato come **open**; inoltre, nella classe derivata, si esplicita l’overriding con la parola **override** . Ciascun metodo segnato come override ammette a sua volta l’overriding (se la sua classe è marcata **open**); se si vuole evitare ciò, si utilizza la parola chiave **final.**

open class Padre {

open fun p() {} // ammette overriding

fun nv() {} // non ammette overriding

}

class Figlio() : Padre() {

final override fun p() {} // non ammette ulteriore overriding

}

La parola chiave **super** si usa, come in Java, per richiamare una funzione della superclasse.

**Bibliografia**

https://developer.android.com/kotlin/

https://kotlinlang.org/docs/reference/comparison-to-java.html

https://android-developers.googleblog.com/2018/02/introducing-android-ktx-even-sweeter.html

<https://codelabs.developers.google.com/codelabs/taking-advantage-of-kotlin/>

https://italiancoders.it/kotlin-cose-cosa-serve-sapere-ai-programmatori-java/

Istruzioni prese dal Sito del prof

“Report: si illustrino le ragioni che hanno spinto i progettisti di Kotlin a sviluppare un nuovo linguaggio; si illustrino le caratteristiche peculiari del linguaggio Kotlin, nonché somiglianze e differenze con il linguaggio Java.”

“Il report deve includere *una bibliografia che elenchi le fonti della ricerca effettuata.  
Il report non deve basarsi su un'unica fonte (ad esempio, la documentazione ufficiale di Google, sebbene tale documentazione sia un buon punto di partenza).  
Non è consentito effettuare operazioni di "taglia e incolla" dalle fonti: il report deve essere una rielaborazione originale delle informazioni trovate, non un riassunto o una copia.  
Poiché gli argomenti sono molto vasti, ciascun gruppo deve eseguire una selezione del materiale da presentare: in altre parole, deve scegliere che taglio dare al report, quali informazioni includere e quali escludere.  
I seguenti aspetti saranno presi in considerazione nella valutazione del report: chiarezza, completezza, capacità di sintesi (il report deve essere completo senza essere prolisso o costellato di dettagli banali), qualità della bibliografia, corrispondenza tra le informazioni selezionate per il report e le informazioni esemplificate nelle app.”*