# Linguaggi di programmazione

**Modulo 1**

Giacomo Fantoni
Telegram: @GiacomoFantoni
Github: https://github.com/giacThePhantom/AlgoritmiStruttureDati

6 agosto 2022

# Indice

# Capitolo 1

# Introduzione

## 1.1 Richiami di C++

### 1.1.1 Modello di memoria

La memoria è suddivisa in segmenti, il text, dove viene salvato il codice eseguibile, non scrivibile, lo stack e la heap.

#### 1.1.1.1 Lo stack

Nello stack sono salvate le variabili allocate staticamente, le funzioni, tutte le variabili allocate in maniera indipendente dal programma. Lo stack cresce e decresce in base alle necessità del programma, il contesto di ogni funzione viene salvata in questa area di memoria. Lo spazio necessario viene automaticamente allocato e liberato dal programma stesso. Se lo stack cresce troppo e si scontra con la heap si genera un fatal error che causa il termine del programma.

### 1.1.2 Gli array

Gli array sono collezioni di oggetti di tipo omogeneo salvate sequenzialmente nello stack. La dimensione dell'array deve essere dichiarata all'inizio del programma e non può essere cambiata. Durante l'esecuzione non c'e alcun controllo sugli indici.

### 1.1.3 Scope delle variabili

Le variabili globali sono delle variabili visibili in ogni contesto, vanno evitate a causa dei side-effects, possono essere sovrascritte da variabili locali con lo stesso nome. In java non esistono variabili globali. Il principio dell'information hiding permette di rendere il codice più leggibile rendendo tutte le variabili locali alle funzioni. Supportato dal principio di Parna secondo cui chi crea, implementa e utilizza deve conoscere solo le istruzioni a lui strettamente necessarie.

### 1.1.4 I puntatori

I puntatori sono delle strutture che contengono l'indirizzo di memoria di variabili, permettono di allocare vettori a run time di lunghezza decisa in quel momento, o per passare parametri che varranno cambiati anche all'esterno della funzione. L'operatore delete è necessario per liberare la memoria

in questo modo, ma per questo si deve prestare attenzione a mantenere salvati i puntatori alle zone salvate, altrimenti si generano dei memori leak.

## 1.2 Cenni storici al java

Il java viene introdotto per superare dei problemi del C++: la gestione della memoria: esiste un costrutto che permette di liberare automaticamente la memoria nell'heap quando non viene utilizzata, elimina l'aritmetica dei puntatori, controlla le dimensioni degli array e introduce una migliore gestione delle stringhe. Aiutando in questo modo a scrivere codice più robusto. Inoltre sposa quasi completamente il paradigma object oriented. In java l'eseguibile generato dal compilatore viene interpretato successivamente da una virtual machine, rendendo molto più semplice la portabilità.

# Capitolo 2

# Classi e oggetti, attributi e metodi

Gli attributi determinano lo stato degli oggetti instanziati a partire da una determinata classe, il cui valore è particolare ad ogni istanza. I metodi definiscono invece il comportamento di tale oggetto, con codice condiviso per tutte le istanze di tale classe. Le istanze di una classe sono oggetti, che contengono un riferimento ad un oggetto. Si noti che in java gli oggetti sono accessibili solo per riferimento.

#### 2.0.0.1 Passaggio di parametri

I parametri formati da tipi primitivi sono passati per copia, mentre i parametri costituiti da oggetti, sono passati per copia del riferimento, pertanto gli oggetti sono sempre passati per riferimento.

### 2.0.1 Package e information hiding

Attributi e metodi di una classe per cui non è stabilita alcuna regola di visibilità sono visibili solo all'interno dello stesso package in cui è contenuta la classe. Ogni compilation unit (file .java) contiene classi appartenenti allo stesso package e contiene un'unica classe public e altre private. Il package java.lang contiene classi di uso molto comune pertanto vengono importate in automatico.

#### 2.0.1.1 Attributi costanti

È possibile creare degli attributi costanti prefissando a essi la keyword final. Le asserzioni (la terminazione di compilazione) sono determinate da un System.exit(1), spesso precedute da un output in console.

### 2.0.2 Distruttore

In java non è permesso distruggere esplicitamente gli oggetti, questa operazione è compiuta in automatico dal garbage collector, che interviene automaticamente quando si rende necessaria della memoria, liberando quella occupata da oggetti la cui referenza non è più attiva. Può essere esplicitamente attivato attraverso System.gc() che di norma non è necessario. È possibile definire nel metodo finalize azioni da eseguire all'atto della distruzione dell'oggetto. Non è un distruttore è utilizzato per rilasciare risorse diverse alla distruzione di un oggetto.

### 2.0.3 Creazione di oggetti

I nuovi oggetti sono creati con l'operatore new, che richiede la chiamata di un particolare metodo: il costruttore che alloca la memoria necessaria all'oggetto e inizializza lo spazio allocato. Il costruttore ha lo stesso nome della classe, non indica il tipo di ritorno e non ha return, è possibile fare dell'overloading. Di default viene creato un costruttore che si limita ad allocare lo spazio necessario alla memoria.

#### 2.0.3.1 Tipi array

Sono anch'essi tipi riferimento come le classi. Dato un tipo T, un array viene diciarato con T[]. In mancanza di inizializzazione la dichiarazione di un'array non alloca spazio per i suoi elementi. L'allocazione si crea dinamicamente attraverso new T[dimensione]. Se non sono di tipo primitivo, alloca spazio per i riferimenti. System.arraycopy(src, int srcPos, dst, int dstPos, int length) copia length elementi da src a partire da srcPos in dst a partire da dstPos.

# Capitolo 3

# Ereditarietà e polimorfismo

Tutte le classi di un sistema object oriented sono legate da una gerarchia di ereditarietà definita mediante la parola extends. La sottoclasse eredita tutti i metodi e attributi della superclasse. Java supporta solo ereditarietà semplice, ovvero una classe può ereditare da un'unica superclasse.

#### 3.0.0.1 Classe Object

Se la parola extends non è specificata la classe eredita di default dalla classe Obejct che fornisce dei metodi: public void equals(Object), protected void finalize(), public String toString(). Le estensioni di una classe possono essere strutturali (aggiunta di nuovi attributi) o comportamentali (aggiunta di nuovi metodi o modifica di metodi esistenti).

#### 3.0.0.2 Overriding

Una sottoclasse può ridefinire i metodi della sua superclasse attraverso l'overriding. All'interno di un metodo che ne ridefinisce uno della superclasse si può fare riferimento ad esso attraverso il costrutto super.

#### 3.0.0.3 Costruttori

I costruttori non vengono ereditati, all'interno di un costruttore si può fare riferimento al costruttore della superclasse attraverso super, se non inserito il compilatore inserisce il codice che fa riferimento al compilatore di default senza parametri.

#### 3.0.0.4 Overloading

All'interno di una classe possono esserci metodi con lo stesso nome purchè si distinguano per numero o tipo di parametri. Il tipo di ritorno non basta a eliminare l'ambiguità.

## 3.1 Costruttori

Se per una classe non viene specificato alcun costruttore viene creato in automatico il costruttore vuoto, se ne specifico uno questo non avviene. Una sottoclasse, a meno che sia diversamente specificato con super chiama come prima cosa il costruttore della superclasse senza parametri. All'interno

di un costruttore si può chiamare un costruttore della stessa classe con this(), a patto che questa sia la prima istruzione del costruttore.

## 3.2 Classi astratte

Un metodo astratto è un metodo per il quale non è implementata alcuna istruzione. Una classe astratta è tale se contiene almeno un metodo astratto. Entrambi sono definiti tali attraverso la keyword abstract. Non è possibile creare istanze di una classe astratta, si necessita di definire una loro sottoclasse che ne implementa i metodi astratti. Sono utili per introdurre astrazioni di alto livello.

## 3.3 Information hiding

La visibilità di metodi e attributi può essere definita come:

- public: visibili a tutti, vengono ereditati,
- protected: visibili solo alle sottoclassi, vengono ereditati.
- private: visibili solo nella classe, non visibili nelle sotto classi.
- Di default il valore di accesso è comune al package.

# Capitolo 4

# Gestione degli errori

Si ottiene grazie al blocco try...catch che una volta trovato un errore in try genera un eccezione che viene catturata in catch che viene eseguito. finally, a differenza del catch viene sempre eseguito, sia che sia stata generata un eccezione che non sia stata generata.

# Capitolo 5

# Polimorfismo

Con polimorfismo si intende la capacità di un elemento sintattico a riferirsi a elementi di diverso tipo.

### 5.0.1 Principio di sostituzione di Liskov

Se S è un sottotipo di T, allora le variabili di tipo T in un programma possono essere sostituite con variabili di tipo S senza alterare alcuna proprietà desiderabile del programma. Pertanto una variabile di tipo T può riferirsi a oggetti di tipo T o di un suo sottotipo, allo stesso modo per i parametri formali di una funzione. In java il legame tra un oggetto e il suo tipo è detto dynamic binding (o lazy/late binding). In presenza di polimorfismo si distingue tra tipo statico, dichiarato a compile time e tipo dinamico, assunto a runtime. In java la chiamata di una funzione dipende dal tipo dinamico di un oggetto e non dal suo tipo statico. Il compilatore determina la firma del metodo da eseguire basandosi unicamente sul metodo statico. In caso di overriding la specifica implementazione del metodo la cui firma è stata decisa dal compilatore viene scelta basandosi sul metodo dinamico.

### 5.0.2 Regole per il binding

Chiamato un metodo che dipenda dal tipo dell'oggetto si cerca all'interno della classe del tipo statico dell'oggetto il metodo con la firma più vicina all'invocazione. Successivamente si guarda al tipo dinamico dell'oggetto e si nota se è stato fatto l'override di tale metodo scelto. Se sì si usa l'implementazione del tipo dinamico. In Java le decisioni sono sempre a runtime salvo quando sia possibile decidere univocamente a compile time per metodi private, static e final e costruttori.

# Capitolo 6

# Interfacce

Un'interfaccia è una classe composta interamente da metodi astratti, senza attributi. Un'interfaccia può contenere costanti, spesso si usano interfacce completamente vuote per taggare classi con speciali proprietà. Un'interfaccia può ereditare da una o più interfacce. Una classe può implementare una o più interfacce e implementarne tutti i metodi astratti a meno che sia astratta. L'interfaccia si limita a definire il comportamento che una classe deve avere. Si utilizza per gestire il problema delle multiple inheritance in modo da non rischiare di incorrere in metodi con la stessa firma. Un'interfaccia può essere utilizzata per definire il tipo di una variabile ma non per creare un oggetto.

## 6.1 Collections

Una collection è un oggetto che raggruppa elementi multipli in una singola entità. A differenza degli array gli elementi possono essere eterogenei e la sua lunghezza non è fissata. Sono utilizzate per immagazzinare, recuperare e traseferire gruppi di dati da un metodo all'altro. Il java collection framework contiene tre tipi di elementi: interfacce per specificare diversi tipi di servizi associati a diversi tipi di collection, potenzialmente associate a diverse strutture dati, implementazioni di specifiche strutture dati di uso comune che implementano le interfacce e algoritmi, codificati in metodi, che implementano operazioni comuni a più strutture dati. Come per esempio ricerca ordinamento, mescolamento, composizione. Lo stesso metodo può essere utilizzato in diverse implementazioni.

### 6.1.1 Interfacce

1. Collection, un'arbitraria collezione di elementi.
    - Set, collection senza duplicati.
        - SortedSet, set su cui è definito un ordinamento.
    - List, collection in sequenza ordinata, accessibili tramite indice, sono ammessi duplicati.
    - Queue, collection in coda in cui sono ammessi duplicati.
        - Deque queue a cui è possibile accedere da entrambe le estremità.
2. Map, una collezione di coppie chiave-valore senza duplicati.
    - SortedMap, map su cui è definito un ordinamento sulle chiavi.

### 6.1.2 Operazioni base

- int size();
- boolean isEmpty();
- boolean contains(Object element);
- boolean add(Object element); (true se la collection è cambiata)
- boolean remove(Object element); (true se la collection è cambiata)
- Iterator iterator();

### 6.1.3 Bulk operation

Consentono di effettuare operazioni su più elementi contemporaneamente.

- boolean containsAll(Collection c);
- boolean addAll(Collection c); (true se la collection è cambiata)
- boolean removeAll(Collection c); (true se la collection è cambiata)
- boolean retainAll(Collection c);
- void clear();
- Object[] toArray();

# Capitolo 7

# JavaFx