Struttura delle classi in Java

Derivando da un modello di Programmazione ad Oggetti, il linguaggio Java è dotato di classi le quali possono contenere **metodi** e **attributi**. Questi due elementi si riverberano sull'aspetto/funzionamento degli **oggetti**, chiamati anche **istanze della classe**. Più nel dettaglio i **metodi** definiscono il **comportamento** di un determinato oggetto mentre gli **attributi** un suo possibile stato. Il vero vantaggio di questo linguaggio è che il codice si dice essere universale, ovvero la descrizione di un metodo o la presenza di un attributo all'interno di una classe specifica, può essere **utilizzata più volte** sui diversi oggetti (istanze) di quella stessa classe.

Passaggio di oggetti

Ogni variabile il cui tipo sia una classe (o alternativamente un'interfaccia) contiene un **riferimento** ad un oggetto. In Java gli oggetti sono accessibili esclusivamente per **riferimento** (a differenza del C/C++ nel quale si poteva decidere se allocare sulla Heap o sullo Stack). Segue direttamente che ad ogni variabile di tipo riferimento può essere assegnato il riferimento *null*.

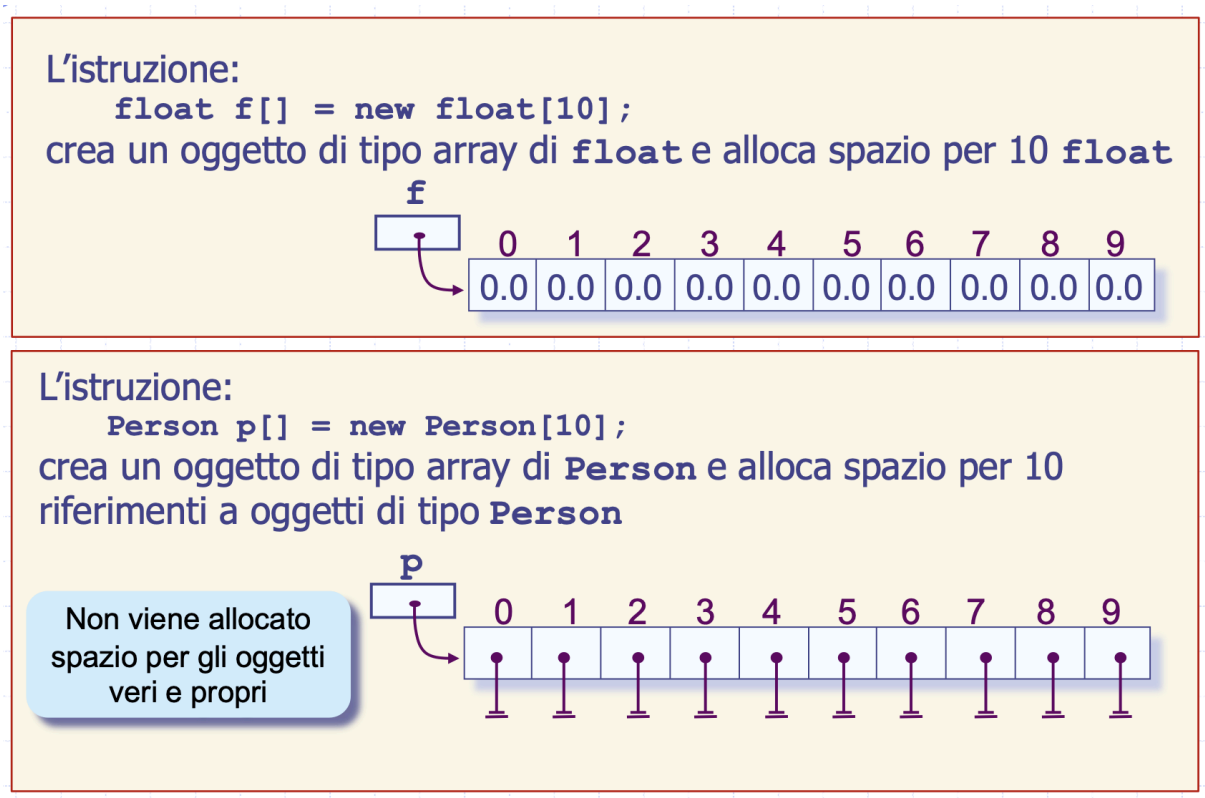
Costruttori e distruttori

Seppur i riferimenti del linguaggio Java sono un concetto molto simile ai puntatori in C/C++, nel primo caso **non è necessario implementare e chiamare un distruttore** a seguito delle allocazioni di memoria effettuate in quanto se ne occupa già il **garbage collector**, funzionalità preimpostata del linguaggio Java che elimina automaticamente le allocazioni obsolete che non vengono utilizzate dal programma.

Per quanto riguarda i costruttori, essi vanno posizionati all'interno della classe ed hanno lo stesso funzionamento dei costruttori nel linguaggio C/C++ posizionati all'interno di *struct*. E' sempre possibile fare un *overloading* di costruttori, dichiarandone più di uno, con firme diverse, all'interno della classe. Se non è presente alcun costruttore all'interno di una determinata classe viene invocato il **costruttore di default** che opera **inizializzando al valore di base tutti gli attributi di tipo primitivo** e **inizializzando a *null* tutti gli oggetti di tipo definito dall'utente**, tutto ciò sempre dopo aver prima allocato spazio in memoria.

Allocazione di tipi Array

La dichiarazione di un tipo Array **non alloca automaticamente** spazio per i suoi elementi, tale allocazione si realizza invece **dinamicamente** tramite l'operatore " new <tipo> [dimensione] ". Questa operazione si comporta in maniera differente in base al tipo scelto: se il tipo è **non è primitivo** allora l'operatore " new " alloca spazio **solo per i riferimenti**. Ogni elemento dell'array andrà inizializzato come se fosse un **oggetto a sé stante.**



Metodi e attributi di classe

Dichiarando un metodo o un attributo come " static … " fa si che quel metodo o attributo appena creato **appartenga alla classe** e quindi che in qualsiasi successiva creazione di istanza, **l'istanza stessa goda dei valori attuali e aggiornati dell'attributo (o del metodo) statico della classe**.

E' possibile accedere ad un attributo o metodo statico di una classe senza la necessità di creare un oggetto, ovvero tramite la notazione " <classe>.<attributoStatico> " (o allo stesso modo metodoStatico). Segue che una modifica ad un attributo statico effettuata da un' istanza di una particolare

classe si **riverbera ed è resa visibile da tutte le altre istanze di quella classe.** Si tengano presenti le seguenti regole:

- un'istanza può accedere sia a **metodi convenzionali** che a **metodi statici**.
- una classe può accedere solo a **metodi statici**.
- un metodo statico può accedere **solo** ad **attributi e metodi statici**
- un metodo convenzionale può accedere **liberamente** a ad **attributi o metodi statici** o **attributi o metodi convenzionali**.

Dichiarazione di valori costanti

E' possibile definire attributi costanti con la notazione " final <definizione di attributo> = <valore> ". Essendo un valore costante è errato da parte di un oggetto (istanza della classe) assegnare un valore ad una costante di tipo *final* in qualsiasi parte del codice.

Proprietà dei Package e Info Hiding

Si noti che di base ogni classe **appartiene ad un package**, se quest'ultimo non viene specificato allora di base tutte le classi create vanno a finire in un package implicito senza nome. Se ho necessità di avere una classe all'interno di un altro package posso sempre importarla con il comando " import <nomeClasse> ".

Non sempre però le operazioni di esportazione di una classe sono eseguibili, è discriminante il fatto che la classe o uno dei suoi metodi o attributi sia dichiarata come *public, private o protected*.

Info Hiding per classi

Solo le **classi *public* possono essere esportate** in un **differente package**, altrimenti se l'ambito di visibilità non dovesse essere specificato, tali classi sono visibili esclusivamente all'interno del package nel quale esse sono state definite.

Info Hiding per metodi e attributi

Si noti che per attributi o metodi di una **classe** per cui **non è dichiarato alcun tipo di visibilità** sono **visibili solo nelle classi appartenenti al package** della prima classe. Altrimenti se specificato che una classe è *public* e quest'ultima viene importata in un nuovo package, porterà con sé i metodi e

gli attributi definiti all'interno di essa rimanendo concorde alla visibilità di questi ultimi.