ASTRAZIONE DEI DATI

Introduzione

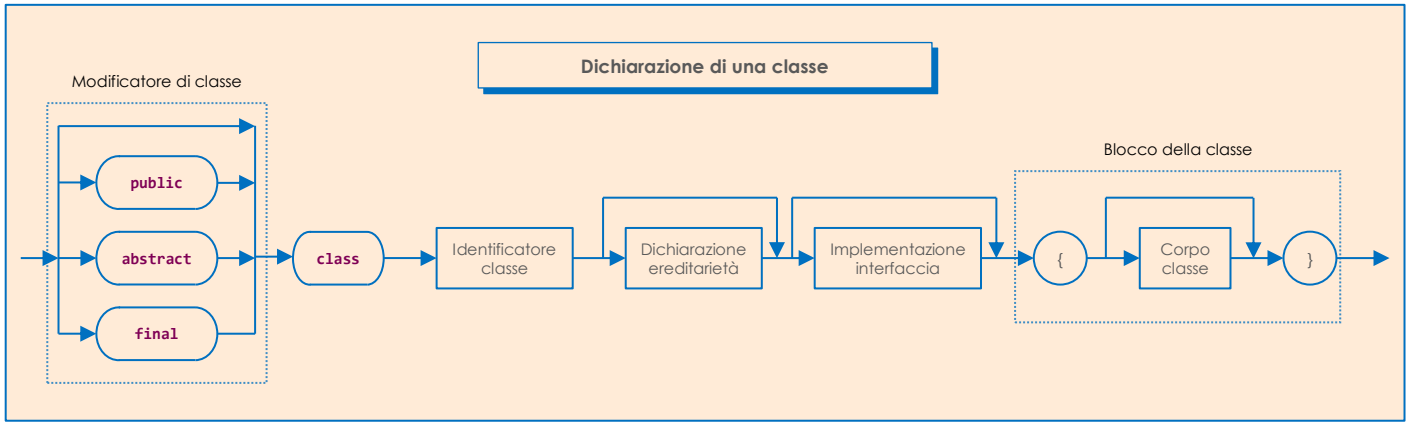
Il Java è un linguaggio di programmazione orientato agli oggetti (OOL), perché permette di realizzare in un programma tutti i concetti alla base dell'OOP quali:

- ✓ l'astrazione dei dati, mediante classi e oggetti;
- ✓ l'incapsulamento (di strutture di dati e operazioni);
- ✓ l'ereditarietà;
- ✓ il polimorfismo.

Durante la realizzazione di un progetto object oriented "puro", utilizzeremo tutti i princìpi studiati alla base del linguaggio (tipi di dato primitivi, istruzioni, array e sottoprogrammi), che sono essenziali per la comprensione di come realizzare i costrutti propri dell'OOP.
La fase di codifica segue quella di progettazione OO (OOD: *Object Oriented Design*) ed è inserita all'interno del ciclo di vita del software. La scrittura di un programma Java non deve quindi essere considerata un'attività autonoma, ma come la traduzione nel codice sorgente di un progetto orientato agli oggetti.

Le classi

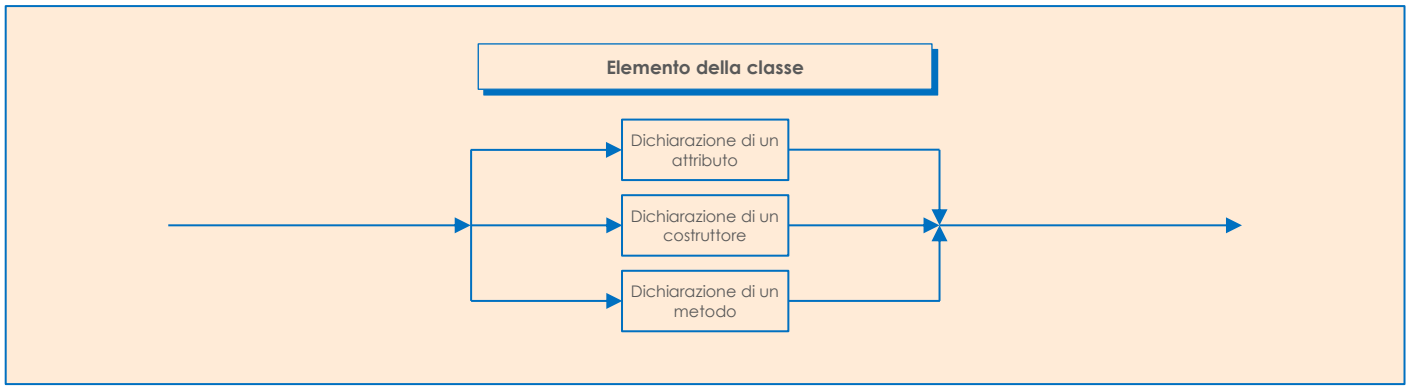
Finora abbiamo sempre utilizzato le classi già disponibili tra quelle del JFC; nel seguito dovremo modificare il nostro modo di pensare, ricreando nel problema da risolvere un insieme di classi da costruire daccapo in un programma sorgente Java. Quest'ultimo meccanismo, proprio dell'OOP, è noto come astrazione dei dati.
L'astrazione dei dati è il processo logico seguito da un progettista (nell'OOD) per cui, prescindendo dalle caratteristiche individuali e personali degli oggetti, si individuano nella realtà soltanto le proprietà (o attributi) e i comportamenti generali racchiudendoli nel concetto di classe. Il Java supporta l'astrazione dei dati permettendo al programmatore di realizzare in un programma sorgente i concetti di classe o di oggetto introdotti nella fase di progettazione OO.
Per realizzare una nuova classe in un programma sorgente, il Java dispone del costrutto del linguaggio **`class`**.
La dichiarazione di una nuova classe Java è basata sul diagramma sintattico seguente.



Nella costruzione delle prime classi, tralasceremo i concetti legati all'ereditarietà e all'implementazione di interfacce che saranno oggetto delle prossime UD.
Le keyword facoltative **`public`**, **`abstract`** e **`final`** sono i modificatori di classe (*class modifier*) e permettono di definire classi con caratteristiche particolari. **`final`** e **`abstract`** sono legati ai concetti dell'ereditarietà e del polimorfismo, mentre **`public`** dichiara la visibilità di una classe (ambito) anche al di fuori del package in cui è contenuta. Studieremo in seguito come costruire nuovi package ovvero gruppi di classi (librerie) in relazione tra loro. Se non si specifica alcun modificatore di classe, l'ambito è soltanto quello del package in cui la classe è dichiarata. Nel seguito costruiremo classi visibili in qualsiasi libreria, per cui le dichiareremo con il modificatore **`public`**.

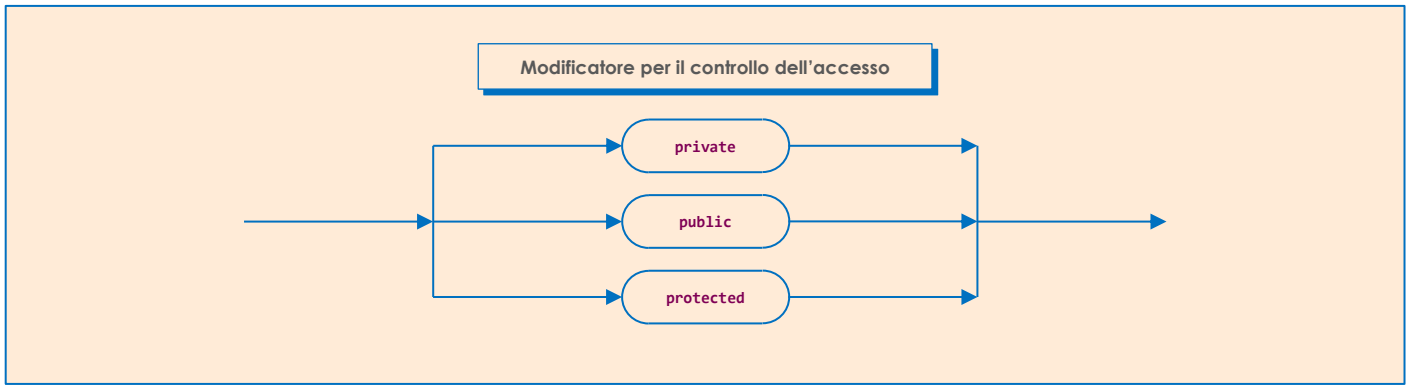
Elementi di una classe

Gli elementi o membri (*member*) di una classe possono essere, come descritto nel seguente diagramma sintattico, attributi, costruttori o metodi.



Realizzazione dell'incapsulamento

Ricordiamo che, nell'OOP, l'incapsulamento (*encapsulation*) è quel meccanismo che permette di includere, all'interno della dichiarazione di una classe, sia il modello dei dati sia gli algoritmi che definiscono l'elaborazione di tutti gli oggetti della classe stessa. In Java, questo meccanismo è realizzato mediante i modificatori per il controllo dell'accesso (o semplicemente modificatori di accesso) agli elementi di una classe, descritti nel grafo sintattico che segue.



I modificatori per il controllo dell'accesso evidenziano, nel corpo di una classe, i membri:

- ✓ incapsulati all'interno della classe e quindi non visibili al suo esterno;
- ✓ accessibili all'esterno.

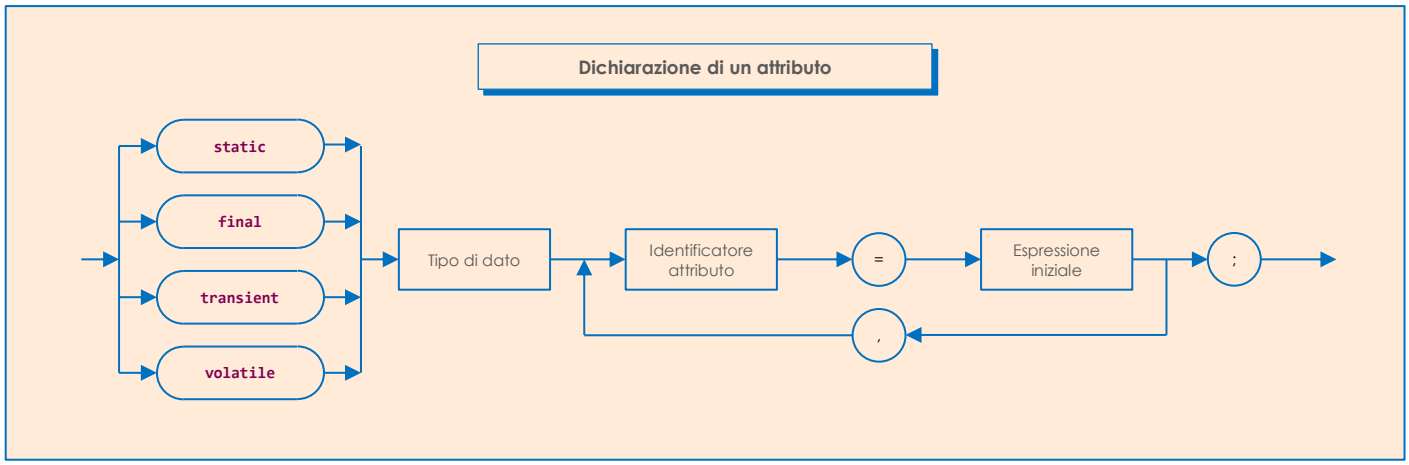
La tabella successiva schematizza il ruolo delle parole **`private`** e **`public`** impiegate, prima della dichiarazione di un elemento, nel corpo di una classe.

Modificatore per il controllo dell'accesso:	Definisce la parte della classe ...	L'elemento dichiarato dopo il modificatore per il controllo dell'accesso è visibile ...
`private`	privata	... esclusivamente all'interno della classe dagli altri membri.
`public`	pubblica	... sia all'interno sia all'esterno della classe.

Come studieremo in seguito, il Java definisce anche un accesso protetto (*protected*), impiegato nel meccanismo dell'ereditarietà, per definire la visibilità degli elementi all'interno di una gerarchia di classi correlate tra loro. In una classe, se non si dichiara un modificatore, l'accesso di default a un membro è quello pubblico all'interno dello steso package.
Seguendo la terminologia dell'OOP, l'incapsulamento del modello dei dati e delle operazioni in una classe include automaticamente le proprietà di occultamento dei dati (*information hiding*) e di indipendenza funzionale.

Attributi

Gli attributi (*attribute*) sono le variabili che memorizzano i dati degli oggetti di una classe e si dichiarano sulla base della sintassi illustrata di seguito.



Gli attributi (o variabili) di istanza contengono tutte le proprietà, tipiche di una classe, individuate durante la fase di progettazione OO. Gli attributi di istanza (o semplicemente attributi) possono essere di un tipo di dato primitivo oppure, a loro volta, del tipo dichiarato in altre classi. Un oggetto è un'istanza (una copia) di una classe, per cui la specificazione "di istanza" ci suggerisce che gli attributi sono le variabili proprie di un particolare oggetto (o istanza).
Frequentemente, per descrivere la realtà, gli attributi dichiarano modelli dei dati concreti complessi, costruiti mediante una o più strutture di dati, semplici o strutturate, dichiarate all'interno della classe.
Come abbiamo già studiato, l'ambito di una variabile di istanza è quello del corpo della classe. Un attributo, dopo essere stato dichiarato in una qualsiasi posizione del corpo di una classe, è quindi visibile all'interno di tutti gli elementi (costruttore e metodi) della classe stessa.
L'accesso agli attributi pubblici in un oggetto di una classe avviene mediante l'operatore "punto" usato con la seguente sintassi generale:

nomeOggetto.nomeAttributoPubblico

Per realizzare in modo corretto il meccanismo dell'incapsulamento dell'OOP, gli attributi di una classe devono essere dichiarati privati e non pubblici. Seguendo la terminologia dell'OOP, l'insieme dei valori assunti dagli attributi privati (e quindi incapsulati in un oggetto) ne definiscono il suo stato interno. Una variabile di istanza pubblica può essere letta o scritta dagli utenti (con l'operatore punto) senza alcun controllo sul suo accesso da parte dei programmatori.
I modificatori di attributo **`transient`** e **`volatile`** sono legati ad aspetti avanzati delle classi (serializzazione degli oggetti e programmazione multithread).
Il modificatore **`final`** trasforma una variabile in una costante, visibile nel bocco della classe. Il modificatore **`static`** permette di richiamare un attributo (o variabile) di classe, ovvero un attributo accessibile anche senza aver creato alcun oggetto di una classe. Le variabili di classe (dette anche attributi statici) sono comuni e condivise tra tutti gli oggetti della medesima classe creati in un programma. Quando si parla di "attributi" si intendono sempre quelli di istanza, mentre per le variabili dichiarate **`static`** si deve specificare: "attributi di classe o statici".

Metodi

Un metodo realizza un'elaborazione specifica del comportamento di una classe.
I metodi del Java realizzano gli omonimi metodi utilizzati nella fase di progettazione OO per definire il comportamento di tutti gli oggetti di una classe.
Abbiamo già studiato come costruire dei metodi intesi come funzioni o sottoprogrammi all'interno delle classi.
In generale, i metodi di una classe possono essere:

- ✓ pubblici (modificatore di accesso **`public`**), quando definiscono un'elaborazione utile sugli oggetti di una classe, richiamata dall'esterno con l'operatore "punto" mediante una sintassi generale del tipo:

  nomeOggetto.nomeMetodoPubblico([*lista parametri*]$_{optional}$)

- ✓ privati (modificatore di accesso **`private`**), se sono utilizzati (e quindi visibili) esclusivamente all'interno della classe, richiamandoli semplicemente con il loro identificatore con una forma del tipo:

  nomeMetodoPrivato([lista parametri]$_{optional}$)

Come per gli attributi, se non si specifica il modificatore di accesso **`public`** o **`private`**, il metodo è visibile da tutte le classi all'interno dello stesso package (accesso di default).

Seguendo la terminologia dell'OOP, la chiamata di un metodo pubblico definisce un "messaggio" (*message*) inviato a un oggetto.
La dichiarazione di un metodo può anche essere preceduta dai modificatori **abstract**, **final** e **synchronized**, utilizzati per realizzare funzionalità particolari legate, rispettivamente, alle interfacce, all'ereditarietà e alla programmazione multithreading.
Il modificatore **static**, applicato ai sottoprogrammi pubblici, permette di dichiarare metodi statici o di classe, condivisi tra tutti gli oggetti della medesima classe e richiamati con una sintassi (a noi nota) del tipo seguente:

NomeClasse.nomeMetodoStatico([*lista parametri*]$_{optional}$)

Per la loro caratteristica di "esistenza senza allocazione di un oggetto", i metodi di classe:

- ✓ possono richiamare soltanto altri metodi statici;
- ✓ possono usare variabili locali o **static**, ma non attributi di istanza della classe che fanno sempre riferimento a un oggetto specifico.

I programmatori utilizzano i metodi statici per realizzare dei sottoprogrammi di utilità generale non legati a specifici tipi di oggetti. A esempio, una classe *Vettore* potrebbe avere un insieme di metodi di classe per effettuare le operazioni di ordinamento, ricerca di un valore, media, calcolo del massimo/minimo ecc.
Particolari tipi di metodi pubblici sono quelli che vengono applicati per l'accesso agli attributi privati, invisibili all'esterno della classe, ma accessibili all'interno del corpo della classe mediante metodi. Questi metodi hanno nomi del tipo: *scriviAttributo*(), per impostare (assegnare) un valore a un attributo privato, e/o *leggiAttributo*() per dichiarare l'operazione di lettura del valore di una variabile di istanza. Lo scheletro del codice sorgente dei metodi di accesso e differente nei due casi:

- ✓ per la scrittura del valore di un attributo:

```
public void scriviAttributo(tipoDatoAttributoPrivato nomeParametro) {
        nomeAttributoPrivato = nomeParametro;
}
```

- ✓ per la lettura del valore di una variabile di istanza:

```
public tipoDatoAttributoPrivato leggiAttributo() {
        return nomeAttributoPrivato;
}
```

In base ai metodi di accesso implementati in una classe, gli attributi privati possono quindi essere:

- ✓ a sola scrittura, se possiedono solo un metodo del tipo scriviAttributo();
- ✓ a sola lettura, se è presente solo un metodo del tipo leggiAttributo();
- ✓ a lettura/scrittura, se sono disponibili entrambi i metodi precedenti.

Costruttori

In Java, il costruttore (*constructor*) è il primo metodo di una classe che viene richiamato dopo l'allocazione di un oggetto nella memoria.
Un costruttore si dichiara come qualsiasi altro metodo della classe con le differenze seguenti:

- ✓ il suo identificatore coincide con il nome della classe a cui appartiene;
- ✓ non restituisce mai alcun valore, per cui non deve essere specificato come tipo *void*.

Lo scheletro generale del codice sorgente di un costruttore è quindi quello riportato nel seguito.

```
[public]optional NomeClasse([lista parametri]optional) {
        // Dichiarazione del corpo del costruttore.
}
```

Per il suo funzionamento, il corpo di un costruttore è il punto ideale della classe per inizializzare tutti gli attributi incapsulati di un nuovo oggetto e quindi dichiarare, secondo l'OOP, il suo stato interno iniziale.
Nel codice sorgente, il costruttore è richiamato dal programmatore nell'istruzione di allocazione di un oggetto con l'operatore new. Quando un programmatore scrive:

```
NomeClasse NomeOggetto = new NomeClasse();
```

in realtà sta richiamando il costruttore della classe con la chiamata *NomeClasse*(). In questa posizione, se il costruttore è dichiarato con dei parametri di ingresso, è quindi possibile passare al metodo iniziale degli argomenti.
L'impiego del costruttore nel metodo *main*() di un'applicazione diventa quindi alternativo per gli utilizzatori di una classe (altri programmatori), all'impostazione dei singoli attributi incapsulati con i metodi di accesso per la lettura e scrittura dei valori delle variabili di istanza.
Ogni classe deve avere almeno un costruttore. Se il programmatore non ne dichiara nessuno, il compilatore inserisce automaticamente un costruttore di default che non ha parametri di ingresso e ha un corpo senza istruzioni, che quindi non svolge nessuna elaborazione. Il costruttore scritto dal programmatore è quindi dichiarato in modo esplicito (*explicit constructor*), mentre quello di default è inserito in modo automatico dal compilatore in modo implicito (*implicit constructor*).
In una classe un costruttore con parametri dichiarato dal programmatore sovrascrive quello di default, per cui la creazione di un oggetto con la forma: **new** *NomeClasse*(), fornisce un errore di compilazione.

I costruttori possono essere soggetti a overloading per cui è possibile dichiarare più metodi, con lo stesso nome della classe ma con un numero di parametri e/o tipo di dato differenti.

Linee guida per la costruzione di una nuova classe

Schematizzando tutti i concetti studiati in precedenza, durante il progetto e la successiva realizzazione di una nuova classe Java, possiamo seguire le seguenti linee guida.

1. Per implementare in modo corretto l'incapsulamento, tutti gli attributi di una classe devono essere dichiarati privati. L'accesso agli attributi di istanza deve essere controllato dal programmatore mediante:
   - ✓ dei metodi per la lettura/scrittura dei valori incapsulati;
   - ✓ dei costruttori con dei parametri di ingresso.
2. Mantenere privati tutti i metodi utilizzati unicamente all'interno della classe.
3. Dichiarare metodi pubblici per richiamare le elaborazioni codificate all'interno della classe (algoritmi propri del progetto incapsulati).

Gli oggetti

Una classe Java definisce un nuovo tipo di dato, creato dal programmatore per modellare la realtà che vuole descrivere.

Un oggetto è una copia (o istanza) degli attributi di istanza dichiarati in una classe.

Il termine istanza (*instance*) indica che un oggetto è una copia delle strutture di dati incapsulate nella classe. In altri termini, una classe dichiara un modello (*template*) per i dati (attributi), mentre un particolare oggetto contiene effettivamente i dati (assegnati alle variabili di istanza). Nell'elaborazione, ciò che contraddistingue un oggetto da un altro consiste, effettivamente, nei dati differenti assunti dalle variabili di ogni specifica istanza.

In Java, la creazione di un oggetto avviene, nell'ordine, nelle seguenti due fasi distinte:

1. dichiarazione di una variabile riferimento;
2. creazione di un oggetto in memoria.