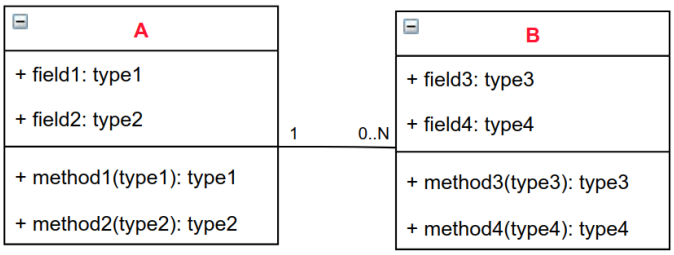
Pillole di Programmazione a Oggetti

Come risolvere il pattern di “associazione 1 a N” fra classi



ANALISI DEL CONTESTO:

Si supponga di avere implementato le due classi **Studente** e **Voto** in figura 1:

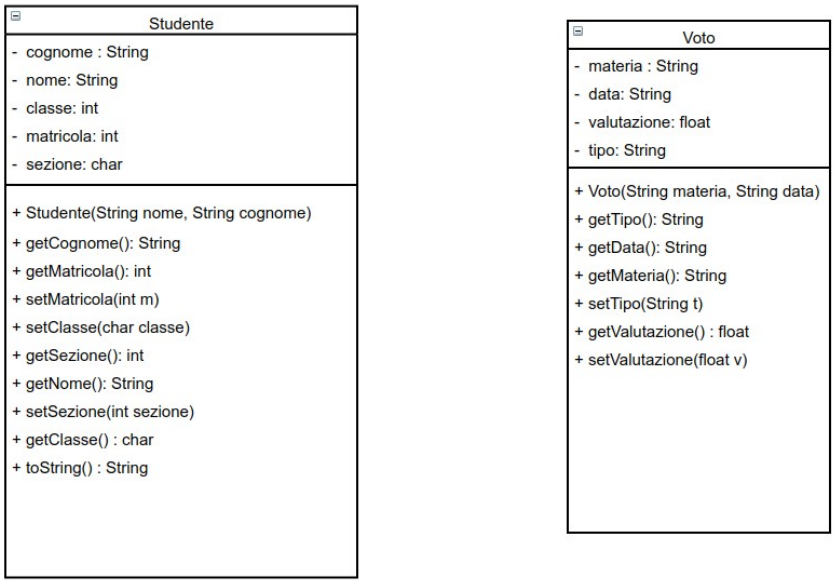


*figura 1: classi **Studente** e **Voto***

Vogliamo adesso legare il concetto di Voto con quello di Studente realizzando quello che viene chiamato “associazione di classi”.
E’ infatti abbastanza evidente che un singolo studente (o studentessa) può ricevere molti voti (ma all’inizio dell’anno non ne ha ancora) mentre un singolo voto è sempre associato ad uno ed un solo studente (o studentessa).

Stiamo pertanto parlando di una associazione concettuale fra 1 istanza della classe Studente e tante istanze della classe Voto.

Nel diagramma di classe questo concetto si esprime collegando con una segmento le due classi ed indicando negli estremi del segmento la “cardinalità” minima e massima fra le classi stesse.

Nel caso in esame il diagramma diventa come in figura 2.

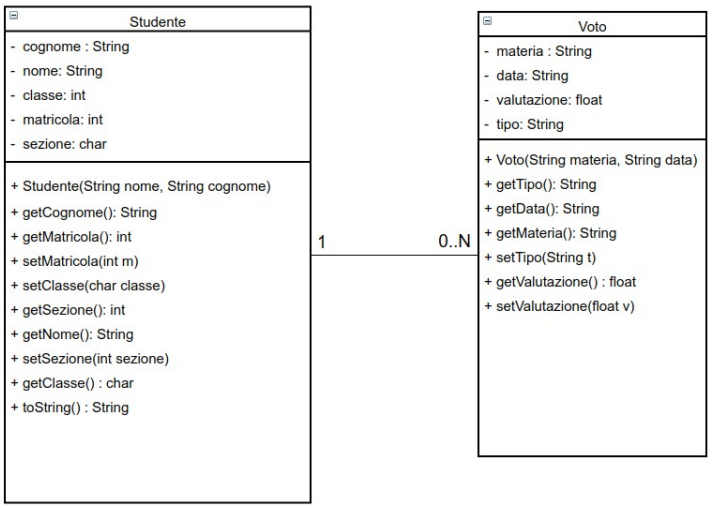


*figura 2: classi **Studente** e **Voto con associazione 1-N***

Procediamo ora a modificare attributi e metodi delle due classi per risolvere correttamente il pattern di programmazione in oggetto:

Passo 1)

Nella classe A (lato 1) dell'associazione aggiungere un attributo privato di tipo ArrayList<> come contenitore di oggetti di tipo classe B (lato 0..N)
Il nome dell'attributo inserito può essere il plurale della classe
Nel caso di specie: aggiungo nella classe **Studente** un attributo ***voti***: ***ArrayList<Voto>***

Passo 2)

Nella classe B dell'associazione aggiungere un attributo privato di tipo classe B
Il nome dell'attributo inserito può essere lo stesso della classe purché siano rispettate regole della Java Naming Conventions (vedi appendice al presente documento)
Nel caso di specie: aggiungo nella classe **Voto** un attributo **studente: Studente**

Passo 3)

Nella classe A creiamo un metodo per aggiungere elementi di tipo classe B all'attributo creato al passo 1. Questo metodo prende come parametro un oggetto di classe B e lo aggiunge alla struttura ArrayList corretta.
Nel caso in esame aggiungiamo un metodo **addVoto(Voto)** il cui corpo prevederà l'istruzione apposita.

```
public void addVoto(Voto v) {
        this.voti.add(v);
}
```

*snippet 1: Esempio di codice del metodo **addVoto** della classe **Studente***

Nulla impedisce, anzi è consigliato, di inserire anche altri metodi di gestione di questo nuovo attributo, ad esempio il metodo getVoti() come indicato in figura 3.

Passo 4)

Nella classe B creiamo i metodi per utilizzare e valorizzare l'attributo creato al passo 2.
Nell'esempio in figura 3 abbiamo realizzato i classici metodi *setter* e *getter.*

```
public Studente getStudente() {
        return this.studente;
}

public void setStudente(Studente s) {
        this.studente = s;
}
```

*snippet 2: Esempio di codice per la creazione dei metodi setter e getter nella classe **Voto***

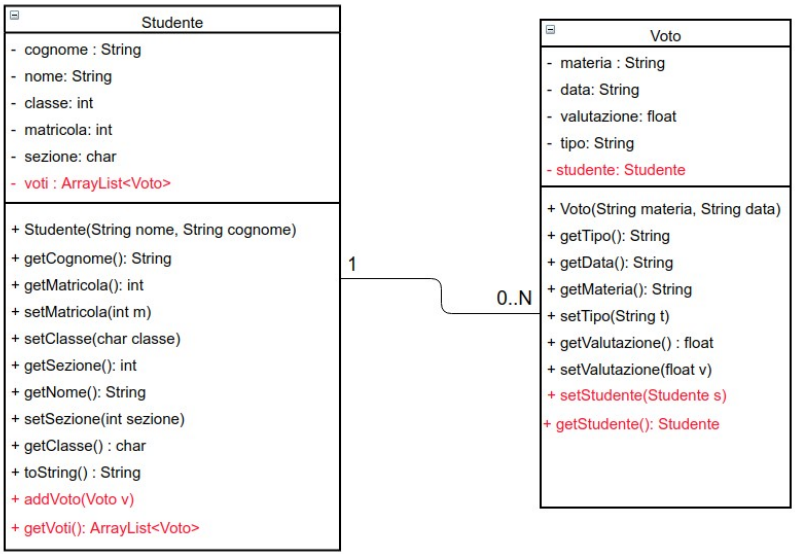


*figura 3: le classi **Studente** e **Voto** dopo l'applicazione dei passi indicati*

COMMENTO:

Al termine dell'applicazione dei passi indicati le due classi sono pronte per essere usate in maniera logicamente connessa.
Al lettore l'esercizio di trovare un modo per eliminare (o rendere superfluo) il metodo **setStudente()** della classe Voto o, se preferisce, il metodo **addVoto()** della classe Studente.
E' importante ricordarsi che i parametri passati sono sempre dei riferimenti ad oggetti e mai copie

Alleghiamo uno snippet dove si possono vedere in uso le classi suddette

```
Studente s = new Studente(“Steve”, “Jobs”);
Voto v. = new Voto("Matematica", "2020/01/23");
v.setTipo("Scritto");
v.setValutazione((float) 9);
s.addVoto(v); // aggancio il voto agli studenti
```

snippet 3: Esempio di uso delle classi Voto e Studente

APPENDICE:

Documento ufficiale ORACLE sulle Java Naming Conventions

https://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367

9 - Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give information about the function of the identifier-for example, whether it's a constant, package, or class-which can be helpful in understanding the code.

Identifier Type	Rules for Naming	Examples

Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.
	Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.
		`com.sun.eng`
		`com.apple.quicktime.v2`
		`edu.cmu.cs.bovik.cheese`

Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	`class Raster;` `class ImageSprite;`
Interfaces	Interface names should be capitalized like class names.	`interface RasterDelegate;` `interface Storing;`

Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	`run();`
		`runFast();`
		`getBackground();`

Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
		`int  i;`
		`char c;`
		`float myWidth;`

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	`static final int MIN_WIDTH = 4;`
		`static final int MAX_WIDTH = 999;`
		`static final int GET_THE_CPU = 1;`