Classi e oggetti: *parte I*

Imparare a progettare semplici classi di tipo *entity* e comprendere concetti base quali classe, oggetto e incapsulamento.

Creare classi *Entity*

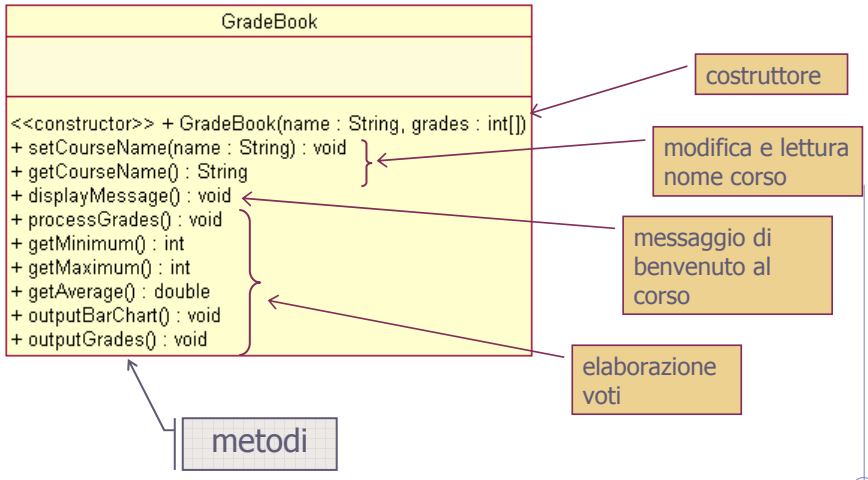
- Primo esempio di progettazione di una classe Entity:
  - Incapsulamento (oggetto)
  - Occultamento informazione ed implementazione
  - Identità oggetti
  - Classe
  - UML per classe

Specifica di oggetti

- Un oggetto GradeBook (registro di classe) ha le seguenti responsabilità:
  - Mantiene informazioni sul nome del corso e il voto di ogni studente del corso (ma non altre info sugli studenti)
  - Permette la modifica del nome del corso
  - Permette la lettura del nome del corso
  - Elabora i voti
    - calcola voto minimo
    - calcola voto medio
    - calcola voto massimo
    - stampa grafico
    - stampa elenco

Oggetto "GradeBook"

Vista Pubblica (o Esterna):



Incapsulamento

Incapsulamento:

raggruppamento in un'unica unità (oggetto) di ***metodi*** e di ***attributi*** . Gli attributi rappresentano lo stato dell'oggetto e sono accessibili e/o modificabili solo attraverso l'insieme dei metodi (interfaccia dell'oggetto)

Es. oggetto di tipo GradeBook ...

Incapsulamento: *attributi e metodi*

Attributi:

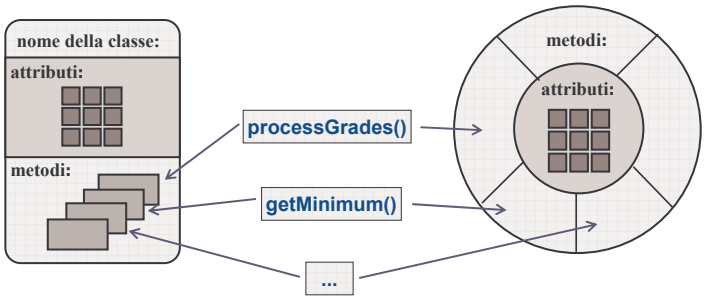
- rappresentano le informazioni che caratterizzano un oggetto, ovvero lo **stato** dell'oggetto
- solo i metodi di un oggetto possono accedere ad essi per leggerli e/o modificarli
- nessun altro oggetto può accedere direttamente ad un attributo: è necessario ricorrere ad uno dei metodi dell'oggetto (se esiste!)

Metodi:

- procedure o funzioni (blocchi di codice) che normalmente risultano visibili e richiamabili da altri oggetti
- permettono di modellare il **comportamento** dell'oggetto

Incapsulamento: *rappresentazione grafica*

Dal momento che **soltanto** le operazioni dell'oggetto possono leggere e aggiornare gli attributi di quest'ultimo, queste operazioni formano un anello di protezione attorno al nucleo centrale delle variabili implementate all'interno dell'oggetto



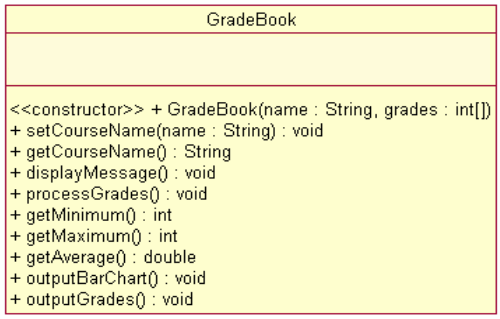
Occultamento informazione/implementazione

- Un'unità incapsulata può essere analizzata sia dall'esterno ("vista pubblica") che dall'interno ("vista privata")
- Buon incapsulamento: eliminazione, nella vista pubblica, della miriade di dettagli che caratterizzano la vista privata
- Questa eliminazione assume due forme:
  - occultamento delle informazioni (**information hiding**)
  - occultamento dell'implementazione (**implementation hiding**)

Occultamento: *definizione ed esempio*

L'**occultamento delle informazioni** e/o dell'**implementazione** è l'uso dell'incapsulamento per limitare la visibilità esterna di certe informazioni o decisioni di implementazione che sono interne alla struttura di incapsulamento

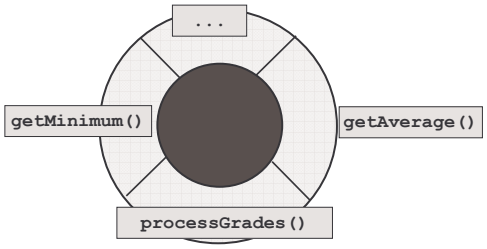
- Internamente, quali informazioni vengono memorizzate in un oggetto GradeBook? E come?
- Quale algoritmo è stato usato per realizzare il metodo `getAverage()`?



Occultamento: *considerazioni*

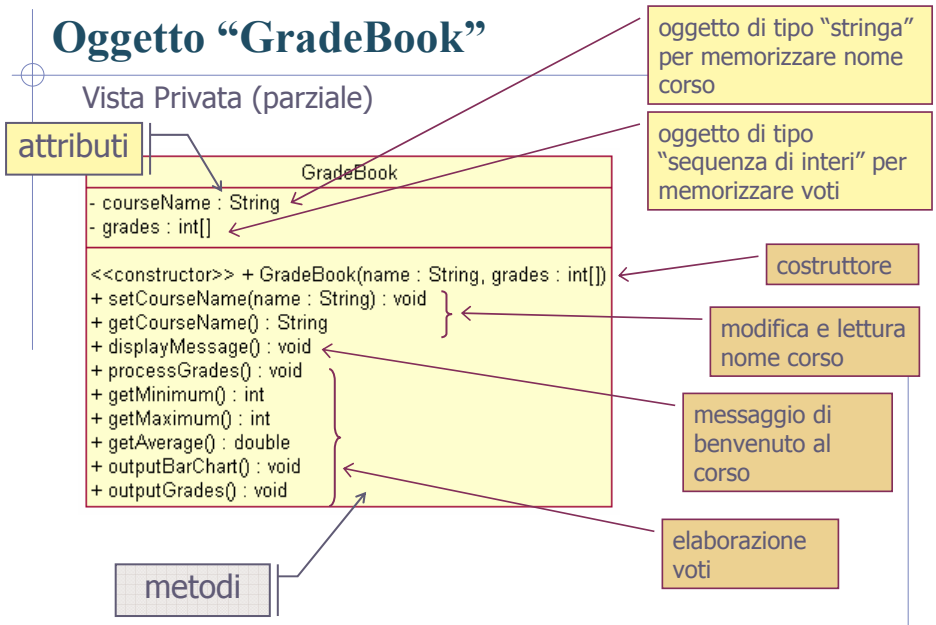
- L'occultamento delle informazioni e dell'implementazione è una tecnica molto efficace per mantenere il controllo sulla complessità del software
- Un oggetto appare come una scatola nera all'osservatore esterno: l'osservatore ha una conoscenza completa di ciò che l'oggetto può fare, ma non sa assolutamente nulla di come l'oggetto possa farlo o di come esso sia costruito internamente

L'oggetto GradeBook visto come una "scatola nera"



Oggetto "GradeBook"

Vista Privata (parziale)



Identità degli oggetti: *definizione*

L'**identità degli oggetti** è la proprietà per cui ciascun oggetto (a prescindere dalla sua classe o dallo stato corrente) può essere identificato e trattato come una distinta entità software

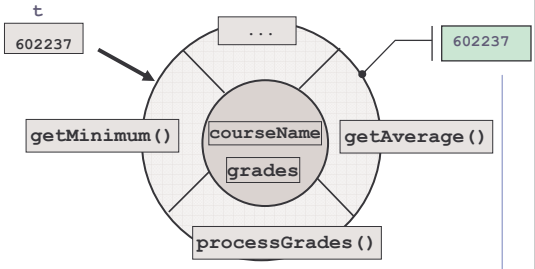
- Ogni oggetto ha una caratteristica unica che lo distingue dai suoi simili
- Tale caratteristica viene offerta dal meccanismo della maniglia *(handle)* dell'oggetto
- La maniglia è nota formalmente come *identificatore di oggetto (OID, Object IDentifier)*
- La maggior parte degli ambienti a oggetti crea automaticamente questo OID

Identità degli oggetti: *esempio*

Creazione di un oggetto di tipo GradeBook:

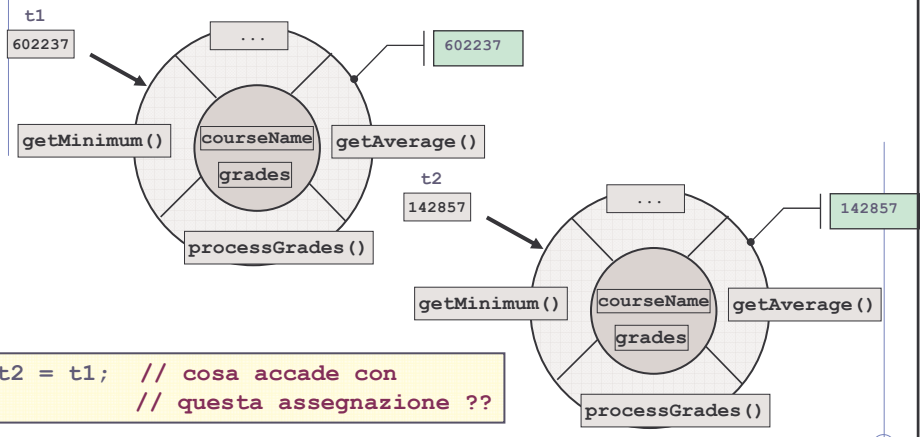
```
GradeBook t = new GradeBook();
```

1. Un oggetto mantiene la stessa maniglia per tutta la sua vita
2. Due oggetti non possono avere la stessa maniglia



Identità degli oggetti: *esempio*

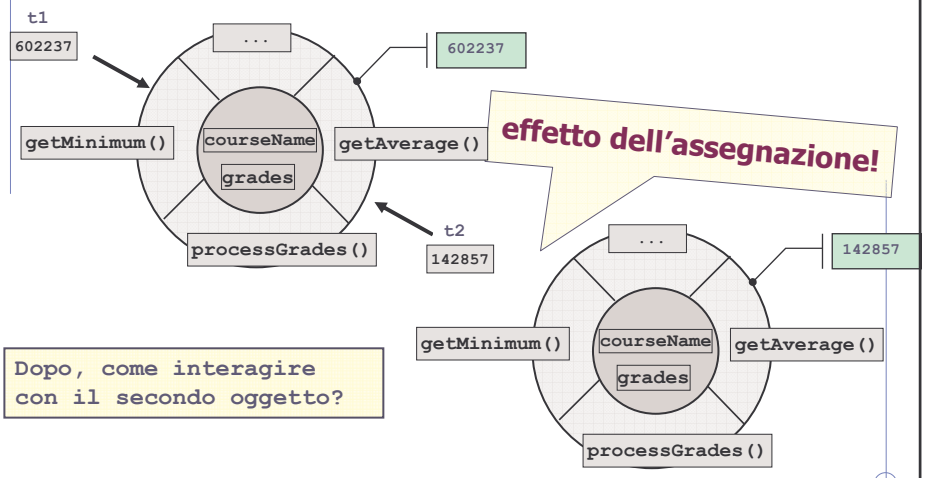
```
GradeBook t1 = new GradeBook();  // creazione di un oggetto
                                 // di tipo GradeBook
GradeBook t2 = new GradeBook()   // altro oggetto
```




```
t2 = t1;  // cosa accade con
          // questa assegnazione ??
```

Identità degli oggetti: *esempio*

```
t2 = t1; // assegnazione a t2 della maniglia di t1
```



Dopo, come interagire con il secondo oggetto?

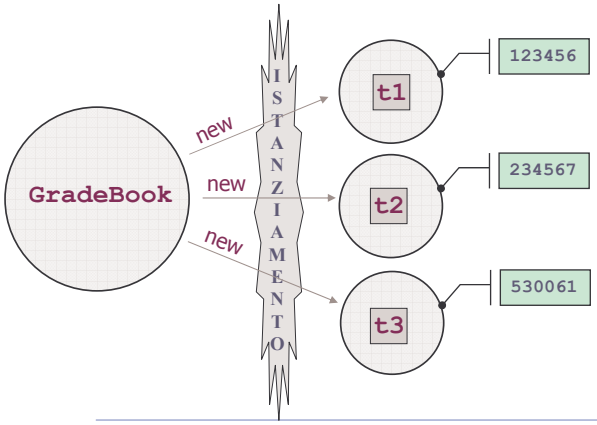
Classe: *definizione*

- **`new GradeBook()`** → istruzione che serve da modello per creare gli oggetti telefonino
- **`new GradeBook()`** → tutti gli oggetti creati con questa istruzione sono "strutturalmente identici"
- Oggetti "strutturalmente identici": stesse operazioni e variabili, quelle che il progettista ha definito nella classe **`GradeBook`**

Una **classe** è la sagoma a partire dalla quale vengono creati (istanziati) gli oggetti. Ogni oggetto ha la stessa struttura e comportamento della classe dalla quale è istanziato. Se l'oggetto `t` appartiene alla classe `C`, diciamo che "`t` è un'istanza di `C`"

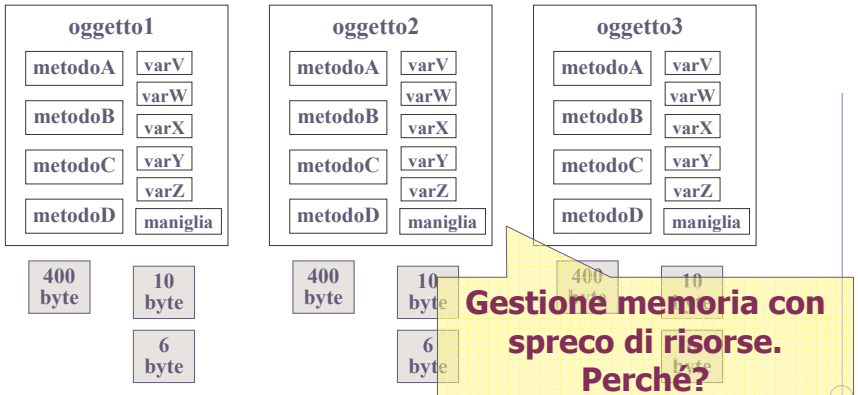
Differenza classe ↔ oggetto

- Una classe è ciò che viene progettato e programmato
- Gli oggetti rappresentano ciò che viene creato (a partire da una classe) in fase di esecuzione



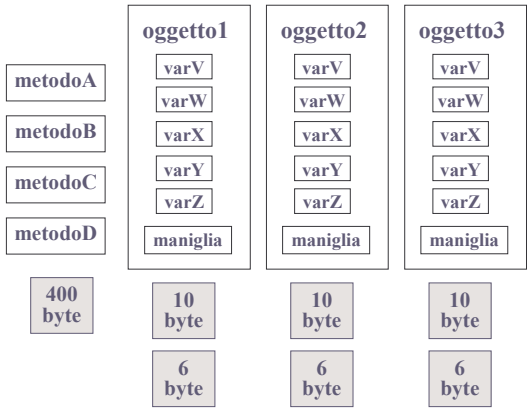
Classe: *gestione memoria* (1)

In linea di principio, l'istanziamento potrebbe produrre tante copie di Metodi ed Attributi quanti sono gli oggetti creati ⇒ esempio, uso di 3*416 = 1248 byte di memoria



Classe: *gestione memoria* (2)

Memoria **effettiva** utilizzata da tre oggetti della stessa classe: 400 + 3*16 = 448 byte



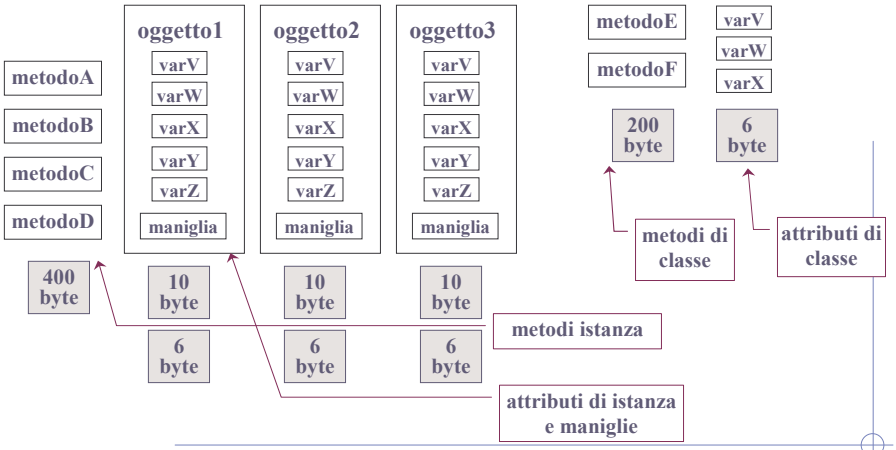
Classe: *attributi e metodi di classe*

- Oltre a **attributi** e **metodi di istanza** esistono anche **attributi** e **metodi di classe** (detti anche attributi e metodi **static**)
- Sono necessari per far fronte alle situazioni che non possono essere responsabilità di un qualunque oggetto singolo

- Esempi per la classe `GradeBook` :
  - Metodo di classe: `new`
  - Attributo di classe: `numeroGradeBookCreati`

Classe: *gestione memoria* (3)

Memoria effettiva utilizzata da tre oggetti e dal "meccanismo di classe":



Osservazioni

- Classe A senza "attributi di classe":
  - L'uso della classe A comporta occupazione di memoria solo nel momento in cui il programmatore istanzia il primo oggetto di A
- Classe B con "attributi di classe"
  - L'uso della classe B comporta immediatamente occupazione di memoria; il progettista di B ha dovuto prevedere allocazione di memoria per gli attributi di classe, e questi esistono prima ancora che vengano creati oggetti di B

Classe: notazione UML