Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

Object Oriented Programming: Fondamenti

Prof. Stefano Mariani

Dott. Alket Cecaj

Indice

- Introduzione all'OOP
- Classi vs. oggetti
- Campi e metodi
- Encapsulation

INTRODUZIONE ALL'OOP

Ricordate?

- *Per ora*
  - una **classe** è l'astrazione di un *concetto*, un *tipo di dato* che rappresenta uno **stato**, descritto da una serie di **attributi**, e un comportamento, descritto da una serie di **metodi**
    - e.g. la classe "docente"
  - un **oggetto** è una specifica *istanza* di una classe, con un preciso stato e un preciso comportamento
    - e.g. lo specifico docente "Stefano Mariani"

- [e, parlando dei tipi composti]
  - in generale, *qualunque **classe** Java*
    - *ogni classe* Java definisce infatti *un tipo* di dato
    - gli oggetti di una *stessa classe* sono dunque tutti dello *stesso tipo*

Flashback: Raffiniamo la definizione

Nella programmazione orientata agli oggetti una **classe** è un costrutto di un linguaggio di programmazione usato come modello per creare oggetti. Il modello comprende attributi e metodi che saranno condivisi da tutti gli oggetti creati (*istanze*) a partire dalla classe. Un "oggetto" è, di fatto, l'istanza di una classe.

Una **classe** è identificabile come un *tipo di dato astratto* che può rappresentare una persona, un luogo, oppure una cosa, ed **è** quindi **l'astrazione di un *concetto***, implementata in un software. Fondamentalmente, essa definisce al proprio interno lo *stato*, i cui dati sono memorizzati nelle cosiddette variabili *membro* o *attributi*, e il comportamento dell'entità di cui è rappresentazione, descritto da blocchi di codice riutilizzabili chiamati *metodi*.

- Riassumendo
  - **classe** = *modello* per definire oggetti
    - *stato* = insieme di **attributi**
    - *comportamento* = insieme di **metodi**
  - **oggetto** = particolare *istanza* di una certa classe

1. "**Everything is an object**"
   - un oggetto è un'*entità* che fornisce *operazioni* per essere manipolata
2. Un programma è un set di oggetti che si comunicano cosa fare **invocando metodi** (le *operazioni*)
3. Un oggetto può essere fatto di altri oggetti
4. Ogni oggetto è **istanza di una classe**
   - una classe (o, **tipo**) descrive il comportamento dei suoi oggetti
5. Tutti gli oggetti di una classe hanno gli *stessi metodi*
   - la classe indica esattamente le *operazioni* fornite, quindi per usare un oggetto basta sapere qual è la sua classe

- Supporta la **modellazione** software degli oggetti del mondo reale o astratto da riprodurre al giusto *livello di astrazione*
  - il problema ha a che fare con studenti ed esami?
  - implemento le classi `Studente` ed `Esame`
  - occorre calcolare la media voto di uno studente?
  - si ordina all'oggetto studente di calcolarla
- L'organizzazione del codice sotto forma di classi favorisce il **riuso**
  - gli oggetti possono essere *composti* in altri oggetti più complessi
  - i metodi evitano la *duplicazione* del codice

Il concetto di interfaccia

- L'insieme dei *metodi* prende il nome di **interfaccia***
  - ogni operazione ha effetto sullo *stato* e sul *comportamento* dell'oggetto
- Tipica struttura di una classe
  - *interfaccia*: la sola visibile ad altri oggetti
  - *membri* (o **campi**) della classe: gli oggetti di cui è costituita
  - *implementazione metodi*: cosa succede all'oggetto
    - come cambia lo *stato* (i membri dell'oggetto)
    - quali metodi invoca su altri oggetti
    - quale risultato fornisce (il valore di *ritorno* del metodo)

*In Java è un vero e proprio elemento del linguaggio, che vedremo se avremo tempo…

CLASSI vs. OGGETTI

Struttura di una classe

- **Nome** della classe
  - è anche il nome del *tipo (di dato)* che essa definisce
  - deve essere uguale al nome del file .java*
- **Campi**
  - descrivono la *struttura* e lo *stato* dell'oggetto
  - ogni oggetto/istanza della classe ha una propria copia delle variabili con propri valori
- **Metodi**
  - descrivono le *operazioni* disponibili e il *comportamento* corrispondente

*Non è sempre vero, tecnicamente, ma per noi sarà sempre così.

Struttura di una classe

```
class Studente {

  String nome;
  String cognome;
  int età;
  String matricola;
  Esame[] carriera;

  double calcolaMedia() {
    ...
  }

  void registraVoto(String materia, int voto) {
    ...
  }

}
```

Nome

definisce un nuovo tipo, Studente

Campi

struttura: l'entità studente è fatta di un nome, un cognome, etc.
stato: lo specifico studente ha una specifica carriera, che cambia nel tempo

Metodi (interfaccia)

operazioni: l'entità studente è in grado di "fare cose", "fornire servizi"
comportamento: a fronte dello specifico servizio richiesto, lo stato dello specifico oggetto può cambiare

La keyword this

- Java fornisce una variabile speciale denotata con **this** che contiene il *riferimento all'oggetto* corrente
  - può essere usata per *accedere ai campi* dell'oggetto dal corpo dei suoi metodi
    - notazione puntata: this.campo
  - può essere usata per *invocare metodi* dell'oggetto dal corpo dei suoi stessi metodi
    - notazione puntata: this.metodo(...)
- Evita ambiguità tra campi, variabili, e parametri

```
double calcolaMedia() {
  double sum = this.sommaVoti(this.carriera);
  return sum / this.carriera.length;
}
```

```
Studente(String nome, String cognome){
  this.nome = nome;
  this.cognome = cognome;
}
```

L'operatore new

- Crea un oggetto invocandone uno dei **costruttori**, ossia metodi speciali che hanno lo *stesso nome della classe*
  - *non* hanno alcun tipo di ritorno
  - possono avere degli argomenti
- Il *costruttore di default* (a zero argomenti) è implicitamente definito
  - solo fintantoché *non* ce ne sono altri
- Solitamente, nel corpo del costruttore si *inizializzano* i campi dell'oggetto

```
Studente(String nome, String cognome, int età) {
  this.nome = nome;
  this.cognome = cognome;
  this.età = età;
  this.matricola = this.generaMatricola();
  this.carriera = new Esame[30];
}
```

CAMPI e METODI

Campi

- Ognuno è una sorta di *variabile* (nome + tipo)
- Lo *stato* di un oggetto in un dato istante è il valore dei suoi campi in quell'istante
  - possono essere valori primitivi o altri oggetti
- Accessibile con la notazione **oggetto.campo**

```
void registraVoto(String materia, int voto) {
  Esame esame = new Esame();
  esame.materia = materia;
  esame.voto = voto;
  this.carriera[this.next] = esame;
}
```

Metodi

- Ognuno ha una *intestazione* (o **signature**) e un **corpo** (il codice tra “{ }”)
  - a sua volta l'intestazione ha
    - nome
    - tipo di ritorno
    - argomenti
- Definiscono il *comportamento* dell’oggetto
- *Invocabili* con la notazione **oggetto.metodo(argomenti)**
  - il comportamento conseguente è dato dall’esecuzione del corpo
  - che può *leggere* e *scrivere* il valore dei campi

```
double registraECalcolaMedia(String m, int v) {
  this.registraVoto(m, v);
  return this.calcolaMedia();
}
```

La keyword static

- Campi e metodi possono essere dichiarati **static**
  - i campi System.out e System.in
  - il main() dal quale parte un programma Java
  - il metodo Arrays.toString() (package java.util)
- Tali campi e metodi sono considerati *proprietà della classe*, *non* del singolo specifico oggetto
  - **Classe.metodo(argomenti)**
  - **Classe.campo**

```
...
this.matricola = Studente.generaMatricola();
...
static String generaMatricola(){
   ...
}
```

Le keyword void e null

- Può essere opportuno che un metodo non abbia alcun valore di ritorno
  - come nel caso del `main()`
  - la keyword **`void`** permette di specificare che *nessun* tipo di ritorno è atteso da un metodo
- Per specificare invece il concetto di "*nessun valore*", si usa la keyword **`null`**
  - le variabili che per tipo hanno un qualunque oggetto (*compresi* `Array` e `String`) vengono inizializzate a `null`

```
this.carriera = null;
...
void registraVoto(String materia, int voto) {
  ...
}
```

Il metodo main

- È un metodo speciale che rappresenta "*l'inizio del programma*"
  - viene invocato *automaticamente* dalla JVM
  - i suoi *argomenti** sono presi da linea di comando
  - signature: `public* static void main(String[] args)`
- Possono essercene tanti in un programma Java
  - ma *uno solo per classe*

*Pazienza, a breve…

Gli argomenti

- I metodi possono avere degli **argomenti** (o *parametri formali*)
  - variabili che forniscono *input* al metodo
  - *visibilità limitata* al corpo del metodo
- L'atto dell'invocazione *passa* al metodo gli argomenti (*parametri attuali*, questa volta) su cui il metodo lavorerà
- Il **passaggio dei parametri** avviene *sempre* **per copia**, ma copia del
  - **valore** (valido per tutti i *tipi primitivi* + `String`)
    - il valore in input al metodo è una copia del valore della variabile passata
  - **riferimento** (valido per tutto il resto, ossia gli *oggetti*)
    - il valore in input al metodo è una *copia del riferimento* alla variabile passata

Gli argomenti

```
void registraVoto(String materia, int voto) {
  Esame esame = new Esame();
  esame.materia = materia;
  esame.voto = voto;
  this.carriera[this.next] = esame;
}
```

Passaggio per copia
dei valori materia e voto

```
String materia = "FONDINF";
int voto = 30;
studente.registraVoto(materia, voto);
```

nel corpo del metodo, *non* ci sono
le variabili materia e voto,
ma delle loro copie
totalmente scorrelate

Passaggio per copia
del riferimento esame

```
void registraVoto(Esame esame) {
  this.carriera[this.next] = esame;
}
```

nel corpo del metodo, c'è la *copia del riferimento* esame

```
Esame esame = new Esame(...);
studente.registraVoto(esame);
```

Passaggio per copia di riferimento

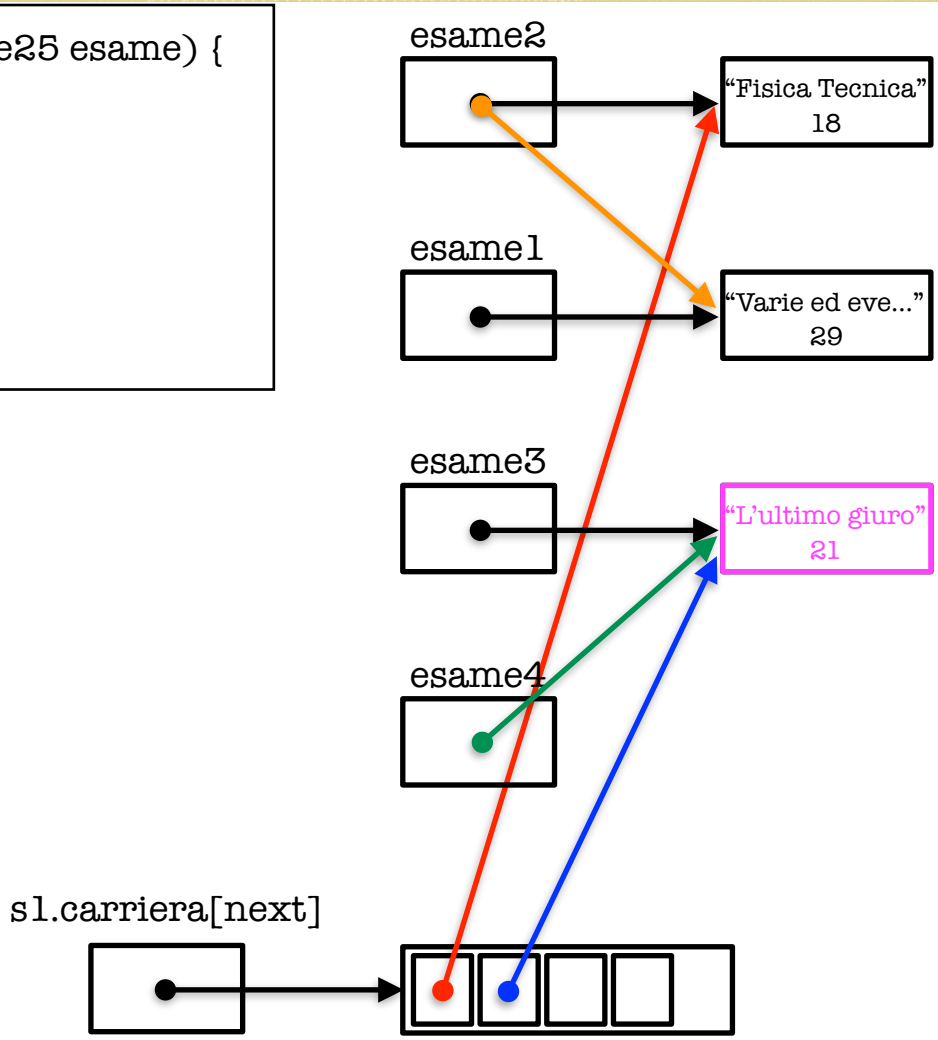
```
public double registraECalcolaMedia(EsameSlide25 esame) {
  this.registraVoto(esame);
  return this.calcolaMedia();
}
public void registraVoto(EsameSlide25 esame) {
  this.carriera[this.next] = esame;
  this.next++;
}
```

```
EsameSlide25 esame2 = new EsameSlide25();
esame2.setMateria("Fisica Tecnica");
esame2.setVoto(18);
sl.registraECalcolaMedia(esame2);

esame2 = esame1;

EsameSlide25 esame3 = new EsameSlide25();
esame3.setMateria("Basta ti prego");
esame3.setVoto(20);
EsameSlide25 esame4 = esame3;
sl.registraECalcolaMedia(esame3);

esame4.setMateria("L'ultimo giuro");
esame4.setVoto(21);
```



La keyword return

- I metodi possono *ritornare* un valore all'istruzione *chiamante* (che li ha invocati)
  - il *tipo* di questo valore è stabilito dalla *signature* del metodo
- La keyword **return** indica
  - la *terminazione* del metodo
  - l'eventuale *valore* ritornato
- In caso il metodo dichiari void come tipo di ritorno, non serve il return
  - il metodo termina alla chiusura della parentesi graffa corrispondente

```
double calcolaMedia() {
  ...
  return sum / this.carriera.length;
}
```

```
...
double media = studente.calcolaMedia();
...
```

ENCAPSULATION

Encapsulation e information hiding

- L'**information hiding** è il principio teorico su cui si basa la tecnica dell'**incapsulamento**
- Information hiding
  - i *dettagli implementativi* di una classe sono *nascosti* all'*utente* (= chi utilizza quella classe, anche un altro oggetto dunque)
  - dunque parte di un programma può nascondere informazioni incapsulandole in un costrutto dotato di *interfaccia*

Visibilità: public vs. private

- Keyword da anteporre a
  - classe
  - metodo (anche costruttori)
  - campo
- **public** : visibilità a *tutte* le classi (e dunque oggetti)
- **private** : visibilità *solo* alla classe corrente
  - consente di gestire a piacimento il concetto di *information hiding*
    - e dunque l'*incapsulamento*
  - *impedisce* l'uso della notazione oggetto.campo!
    - opportuni metodi `public` (chiamati comunemente "**getter**" e "**setter**") stabiliscono quali campi sono *accessibili e/o modificabili*

Fondamenti di Informatica

(L-Z)

Corso di Laurea in Ingegneria Gestionale

Object Oriented Programming: Fondamenti

Prof. Stefano Mariani

Dott. Alket Cecaj