Corso Java Base

Ing. Sandro Zacchino

email sandro.zacchino@gmail.com
github https://github.com/szacchino/Lezione200219

Problema

- Progettare le classi java necessarie ad elaborare la media degli esami sostenuti da un generico studente
- In questo problema riusciamo ad identificare due chiare entità: lo Studente e l'Esame
- In uno scenario reale i dati relativi ad uno studente potrebbero provenire da un database, da un file o da altre fonti

Progettazione

- Il nostro compito è quello di mappare le entità che hanno una definizione chiara in classi
- Le nostre classi saranno contenute in un package

Definizione di package

- Una classe ha un nome (solitamente con iniziale maiuscola) ed è definita all'interno di **package**.
- Un package è un contenitore con un nome che serve a distinguere classi che hanno lo stesso nome. Ad esempio

```
package it.universita;
```

e

```
package it.scuolasuperiore;
```

sono due package che potrebbero contenere una classe Studente ma le due classi potrebbero avere scopi e funzionamenti differenti

- All'interno di un package possono esserci più classi: per ciascuna possiamo indicare quale è visibile e quale no alle altri classi del contenitore.

Definizione di classe

- Una classe è la definizione di un tipo di dato complesso, dotato di uno stato interno e di metodi in grado di modificarlo
- lo **stato** è definito da attributi i quali possono essere di tipo primitivo (int, double, float, ecc) oppure di tipo complesso (altre classi)
- lo stato può comprendere tutte le classi *visibili* o sulle quali si ha accesso
- esistono le keyword di java che permettono di definire la visibilità di classi, attributi e metodi verso l'accesso da parte di una classe esterna

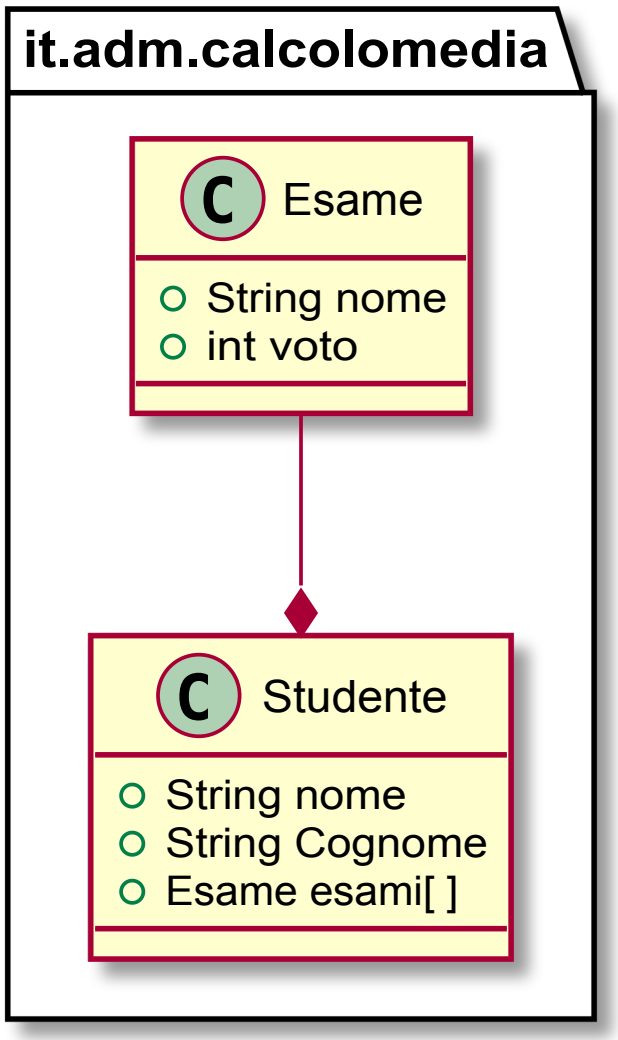
Modificatore	Package	Eredi	Esterno	Classe
public	Si	Si	Si	Si
protected	Si	Si	No	No
private	Si	No	No	No

Tipi di dati

- Java fornisce alcuni tipi di dati: alcuni di questi si identificano con *wrapper dei tipi primitivi*
- Questi wrapper sono le classi corrispondenti ai tipi primitivi
- i tipi primitivi ad esempio sono `int, float, double`
- i corrispondenti wrapper sono `Integer, Float, Double`
- l'uso dell'iniziale minuscola o maiuscola ci aiuta a capire quando stiamo usando un tipo primitivo e quando una classe

Diagramma delle classi

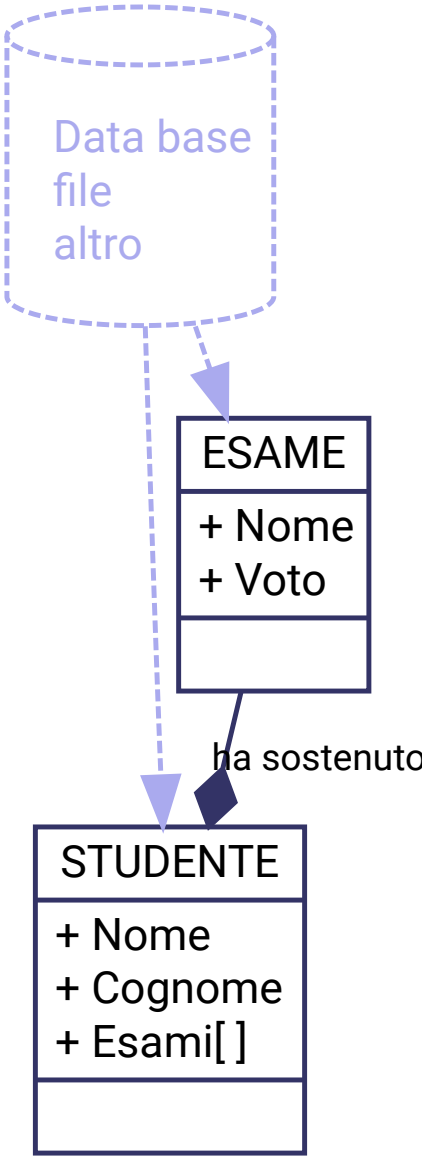
- Un primo progetto potrebbe essere rappresentato dal seguente diagramma UML (class diagram)



- **N.B.:** la presente non è necessariamente la soluzione finale

Un possibile schema di funzionamento

- Possiamo immaginare le nostre classi connesse ad altri pezzi del nostro sistema informativo



Scrittura delle classi

- Ciascuna delle classi progettate andrà scritta in un file con lo stesso nome della classe ed estensione `.java`:

```
Studente.java
Esame.java
```

Studente.java

```java
package it.adm.calcolomedia;

public class Studente {
    public String nome;
    public String cognome;
    public Esame esami[];
}
```

Esame.java

```java
package it.adm.calcolomedia;

public class Esame {
    public String nome;
    public int voto;
}
```

Getters e Setters

- per quanto corrette, le suddette classi non seguono le *best practices* e non sfruttano i vantaggi che alcuni framework garantiscono a chi le segue
- a meno di alcune eccezione lo stato di una classe dovrebbe essere definito con visibilità `private`
- per consentire l'accesso in lettura o scrittura dello stato di una classe si utilizzano i metodi di tipo *getter* (per la lettura) o *setter* (per la scrittura): il nome di questi metodi è che il verbo *to get* indica *prendere* e il verbo *to set* indica *impostare*
- per ciascun attributo privato creeremo quindi dei semplici getter e setter (eclipse offre un automatismo per questo)

Studente.java

```
package it.adm.calcolomedia;

public class Studente {
        private String nome;
        private String cognome;
        public Esame esami[];

        public String getNome() {
                return nome;
        }

        public void setNome(String nome) {
                this.nome = nome;
        }

        public String getCognome() {
                return cognome;
        }

        public void setCognome(String cognome) {
                this.cognome = cognome;
        }
}
```

- Per ora non scriviamo il getter/setter dell'attributo *esami*

La prima parte del programma

```
1  package it.adm.calcolomedia;
2
3  public class TestEsame {
4          public static void main(String[] args) {
5                  Studente studente = new Studente();
6                  studente.setNome("Mario");
7                  studente.setCognome("Rossi");
8          }
9  }
```

- alla linea 5 creiamo una *istanza* della classe Studente invocando un metodo preceduto da **new**

Dove è definito quel metodo?

- nelle linee successive invece invochiamo i setter che alterano lo stato dell'istanza

Il costruttore

- La linea 5 del precedente listato crea una istanza della classe Studente invocando il costruttore
- Tutte le classi Java, anche quando non specificato, funzionano seguendo la definizione di una classe speciale chiamata Object
- Per questo motivo si dice che la classe Studente (così come la classe Esame) **eredita** dalla classe Object
- i tipi primitivi (int, float, double, ecc) non sono classi e quindi non ereditano alcun funzionamento da nessuno
- i wrapper dei tipi primitivi (Integer, Float, Double) invece sono classi
- quando nella classe manca il costruttore, viene invocato quello della classe madre (quindi quello di Object); stesso vale per gli altri metodi

Il costruttore

- Il costruttore può essere scritto quando si vuole inizializzare l'istanza in modo particolare o in base a dei parametri

```
1  public Studente(String nome, String cognome) {
2      this.nome = nome;
3      this.cognome = cognome;
4      this.esami = new Esame[28];
5  }
```

- Alla linea 4 vediamo come si inizializza un array di istanze.
- Alcuni framework potrebbero richiedere un costruttore senza parametri anche vuoto

Cosa succede se nel main() aggiungiamo:
System.out.println(studente.esami[0].getVoto());

Aggiungiamo un esame

```
package it.adm.calcolomedia;

public class TestEsame2 {
	public static void main(String[] args) {
		Studente studente = new Studente("Mario", "Rossi");
		studente.esami = new Esame[28];

		Esame esame = new Esame();
		esame.setNome("Analisi Matematica I");
		esame.setVoto(30);
		studente.esami[0] = esame;

		System.out.println(studente.esami[0].getVoto());
	}
}
```

Miglioramento del modello

- Cosa succede se più studenti fanno lo stesso esame?
- Cosa occorre rimodellare per ottimizzare la gestione delle informazioni?

Class Diagram