

# 15 - Ereditarietà

Programmazione e analisi di dati  
Modulo A: Programmazione in Java

Paolo Milazzo

Dipartimento di Informatica, Università di Pisa  
<http://pages.di.unipi.it/milazzo>  
milazzo@di.unipi.it

Corso di Laurea Magistrale in Informatica Umanistica  
A.A. 2019/2020

## Classi simili...

A volte in uno stesso programma ci sono classi che descrivono **cose simili tra loro**.

Esempio:

- supponiamo di voler realizzare un programma per la **gestione degli insegnamenti** di un corso di laurea
- per ogni insegnamento vogliamo sapere chi è il **docente responsabile** e chi sono gli **studenti frequentanti**
- per descrivere docenti e studenti ci servono due classi: Professore e Studente

# Studenti e professori (1)

Partiamo con la classe Studente

```
public class Studente {  
  
    private String nome;           // nome e cognome  
    private String indirizzo;     // indirizzo  
    private int matricola;        // numero di matricola  
    private int anno;             // anno di frequentazione  
  
    private static int ultimaMatricola = 0 // variabile statica  
  
    // costruttore  
    public Studente(String nome, String indirizzo) {  
        this.nome = nome;  
        this.indirizzo = indirizzo;  
  
        // genera la matricola usando la variabile statica  
        this.matricola = ultimaMatricola + 1;  
        ultimaMatricola++;  
  
        // si assume che un nuovo studente sia al primo anno  
        this.anno = 1;  
    }  
}
```

(segue)

## Studenti e professori (2)

(segue Studente)

```
// fornisce il nome dello studente
public String getNome() { return nome; }

// fornisce l'indirizzo dello studente
public String getIndirizzo() { return indirizzo; }

// consente di modificare l'indirizzo
public void setIndirizzo(String indirizzo) {
    this.indirizzo = indirizzo;
}

// fornisce la matricola
public int getMatricola() { return matricola; }

// fornisce l'anno di frequentazione
public int getAnno() { return anno; }
```

(segue)

## Studenti e professori (3)

(segue Studente)

```
// modifica l'anno di frequentazione
public void setAnno(int anno) {
    if (anno>0) this.anno = anno;
}

// verifica se lo studente e' fuoricorso
public boolean isFuoricorso() { return (anno>5); }

// stampa le informazioni sullo studente
public void visualizza() {
    System.out.println("    Nome: " + nome);
    System.out.println("Indirizzo: " + indirizzo);
    System.out.println("Matricola: " + matricola);
    System.out.println("    Anno: " + anno);
    if (isFuoricorso())
        System.out.println("    ( Studente fuoricorso )");
    else
        System.out.println("    ( Studente in corso )");
    System.out.println();
}
}
```

# Studenti e professori (4)

Ora è la volta della classe Professore

```
public class Professore {  
  
    private String nome;           // nome e cognome  
    private String indirizzo;     // indirizzo  
    private String codiceDocente; // codice del docente  
    private String dipartimento;  // dipart. di afferenza  
  
    // costruttore  
    public Professore(String nome, String indirizzo,  
                       String codiceDocente,  
                       String dipartimento) {  
  
        this.nome = nome;  
        this.indirizzo = indirizzo;  
        this.codiceDocente = codiceDocente;  
        this.dipartimento = dipartimento;  
    }  
}
```

(segue)

## Studenti e professori (5)

(segue Professore)

```
// fornisce il nome del professore
public String getNome() { return nome; }

// fornisce l'indirizzo del professore
public String getIndirizzo() { return indirizzo; }

// consente di modificare l'indirizzo
public void setIndirizzo(String indirizzo) {
    this.indirizzo = indirizzo;
}
```

(segue)

## Studenti e professori (6)

(segue Professore)

```
// fornisce il codice docente
public String getCodiceDocente() {
    return codiceDocente;
}

// fornisce il dipartimento
public String getDipartimento() {
    return dipartimento;
}

// stampa le informazioni sul professore
public void visualizza() {
    System.out.println("        Nome: Prof. " + nome);
    System.out.println("    Indirizzo: " + indirizzo);
    System.out.println("        Codice: " + codiceDocente);
    System.out.println("Dipartimento: " + dipartimento);
    System.out.println();
}
}
```



## Studenti e professori (7)

Tutto OK.... ma quanto codice Java **ripetuto**...

Studente e Professore hanno diverse **cose in comune**:

- due variabili d'istanza: nome e indirizzo
- alcuni metodi: `getNome()`, `getIndirizzo()` e `setIndirizzo()`

Questi membri in effetti non descrivono caratteristiche **specifiche** degli studenti e dei professori

- sono caratteristiche **comuni** di tutte le **persone**

Isoliamo dunque i membri condivisi di Studente e Professore in una nuova classe **Persona**

- aggiungiamo anche un costruttore e un metodo `visualizza()`

# Persone

Ecco la classe Persona

```
public class Persona {  
  
    private String nome;           // nome e cognome  
    private String indirizzo;     // indirizzo  
  
    // costruttore  
    public Persona(String nome, String indirizzo) {  
        this.nome = nome;  
        this.indirizzo = indirizzo;  
    }  
  
    public String getNome() { return nome; }  
  
    public String getIndirizzo() { return indirizzo; }  
  
    public void setIndirizzo(String indirizzo) {  
        this.indirizzo = indirizzo;  
    }  
  
    // visualizza i dati della persona  
    public void visualizza() {  
        System.out.println("    Nome: " + nome);  
        System.out.println("Indirizzo: " + indirizzo);  
        System.out.println();  
    }  
}
```

## Estensione di una classe (1)

Ciò che si può fare ora è ridefinire le classi `Studente` e `Professore` come **estensioni** della classe `Persona`

Le classi `Studente` e `Professore` **erediteranno** così i membri di `Persona` (senza doverli definire di nuovo)

- tali membri saranno quindi definiti una volta sola (in `Persona`)

Per definire una classe come estensione di un'altra si deve usare la primitiva **`extends`**

- deve essere usata all'inizio della definizione della classe

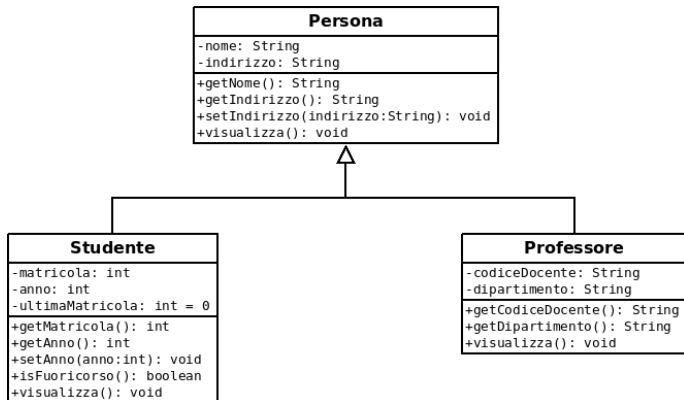
```
public class Studente extends Persona {  
    ...  
}
```

## Estensione di una classe (2)

Terminologia:

- La classe che viene estesa (es. Persona) è detta **superclasse**
- La classe che estende (es. Studente) è detta **sottoclasse**

Questa terminologia è legata al fatto che solitamente la relazione di estensione viene rappresentata così (superclasse sopra, sottoclassi sotto):



## Estensione di una classe (3)

Cosa succede quando si crea un oggetto di una classe che ne estende un'altra?

```
public class Studente extends Persona {  
    ...  
}
```

Altrove:

```
...  
Studente s = new Studente();  
...
```

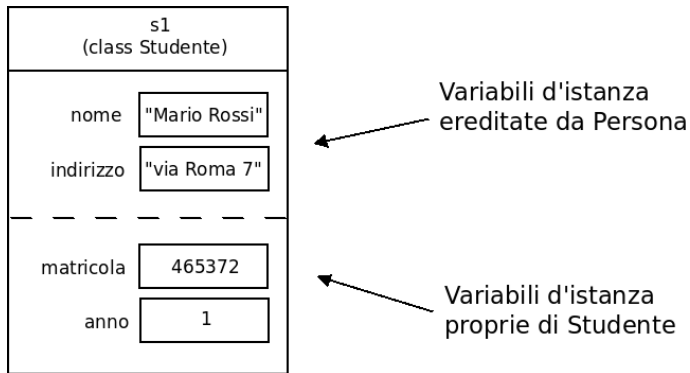
## Estensione di una classe (4)

La creazione di un oggetto di una sottoclasse (es. `Studente`) segue i seguenti passi:

1. Viene creato un oggetto della superclasse (es. `Persona`)
  - ▶ tramite il costruttore di default (`Persona()`)
  - ▶ oppure, tramite un altro costruttore richiamato **esplicitamente** tramite il riferimento **super** (simile a `this...`)
2. Vengono aggiunte all'oggetto le variabili della sottoclasse
3. Viene eseguito il costruttore della sottoclasse (es. `Studente`)

## Estensione di una classe (5)

Il risultato della creazione (nella memoria Heap) di un oggetto di una sottoclasse è quindi il seguente:



# Nuove versioni di Studente e Professore (1)

Ridefiniamo la classe **Studente**

```
public class Studente extends Persona {  
  
    private int matricola;    // numero di matricola  
    private int anno;        // anno di frequentazione  
  
    private static int UltimaMatricola = 0;  
  
    // costruttore (uguale a prima)  
    public Studente(String nome, String indirizzo) {  
  
        // chiama il costruttore di Persona  
        super(nome, indirizzo);  
  
        // inizializza le altre variabili  
        this.matricola = ultimaMatricola + 1;  
        ultimaMatricola++;  
        this.anno = 1;  
    }  
  
    (segue)  
  
}
```



## Nuove versioni di `Studente` e `Professore` (2)

`super` è un riferimento all'oggetto stesso (come `this`) ma che consente di utilizzare i costruttori e i metodi della superclasse

Ad esempio, dentro a `Studente`:

- `this(...)` richiama un costruttore della classe `Studente`
- `super(...)` richiama un costruttore della classe `Persona`

NOTA: Come `this(...)` anche `super(...)` deve essere eseguito come primo comando di un costruttore.

In caso la chiamata `super(...)` sia omessa, viene invocato **automaticamente** il costruttore senza parametri della superclasse

Quindi, nel costruttore della sottoclasse la chiamata `super(...)` può essere omessa **solo se**:

- la superclasse **prevede** un costruttore senza parametri (es. il costruttore di default)

## Nuove versioni di Studente e Professore (3)

(segue Studente)

```
public int getMatricola() { return matricola; }
public int getAnno() { return anno; }

public void setAnno(int anno) {
    if (anno>0) this.anno = anno;
}

public boolean isFuoricorso() { return (anno>5); }

// stampa le informazioni sullo studente
// NOTA: per recuperare nome e indirizzo deve usare
// i metodi pubblici di Persona!
public void visualizza() {
    System.out.println("    Nome: " + getNome());
    System.out.println("Indirizzo: " + getIndirizzo());
    System.out.println("Matricola: " + matricola);
    System.out.println("    Anno: " + anno);
    if (isFuoricorso())
        System.out.println(" ( Studente fuoricorso )");
    else
        System.out.println(" ( Studente in corso )");
    System.out.println();
}
}
```

## Nuove versioni di Studente e Professore (4)

Il metodo `visualizza()` di `Studente` richiede due osservazioni

**OSSERVAZIONE 1:** `nome` e `indirizzo` sono variabili **private** di `Persona`

- Il fatto che `Studente` sia sottoclasse di `Persona` non la autorizza ad accedere alle variabili private...

**CONSEGUENZE:**

- devo usare i **metodi pubblici** (se disponibili, come in questo caso)
- altrimenti dovrei **cambiare il modificatore** di visibilità di `nome` e `indirizzo` **in `Persona`**
  - ▶ Ma quale devo usare???

## Nuove versioni di Studente e Professore (5)

Rivediamoli ancora una volta:

<code>private</code>	Utilizzabile solo all'interno della stessa classe
senza modificatore	Utilizzabile solo nel <code>package</code> che contiene la classe
<code>protected</code>	Utilizzabile nel <code>package</code> che contiene la classe, e in tutte le classi che <code>ereditano</code> da essa
<code>public</code>	Utilizzabile ovunque

Visto che non ho definito packages (quindi uso il package default) i casi "senza modificatore", `protected` e `public` sono di fatto **equivalenti**

- Ergo, se non voglio rendere completamente pubbliche le variabili di `Persona` devo inserire le mie classi in un `package`:
  - ▶ Ometterò il modificatore per limitare la visibilità alle **sole classi del package**
  - ▶ oppure userò `protected` per estendere la visibilità **anche a sottoclassi** di `Persona` **in altri packages**

## Nuove versioni di Studente e Professore (6)

**OSSERVAZIONE 2:** `visualizza()` era già stato definito in `Persona`

- Questo è un caso di **overriding** di un metodo
- La sottoclasse ridefinisce (sostituendolo) un metodo della superclasse

Per ottenere l'overriding di un metodo è necessario che **la firma** del metodo nella sottoclasse sia identica a quella del metodo nella superclasse!

- in caso contrario avremmo **overloading**: entrambi i metodi (il vecchio e il nuovo) sarebbero abilitati ed eseguiti in base alla firma

**ATTENZIONE:** **overloading** e **overriding** di metodi sono due cose diverse:

- l'overloading è un meccanismo che consente di avere **più metodi con lo stesso nome**
- l'overriding è un meccanismo che consente di **sostituire un metodo** di una superclasse

# Nuove versioni di Studente e Professore (7)

Ridefiniamo anche la classe Professore

```
public class Professore extends Persona {

    private String codiceDocente;
    private String dipartimento;

    public Professore(String nome, String indirizzo,
                      String codiceDocente,
                      String dipartimento) {

        super(nome, indirizzo);
        this.codiceDocente = codiceDocente;
        this.dipartimento = dipartimento;
    }

    public String getCodiceDocente() { return codiceDocente; }

    public String getDipartimento() { return dipartimento; }

    public void visualizza() {
        System.out.println("        Nome: Prof. " + getNome());
        System.out.println("        Indirizzo: " + getIndirizzo());
        System.out.println("        Codice: " + codiceDocente);
        System.out.println("Dipartimento: " + dipartimento);
        System.out.println();
    }
}
```

## Nuove versioni di Studente e Professore (7)

Per concludere, un main:

```
public class UsaStudenteProfessore {  
    public static void main(String[] args) {  
        Persona x = new Persona("Gino", "Via Milano 10");  
        x.visualizza();  
  
        Studente s1 = new Studente("Carlo Bianchi", "Via Garibaldi 71");  
        s1.visualizza();  
  
        Studente s2 = new Studente("Mario Rossi", "Via Mazzini 11");  
        s2.visualizza();  
  
        Professore p1 = new Professore("Mario Rossi",  
                                       "Via Marconi 10", "a1123",  
                                       "Dipartimento di Informatica");  
        p1.visualizza();  
  
        Professore p2 = new Professore("Luigi Verdi",  
                                       "Via Verdi 70", "a9521",  
                                       "Dipartimento di Biologia");  
        p2.visualizza();  
    }  
}
```