

La programmazione OO

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
<http://www.dimi.uniud.it/mizzaro/>
mizzaro@dimi.uniud.it
Programmazione, lezione 17
29 novembre 2006

Riassunto

- "Mattoni" (cap 2)
- Sequenza, Selezione, Iterazione (3)
- Array (4)
- Sottoprogrammi, metodi (5)
- Ricorsione (6)
- TDA (7)
- OO = TDA + scambio msg + eredità + polimorfismo
- TDA -> OO, Scambio messaggi (8)

Stefano Mizzaro - OO

2

Oggi

- Esempio
- Ereditarietà
- Polimorfismo

Stefano Mizzaro - OO

3

Esempio

- **Punto, Cerchio (e Quadrato)** con approccio OO
- Vediamo il codice
 - Provare a eseguirlo, modificarlo, ecc.
 - Ragionare su classi, istanze, stack, heap

Stefano Mizzaro - OO

4

Punto.java (OO)

```
class Punto {
    private double x; private double y;
    public Punto() {this(0,0);}
    public Punto(double x, double y) {
        this.x = x; this.y = y;
    }
    public void set(double x, double y) {
        this.x = x; this.y = y;
    }
    public void setX(double x) { this.x = x; }
    public void setY(double y) { this.y = y; }
    public double getX() { return x; }
    public double getY() { return y; }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

Stefano Mizzaro - OO

5

Cerchio.java (OO)

```
class Cerchio {
    private Punto centro; private double raggio;
    public Cerchio() {this(new Punto(),0);}
    public Cerchio(double x, double y, double r) {
        this(new Punto(x,y),r);
    }
    public Cerchio(Punto c, double r) {
        centro = c;
        raggio = r;
    }
    public void setCentro(Punto c) {centro = c;}
    public void setRaggio(double r) {raggio = r;}
    public Punto getCentro() {return centro;}
    public double getRaggio() {return raggio;}
    public String toString() {
        return "cerchio: centro " + getCentro() +
            " raggio " + getRaggio();
    }
}
```

Grafica.java (OO)

```

class Grafica {
    public static void main (String[] args) {
        Punto p1 = new Punto();
        Punto p2 = new Punto(1,1);
        Cerchio c1 = new Cerchio();
        System.out.println(c1);
        Cerchio c2 = new Cerchio(p1,0);
        System.out.println(c2);
        p2.set(1.2,3.3);
        c1.setCentro(new Punto(12,12));
        System.out.println(c1);
        System.out.println(c1.getCentro());
        System.out.println((c1.getCentro()).getX());
        Cerchio c3 = new Cerchio(p2,1);
        System.out.println(c3);
    }
}
    
```

Stefano Mizzaro - OO 7

Scaletta

- Esempio
- Ereditarietà
- Polimorfismo

Stefano Mizzaro - OO 8

Terzo ingrediente: ereditarietà

- La classe **Persona**...
- ... e la classe **Studente**

| | |
|---|---|
| Persona - nome : String - cognome : String - dataDiNascita : Date + eta() : int + setNome(String) : void + ...() | Studente - nome : String - cognome : String - dataDiNascita : Date - matricola : int + eta() : int + setNome(String) : void + ...() + getMatricola() + setMatricola() |
|---|---|

Stefano Mizzaro - OO 9

Un problema

- Duplicazione di codice
 - Fatica inutile
 - Pericolo di incoerenze
- Soluzione: ereditare
 - studente** eredita da **Persona** attributi e metodi (e ne aggiunge, sovrascrive, ...)

Stefano Mizzaro - OO 10

Persona e Studente

| | |
|---|--|
| Persona - nome : String - cognome : String - dataDiNascita : Date + eta() : int + setNome(String) : void + ...() | |
| Studente - matricola : int + getMatricola() : int + setMatricola() : void | |

Stefano Mizzaro - OO 11

... e Lavoratore

| | |
|---|---|
| Persona - nome : String - cognome : String - dataDiNascita : Date + eta() : int + setNome(String) : void + ...() | |
| Lavoratore - stipendio : int + setStipendio() : void + getStipendio() : int | Studente - matricola : int + getMatricola() : int + setMatricola() : void |

Stefano Mizzaro - OO 12

Ereditarietà in Java

- Parola riservata **extends**
- class Studente extends Persona** {
 - ... e **Studente** ha metodi e attributi di **Persona**
- N.B. Ereditarietà troppo enfattizzata...

Stefano Mizzaro - OO 13

La classe Persona

```
class Persona {
    private String nome;
    private String cognome;
    private Data nascita;

    public void setNome(String nome) { ... }
    public String getNome() { ... }
    public void setNascita( ... ) { ... }
    public int eta() { ... }
    public String toString() { ... }
    ...
}
```

- ... una classe come un'altra...

Stefano Mizzaro - OO 14

La classe studente

```
class Studente extends Persona {
    private int matricola;

    public int getMatricola() { ... }
    public void setMatricola( ... ) { ... }
    ...
}
```

- Oltre a quello che c'è in **Studente**, anche quello che c'è in **Persona**
- ... ma ciò che è privato in **Persona** non è visibile in **Studente**

Stefano Mizzaro - OO 15

Uso

```
...
Persona p1 = new Persona( ... );
Studente s2 = new Studente( ... );
...
p1.setNome("Gino");
s2.setMatricola(1234);
s2.setNascita( ... );
...
System.out.println(p1.getNome());
System.out.println(s2.eta());
...
```

- Uno studente è una persona (!)
- Una persona potrebbe non essere uno studente

Stefano Mizzaro - OO 16

Quindi, l'ereditarietà

- Consente a una classe di ereditare da altre classi, evitando duplicazione di codice
- Consente di vedere un'istanza di una sottoclasse "come se" fosse un'istanza della sovraclassa
 - Vedo uno studente come se fosse una persona (!)
- extends** in Java

Stefano Mizzaro - OO 17

instanceof

- Operatore (parola riservata) Java
- Dice se (il valore contenuto in) una variabile è istanza di una classe

```
Persona p = new Persona( ... );
Studente s = new Studente( ... );
...
System.out.println(p instanceof Persona);
System.out.println(p instanceof Studente);
System.out.println(s instanceof Studente);
System.out.println(s instanceof Persona);
```

```
true
false
true
true
```

Stefano Mizzaro - OO 18

Ereditarietà e private

- Si eredita tutto (anche attributi e metodi **private**) ma non tutto è visibile (solo ciò che non è **private**)

```
class A {
    private int x;
    public int getX(){
        return x;
    }
}

class B extends A {
    public void m(){
        System.out.println(x);
    }
}

class B extends A {
    public void m() {
        System.out.println(getX());
    }
}
```

Stefano Mizzaro - OO 19

La sovrascrittura (overriding)

- Un metodo di una sottoclasse può sovrascrivere un metodo con la stessa firma in una sopraclasse
- Es.: visualizziamo anche la matricola di uno studente (e quindi il `toString` va ridefinito/sovrascritto nella sottoclasse)

```
class Studente extends Persona {
    private int matricola;
    ...
    public String toString() {
        return ...//COPIO il toString di Persone
        + matricola; // e aggiungo la matricola
    }
}
```

- `System.out.println(p2)`, con `p2` istanza di `Studente`, visualizza anche la matricola

Stefano Mizzaro - OO 20

Sovrascrittura e sovraccarico

- Sovrascrittura: nella sottoclasse posso specializzare il comportamento
 - N.B. Sovrascrittura (*overriding*) \neq sovraccarico (*overloading*)
 - Sovraccarico: nome =, firme \neq
 - Sovrascrittura: nome =, firme =

Stefano Mizzaro - OO 21

Sovrascrittura \neq Sovraccarico

- Sovrascrittura (firme =) \neq sovraccarico (firme \neq)

```
class A {
    public void m1() { ... }
    public void m2() { ... }
    public void m2(.) { ... }
    public void m3() { ... }
}

class B extends A {
    public void m1() { ... }
    public void m3(.) { ... }
}
```

Stefano Mizzaro - OO 22

Il super (1/2)

- Duplicazione di codice...@#}\$[%&>~?
- Con **super** si fa riferimento alla sopraclasse
- Esempio:

```
class Studente extends Persona {
    private int matricola;
    ...
    public String toString() {
        return
            super.toString() // chiamo toString di Persone
            + matricola; // E aggiungo la matricola
    }
}
```

Stefano Mizzaro - OO 23

Il super (2/2)

- Il **super** nel costruttore


```
public B() {
    super();
    ...;
}
```
- Il costruttore di default:


```
public B() { super(); }
```
- super** (riferimento alla sopraclasse) simile al **this** (riferimento alla classe)

Stefano Mizzaro - OO 24

Relazioni fra classi (1/2)

- **È-un (is-a)**
 - Relazione fra sopraclassi e sottoclassi
 - Il trapezio è un quadrilatero, che è un poligono, che è una figura geometrica bidimensionale, che è una figura geometrica; l'automobile è un veicolo, ecc. ecc.
 - È la relazione che corrisponde all'eredità
- **Parte-di (part-of, o has-a, o ha-un)**
 - Relazione fra un oggetto e le sue componenti
 - Il punto è una parte del cerchio (è il suo centro); il pistone è parte del motore, che è parte dell'automobile
 - Relazione che corrisponde all'operazione di composizione.
- **Usa**
 - Relazione fra le classi che implementano TDA e le classi che li usano: classi di prova.

Stefano Mizzaro - OO 25

Relazioni fra classi (2/2)

- **[Istanza-di (instance-of, membro-di, member-of)]**
 - Relazione fra una classe e i suoi elementi (istanze, oggetti, esemplari), o fra un insieme e i suoi elementi:
 - Il cerchio `c1` è un'istanza della classe `Cerchio`; la mia automobile è un'istanza della classe delle automobili; ecc.
- **Oppure (?)**:
 - `Cerchio` è-un `Punto` (cerchio = punto + raggio)
 - `Studente` parte-di `Persona` (in `toString()` di `studente` invoco `toString()` di `Persona` per visualizzare la "parte dello studente che è una persona")...
 - Buon senso, "regola del sottoinsieme", ...

Stefano Mizzaro - OO 26

Scaletta

- Esempio
- Ereditarietà
- Polimorfismo

Stefano Mizzaro - OO 27

Polimorfismo (1/2)

- Programma di grafica
- Struttura dati per tutte le figure
- Tanti array...
- Scomodo!

```

Punto[] punti;
Cerchio[] cerchi;
...
if (x instanceof Punto)
    punti[i] = x;
else if (x instanceof Cerchio)
    cerchi[i] = x;
else if ...
for (int i = ...) {
    punti[i].draw();
    cerchi[i].draw();
}
    
```

Stefano Mizzaro - OO 28

Polimorfismo (2/2)

- È più comodo parlare alla classe base!
- Si può fare!

```

Figura[] figure = new Figura[100];
figure[i] = new Punto();
figure[j] = new Cerchio();
for (int k = ...)
    figure[k].draw();
    
```

Stefano Mizzaro - OO 29

Perché il polimorfismo funziona

- N.B. Polimorfismo va combinato con (dipende da, è basato su):
 - eredità (`Figura` è sopraclasse) e
 - sovrascrittura (`draw()` è sovrascritto)
- Maniglie
 - Le variabili non contengono gli oggetti
 - Le variabili contengono il riferimento (la maniglia) agli oggetti

Stefano Mizzaro - OO 30

Le maniglie

`A a = new A();`

`A a = new A();`

Stefano Mizzaro - OO 31

Maniglie ed eredità

`A a = new A(); a.m();`

`B b = new B(); b.m();`

`A a = new B(); a = b; a.m();`

Stefano Mizzaro - OO 32

Terminologia

- Late binding, run-time binding
 - (durante l'esecuzione)
- Non early binding, compile-time binding
 - (durante la compilazione)
- Dynamic method lookup
- Sovrascrittura, sovraccarico
- Eckel: "If it isn't late binding, it isn't polymorphism"

Stefano Mizzaro - OO 33

Il codice Java

- 4 + 1 classi
 - Punto.java
 - Cerchio.java
 - Figura.java
 - (Linea.java)
 - UsaFigura.java

Stefano Mizzaro - OO 34

Punto.java, Cerchio.java

```
class Punto extends Figura {
    //tutto come prima
    public void draw() {
        ...
    }
}

class Cerchio extends Figura {
    //tutto come prima
    public void draw() {
        ...
    }
}
```

Stefano Mizzaro - OO 35

Figura.java

- Una figura generica non sa disegnarsi...

```
class Figura {
    public void draw() {}
}
```

Stefano Mizzaro - OO 36

UsaFigura.java

```
class UsaFigura {
    public static void main (String[] args) {
        Figura[] figure = new Figura[4];
        figure[0] = new Punto();
        figure[1] = new Cerchio();
        figure[2] = new Punto();
        figure[3] = new Cerchio();
        for (int i = ...)
            figure[i].draw(); // Il draw() "in basso"
    }
}
```

Stefano Mizzaro - OO

37

Esercizio: cosa visualizza?

```
class A {
    public void m(){
        System.out.println("A");
    }
}
class B extends A {
    public void m(){
        System.out.println("B");
    }
}
class P {
    public static void main (String[] args) {
        A a = new A();
        B b = new B();
        a.m();
        b.m();
        a = b;
        a.m();
    }
}
```

```
>java P
A
B
B
```

Stefano Mizzaro - OO

38

Riassumendo: cos'è la OOP?

- TDA
 - Incapsulamento, interfaccia, implementazione
 - Composizione, uso
- Scambio messaggi
 - Oggetti attivi che si "parlano"
 - Metodi/attributi d'istanza e di classe
- Ereditarietà
 - Estensione, sovrascrittura, sottotipo, **extends**, **super**, relazioni fra classi...
- Polimorfismo
 - Maniglie, late binding, "talk to the base class"

Stefano Mizzaro - OO

39

Riassunto

- Ereditarietà (cenni)
- Polimorfismo (cenni)
- Prossima lezione
 - Riassunto
 - Esercizi
 - Info su provetta
 - Info su autovalutazione
- Prossimamente
 - OO in Java

Stefano Mizzaro - OO

40