Dai TDA agli oggetti

Stefano Mizzaro

Dipartimento di matematica e informatica
Università di Udine
http://www.dimi.uniud.it/mizzaro
mizzaro@dimi.uniud.it

Programmazione, lezione 18
19 gennaio 2006

Prossime lezioni

- Aula (8 lez.)
  - 18/1, 19/1, 25/1, 26/1,
  - 1/2, 8/2, 15/2, 22/2
- Laboratorio (9 lez.)
  - 16/1,
  - 23/1, 25/1, 30/1, 1/2,
  - 6/2, 13/2, 20/2, 27/2
- (salvo contrattempi/variazioni)
- Ricevimento: Martedì 10:00 – 12:00

Riassunto

- "Mattoni"
- Sequenza, Selezione, Iterazione
- Array
- Sottoprogrammi (metodi)
- Ricorsione
- TDA

Riassunto TDA

- Astrazione sui tipi, non sulle istruzioni
  - "Aggiungo al Java i tipi (istruzioni) che non ha"
- Implementazione (attributi) + Operazioni (metodi)
- Dichiarazione, allocazione, invocazione
- `class`, `private`, `new`, notazione puntata
- Occultamento delle informazioni
  - Incapsulamento = unitarietà + inaccessibilità
  - >Comprensibilità, modificabilità, riusabilità

Oggi

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato

Interazione fra TDA

- Finora:
  - un unico TDA
  - una classe che lo usa
- In generale:
  - più TDA
  - che si usano a vicenda
  - una classe con il main

Esempio

- Programma di "grafica"
  - Ultrasemplificato
  - Solo punti e cerchi
  - Solo rappresentazione
  - Visualizzazione solo testuale
- Classe `Punto`: rappresenta punti
- Classe `Cerchio`: rappresenta cerchi
- Classe `Grafica`: con main

Punto.java

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


Cerchio.java

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

Grafica.java

```
/** Programma per la
 * prova di cerchi e punti */
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(Cerchio.toString(c1));
    Cerchio c2 = new Cerchio(p1,0);
    System.out.println(Cerchio.toString(c2));
    Cerchio c3 = new Cerchio(p2,1);
    System.out.println(Cerchio.toString(c3));
  }
}
```


Osservazioni

- Sovraccarico costruttori
  - Più comodo creare istanze
- Cosa succede quando viene invocato il costruttore di `Cerchio`?
  - Viene invocato anche il costruttore di `Punto`
- La relazione fra `Cerchio` e `Punto` è diversa da quella con `Grafica`:
  - Uso
  - Composizione (una parte di un cerchio è un punto)

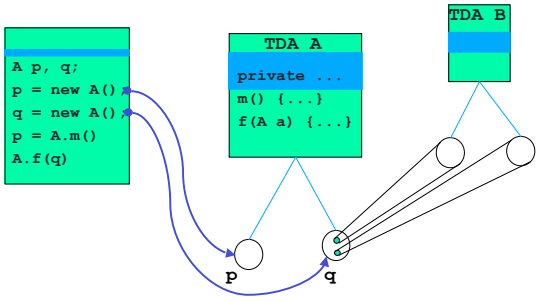
Programma con TDA (1/2)

```
class C {
  ... main ... {
    A p, q;
    p = new A();
    q = new A();
    A.m();
    p = A.f(q);
  }
}
```

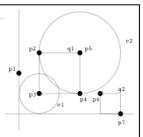
```
class A {
  ... private ...
  ... static ...
  ... void  m() {...}
  ... A f(A a) {...}
}
```

```
class C {
class B {
  ... private ...
  ... static ...
  ... void  m() {...}
  ... B f(B b) {...}
}
```

Programma con TDA (2/2)



Esercizi



- Modificare `Grafica.java`
- Aggiungere `Quadrato.java` (e modificare `Grafica.java`)
- Confrontare il codice sui lucidi con quello sul testo. Osservare come l'uso del `this` evita duplicazioni
- Cercare e invocare/collaudare versioni alternative dei costruttori
- Modificare i nomi dei parametri formali per
  - Renderli uguali agli attributi
  - Usare il `this`

Scaletta

- Interazione fra TDA
  - (Fine cap. 7)
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato

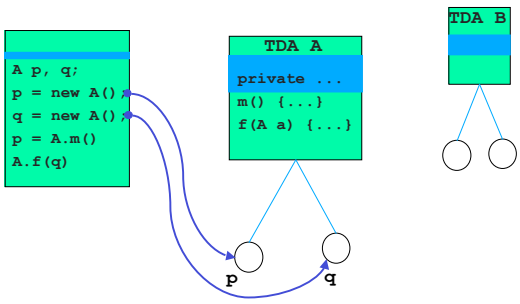
OO è...

- (O-O, Object Oriented, Orientato agli Oggetti)
- OO è:
  - TDA +
  - Scambio messaggi +
  - Eredità +
  - Polimorfismo

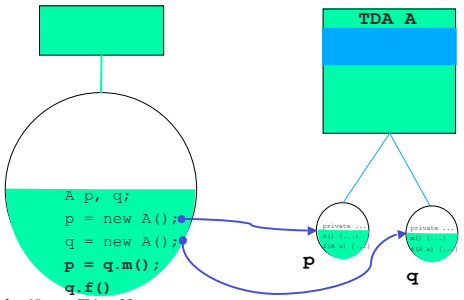
Scambio messaggi

- Da TDA e funzioni/procedure definite nel TDA a oggetti attivi che si scambiano messaggi
- Da "esegui il metodo **`getX()`** della classe **`Punto`** con argomento **`q`**":
  - `double a = Punto.getX(q);`
- a "manda il messaggio **`getX`** all'oggetto **`q`**":
  - `double a = q.getX();`
- "Approccio TDA":
  - variabili (passive) contengono stato e basta
  - i metodi sono nella classe
- "Approccio OO":
  - Oggetti/istanze (attive) contengono stato e metodi

Programma con TDA (di nuovo)



Programma con scambio messaggi



La classe `Punto` "a oggetti"

```
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 setX(double x) {this.x = x;}
  public void setY(double y) {this.y = y;}
  public double getX() {return x;}
  public double getY() {return y;}
}
```


Programma che usa `Punto`

```
class UsaPunto {
  public static void main(String[]args) {
    Punto p;
    p = new Punto(12.0, 34.9);
    Punto q = new Punto(0, 0);
    p = new Punto(2.3, 3.4);
    p.setX(2.5);
    double x = p.getX();
    p = q;  // Alias
  }
}
```


Confronto: TDA vs. OO

- Filosoficamente:
  - variabili passive vs. oggetti attivi
- In pratica: metodi d'istanza e un parametro in meno
  - `public static void setX(Punto p, double x)` VS. `public void setX(double x)`
  - `Punto.getX(q)` VS. `q.getX()`
- Non c'è più lo `static`
- Terminologia:
  - Attributi
  - Metodi

`Punto.java` (non a oggetti)

```
class Punto {
  private double x; private double y;
  ...
  static void set(Punto p, double x, double y){
    p.x = x; p.y = y;
  }
  static void setX(Punto p, double x) {
    p.x = x;
  }
  static void setY(Punto p, double y) {
    p.y = y;
  }
  static double getX(Punto p) {return p.x;}
  static double getY(Punto p) {return p.y;}
  ...
}
```


`UsaPunto.java` (non a oggetti)

```
/** Programma per la prova di cerchi e punti */
class UsaPunto {
  public static void main (String[] args) {
    Punto p1 = new Punto();
    Punto p2 = new Punto(1,1);
    Punto.setX(p1,5);
    System.out.println(Punto.getX(p2));
    ...
  }
}
```

Cosa fa il main

- Inizia l'esecuzione
- Crea oggetti
- Manda messaggi a oggetti (invoca metodi d'istanza)
- Invoca direttamente metodi **`static`**

Altro esempio

- Di nuovo la pila di interi limitata
- Però "a oggetti"

Pila.java

```
class Pila {
  private static final int MAX = 10;
  private int[] elementi;
  private int numElementi;
  Pila() {
    numElementi = 0;
    elementi = new int[MAX];
  }
  boolean vuota() {return numElementi == 0;}
  boolean piena() {return numElementi == MAX;}
  void push(int e) {
    if (!piena()) elementi[numElementi++] = e;
  }
  int top() {
    if (!vuota()) return elementi[numElementi-1];
    else return -1;
  }
  void pop() {if (!vuota()) numElementi--;}
}
```

UsaPila.java

```
class UsaPila {
  public static void main(String[] args) {
    Pila p = new Pila ();
    while (!p.piena()) {
      System.out.print("Inserisci un elemento:");
      p.push(Leggi.unInt());
    }
    while (!p.vuota()) {
      System.out.println(p.top());
      p.pop();
    }
  }
}
```


Confronto con la versione TDA

- Il TDA
  - Parametro in meno
  - Senza **`static`**
- Uso
  - Parametro in meno
  - Nome istanza anziché nome classe
- Filosoficamente: si parla alle istanze
- Si scrive di meno (meno parametri, non il nome della classe)

TDA vs. OO

- TDA: classi e variabili
  - Variabili = implementazione (com'è fatto)
  - Classi forniscono metodi per lavorare sulle variabili
  - I metodi sono di classe, si parla alle classi
- OO: classi e istanze
  - Istanza = implementazione + operazioni
  - Le operazioni sono a livello di ogni istanza
  - I metodi sono d'istanza, si parla alle istanze

Scaletta

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato

Il `this` rivisitato

- Ora si capisce: "this" = "questa istanza", "questo oggetto"
  - Il costruttore di questo oggetto
  - Gli attributi di questo oggetto
- Si potrebbe premettere sempre
  - a invocazioni di altri metodi dell'oggetto
  - all'accesso agli attributi dell'oggetto

`Pila.java` col `this`...

```
class Pila {
  ...
  PilaInteriLimitata() {
    this.numElementi = 0;
    this.elementi = new int[MAX];
  }
  boolean vuota() {return this.numElementi == 0;}
  boolean piena() {return this.numElementi==MAX;}
  void push(int e) {
    if (!this.piena())
      this.elementi[this.numElementi++] = e;
  }
  int top() {
    if (!this.vuota()) return
      this.elementi[this.numElementi-1];
    else return -1;
  }
  void pop() {
    if (!this.vuota()) numElementi--;}
}
```

Il `toString` rivisitato

- Avevamo visto che se un TDA C ha il metodo `toString`, per visualizzare si fa:
  - `System.out.println(C.toString(x))`
- Ma ora sappiamo che `toString` può/deve essere d'istanza, non `static` => Lo si invocherà così:
  - `System.out.prinltn(x.toString())`
  - È più comodo!
- Di più: l'invocazione è automatica!!
  - `System.out.prinltn(x)`
- Se una classe non ha il `toString` ce n'è uno di default, ma non molto utile...
- Ex.: aggiungere il `toString` alle classi viste

Riassunto

- Interazione fra TDA
- Da TDA a OO: lo scambio messaggi
  - Concetto ed Esempi
- Il `this` rivisitato
- Il `toString` rivisitato

Prossima lezione

- Altri 2 ingredienti dell'OO:
  - Ereditarietà
  - Polimorfismo
- (Poi, OO in Java:
  - Varianti
  - Librerie (API)
-)