

## I Tipi di Dato Astratti (TDA) - 2

Stefano Mizzaro

Dipartimento di matematica e informatica  
Università di Udine  
<http://www.dimi.uniud.it/mizzaro/>  
mizzaro@dimi.uniud.it  
Programmazione, lezione 14  
17 novembre 2006

## Oggi

- Riassunto
  - Dalla programmazione strutturata ai TDA (introduzione; evoluzione della programmazione strutturata; occultamento delle informazioni)
- Fine TDA
  - Costruttore
  - `this`
  - Metodi/costruttori sovraccarichi (overloaded)
  - `toString`
  - Procedurale vs. funzionale
  - Esempi

Stefano Mizzaro - TDA 2

2

## TDA: l'idea

- Tipo di Dato Astratto (TDA)
- Abstract Data Type (ADT)
- L'idea è:
  - Programma che gestisce numeri complessi => aggiungo il tipo di dato astratto `NumeroComplesso`
  - Programma dell'anagrafe => aggiungo i TDA `Persona`, `Indirizzo`, `Data`, ...
  - ...
- Non "aggiungo al Java i metodi che non ha" ma "aggiungo al Java i tipi che non ha"
  - E poi li uso astruendo dall'implementazione
  - È un meccanismo di astrazione

Stefano Mizzaro - TDA 2

3

## TDA: i concetti (1/2)

- Astrazione: nuovo tipo (non nuova istruzione)
- Definire
  - Come sono fatti (attributi): implementazione
  - Operazioni (metodi, per ora `static`): interfaccia
  - `class`
- Usare
  - Dichiarazione
  - E allocazione, sullo heap
  - `new` (ricorda qualcosa?)
  - Notazione puntata (ricorda qualcosa?)
    - Non solo per accedere alle componenti
    - Anche per invocare i metodi

Stefano Mizzaro - TDA 2

4

## TDA: i concetti (2/2)

- Occultamento delle informazioni (dell'implementazione)
  - Information (implementation) hiding
  - Cambio implementazione => cambio uso? NO!!
  - `private`
  - Operazioni di accesso con metodi
  - Astratto = si astrae dall'implementazione quando si usa
  - Limitazione alla libertà del programmatore? Sì, ma positiva!
    - Divide et impera (anche con i metodi)
    - Riutilizzo (i TDA creati per un programma spesso funzionano anche per altri programmi; i metodi no)

Stefano Mizzaro - TDA 2

5

## Esempio: definizione e uso

```
class Ora {
    private int secondi;
    static void impostaOra(Ora x, int o, int m, int s) {
        x.secondi = o*3600+m*60+s;
    }
    static int getOre(Ora x) {return x.secondi/3600;}
    static int getMinuti(Ora x) {
        return (x.secondi%3600)/60;
    }
    static int getSecondi(Ora x) {return x.secondi%60;}
}

class UsaOra {
    public static void main(String[] args) {
        ...
        Ora h = new Ora();
        Ora.impostaOra(h,12,22,31);
        System.out.println("Sono le ore " +Ora.getOre(h));
        ...
    }
}
```

## Terminologia

- Tipi
- TDA
- Classi
- Variabili
- (Valori, Istanze, Oggetti)
- Attributi
- Incapsulamento = unitarietà + inaccessibilità
  - Unitarietà:
    - "raggruppamento"
    - definizione in un unico punto
  - Inaccessibilità: dell'implementazione

Stefano Mizzaro - TDA 2

7

## Scaletta

- Costruttore
- `this`
- Metodi/costruttori sovraccarichi (overloaded)
- `toString`
- Procedurale vs. funzionale
- Esempi

Stefano Mizzaro - TDA 2

8

## Il costruttore

- Le classi hanno un metodo particolare, il costruttore, che viene invocato quando si istanzia un TDA
  - Essendo il primo metodo della classe che viene eseguito, di solito viene usato per inizializzare la variabile creata
- Il costruttore è un metodo "strano":
  - ha (deve avere) lo stesso nome della classe e
  - non ha un tipo restituito (nemmeno void)

Stefano Mizzaro - TDA 2

9

## Esempio

```
class Coordinate {
    private int x, y;
    //il costruttore ha lo stesso nome della classe
    Coordinate(int a, int b) {
        x = a;
        y = b;
    }
}
```

- Il costruttore viene invocato (con la **new**) per istanziare un TDA:

```
Coordinate p = new Coordinate(2,6);
```

Stefano Mizzaro - TDA 2

10

## Costruttori

- Si possono definire più costruttori per una singola classe, a patto che abbiano tipo e/o numero di parametri diversi
- Costruttore (metodo) sovraccarico
- Esempio:

```
/* Crea un nuovo oggetto con coordinate diverse */
Coordinate(int a, int b) {
    x = a; y = b;
}
/* Crea un oggetto con le coordinate uguali */
Coordinate(int a) {
    x = a; y = a;
}
```

Stefano Mizzaro - TDA 2

11

## Esempio: il TDA Ora

```
class Ora {
    private int ore, minuti, secondi;
    Ora(int o, int m, int s) {
        ore = o; minuti = m; secondi = s;
    }
    static void setOra(Ora x, int o, int m, int s) {
        x.ore = o; x.minuti = m; x.secondi = s;
    }
    static void setOra(Ora x, int o, int m) {
        setOra(x,o,m,0);
    }
    static int getOre(Ora x) {return x.ore;}
    static int getMinuti(Ora x) {return x.minuti;}
    static int getSecondi(Ora x) {return x.secondi;}
}
```

Stefano Mizzaro - TDA 2

12

## Esempio: uso del TDA ora

```
class UsaOra {
    public static void main(String[] args) {
        Ora x = new Ora(12,32,05);
        Ora y = new Ora(Ora.getOre(x), 20, 3);
        System.out.println(Ora.getMinuti(y));
        Ora.setOra(y, 20, 4);
        System.out.println(Ora.getOre(y) + ":" +
            Ora.getMinuti(y) + ":" +
            Ora.getSecondi(y));
    }
}
```

- Qs. è un caso banale, immaginarsi in generale..

Stefano Mizzaro - TDA 2

13

## Esercizi

- Che cosa fa il seguente costruttore?
 

```
Ora(int o) {
    ore = o; minuti = 0; secondi = 0;
}
```
- E questo?
 

```
Ora(Ora o) {
    ore = o.ore;
    minuti = o.minuti;
    secondi = o.secondi;
}
```

Stefano Mizzaro - TDA 2

14

## Costruttori

- Un costruttore può avere parametri di ogni tipo, anche della sua stessa classe
- Può invocare altri metodi
  - Anche della stessa classe, ma attenzione!
- Quando si deve creare un'istanza di una certa classe, si deve invocare uno dei suoi costruttori
- Se non è stato specificato nessun costruttore
  - allora si può usare quello senza parametri (che c'è "automaticamente"),
- altrimenti
  - si deve usare uno di quelli definiti

Stefano Mizzaro - TDA 2

15

## Esempi

- Se la classe è:
 

```
class Coordinate {
    private int x, y;
}
```
- si deve scrivere
 

```
Coordinate p =
    new Coordinate();
```
- Se invece è:
 

```
class Coordinate {
    private int x, y;
    Coordinate(int a, int b) {
        x = a;
        y = b;
    }
}
```
- si **deve** scrivere
 

```
Coordinate p =
    new Coordinate(2,3);
```

Stefano Mizzaro - TDA 2

16

## Ancora esempi

- Se, infine, la classe ha entrambi i costruttori:

```
class Coordinate {
    private int x, y;
    Coordinate() {x = 0; y = 0;}
    Coordinate(int a, int b) {x = a; y = b;}
}
```

- si possono scrivere entrambe le istruzioni:

```
Coordinate p = new Coordinate();
Coordinate q = new Coordinate(0,0);
```

Stefano Mizzaro - TDA 2

17

## Esempio Freccia

```
class FrecciaOrizzontale {
    private int lunghezza;
    private char verso;
    /* Questo costruttore permette di creare solo frecce che
    vanno a sinistra */
    FrecciaOrizzontale(int l) {
        lunghezza = l;
        verso = 's';
    }
    static void cambiaVerso(FrecciaOrizzontale f) {
        if (f.verso == 's')
            f.verso = 'd';
        else f.verso = 's';
    }
}
```

Stefano Mizzaro - TDA 2

18

## Scaletta

- Costruttore
- **this**
- Metodi/costruttori sovraccarichi (overloaded)
- toString
- Procedurale vs. funzionale
- Esempi

Stefano Mizzaro - TDA 2

19

## Visibilità

- Nei costruttori precedenti ho dovuto inventare nomi di parametri che fossero diversi da quelli delle variabili della classe
- Cosa succede se usiamo gli stessi nomi?

```
class Coordinate {
    private int x, y;
    Coordinate(int x, int y) {
        x = x;
        y = y;
    }
}
```

Stefano

20

## Il this

- Regola di visibilità
  - Se nomi dei parametri = nomi degli attributi
  - Gli attributi vengono nascosti
  - (del costruttore, o di qualsiasi altro metodo)
- Per accedervi occorre specificare che sono le variabili di "questo oggetto", con la parola riservata **this**:
 

```
Coordinate(int x, int y){
    this.x=x; this.y=y;
}
```

  - «assegno alla variabile **x** di questo oggetto il valore del parametro **x**, assegno alla variabile **y** di questo oggetto il valore del parametro **y**»
  - ```
Coordinate(int x, int y) {x = x; y = y;}
```
  - «assegno al parametro **x** il valore del parametro **x**... e quindi non modifico la variabile **x** della classe!

Stefano Mizzaro - TDA 2

21

## Riassunto visibilità

- Variabili locali a un blocco
  - ```
for (int i = ...)
```
- Variabili locali a un metodo
  - ```
static void m(int x){
    int y...
```
- Variabili locali a una classe
  - ```
class C {
    int x...
```
  - **public** (o niente): visibili anche all'esterno della classe
  - **private**: solo all'interno della classe

Stefano Mizzaro - TDA 2

22

## Il this

- **this** può essere usato anche per riferirsi ad un altro costruttore (sovraccarico). Esempio:

```
class Coordinate {
    private int x, y;
    Coordinate(int x, int y) {
        this.x = x; this.y = y;
    }
    /** costruttore che crea un oggetto con
     * coordinate uguali */
    Coordinate(int a) {this(a,a);}
}
```

← questo costruttore

Stefano Mizzaro - TDA 2

23

## Costruttori e metodi sovraccarichi

- A cosa servono?
- Semplificano uso e allocazione delle variabili del TDA
- Evitare duplicazione del codice
  - Con il **this**, o invocando altri metodi
- Può essere comodo, ma non esagerare...

```
Coordinate c = new Coordinate();
Coordinate c = new Coordinate(0);
Coordinate c = new Coordinate(0,0);
```

Stefano Mizzaro - TDA 2

24

## Scaletta

- Costruttore
- **this**
- Metodi/costruttori sovraccarichi (overloaded)
- **toString**
- Procedurale vs. funzionale
- Esempi

Stefano Mizzaro - TDA 2

25

## Il metodo toString

- Chi deve avere la conoscenza/responsabilità di sapere come visualizzare il valore di un TDA?
  - La classe! (cfr. *Ora* e *UsaOra*: scomodo!)
- Si potrebbe definire un metodo nel TDA
  - `static void print(...)`
- ... ma si preferisce definire (vedremo perché)
  - `static String toString(...)`
  - che restituisce una rappresentazione visualizzabile dell'istanza (Vedremo l'invocazione automatica di `toString`)
- `toString` verrà invocato nell'istruzione `System.out.print` o `println`
- In questo modo è possibile specificare come verranno visualizzati gli oggetti di una certa classe

Stefano Mizzaro - TDA 2

26

## toString

- `toString` restituisce la rappresentazione dell'oggetto in forma di stringa. Ad es. :

```
class Coordinate {
    private int x, y;
    ...
    static String toString(Coordinate c){
        return "(" + c.x + "," + c.y + " ";
    }
}
```

```
class UsaCoordinate {
    Coordinate a = new ...;
    ...
    System.out.println(Coordinate.toString(a));
}
```

27

## Ora.java

```
class Ora {
    private int ore, minuti, secondi;
    Ora(int ore, int minuti, int secondi) {
        this.ore=ore; this.minuti=minuti; this.secondi=secondi;
    }
    static void impostaOra(Ora x, int ore, int minuti,
        int secondi) {
        x.ore = ore;
        x.minuti = minuti;
        x.secondi = secondi;
    }
    static int getOre(Ora x) {return x.ore;}
    static int getMinuti(Ora x) {return x.minuti;}
    static int getSecondi(Ora x) {return x.secondi;}
    static String toString(Ora x) {
        return x.ore+" "+x.minuti+" "+x.secondi;
    }
}
```

Stefano Mizzaro - TDA 2

28

## UsaOra.java

```
class UsaOra {
    public static void main(String[] args) {
        Ora x = new Ora(10,2,3);
        System.out.println(Ora.toString(x));
        Ora.impostaOra(x,12,2,0);
        System.out.println(Ora.getMinuti(x));
    }
}
```

Stefano Mizzaro - TDA 2

29

## Scaletta

- Costruttore
- **this**
- Metodi/costruttori sovraccarichi (overloaded)
- **toString**
- Procedurale vs. funzionale
- Esempi

Stefano Mizzaro - TDA 2

30

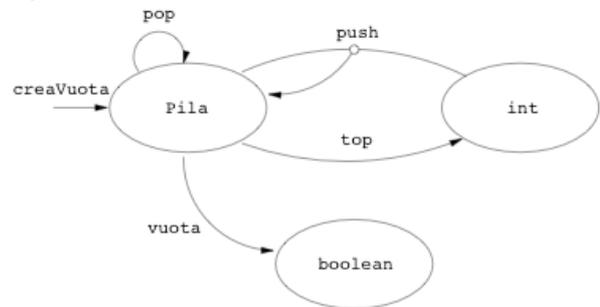
## Pila (di interi limitata)

- Pila (stack): struttura dati a gestione LIFO (Last In First Out, il primo che entra è l'ultimo che esce)
- Operazioni:
  - push: aggiungi un elemento in cima
  - top: restituisce l'elemento in cima
  - pop: rimuove un elemento dalla cima
  - piena: dice se c'è ancora posto
  - vuota: dice se è vuota
- Vediamo una versione semplice: la pila limitata di interi
  - C'è un numero massimo di elementi
  - Quindi potremo implementarla con un array

Stefano Mizzaro - TDA 2

31

## Segnatura del TDA Pila



Stefano Mizzaro - TDA 2

32

## Pila.java (1/2)

```

/**Il TDA pila di interi */
class Pila {
    /** Il numero di elementi massimo */
    private static final int MAXELEMENTI = 10;
    /** Il vettore che contiene gli elementi*/
    private int[] elementi;
    /** Il numero di elementi effettivamente presenti */
    private int numElementi;
    /** Costruttore: costruisce (e restituisce
     * implicitamente) una pila vuota */
    public Pila() {
        numElementi = 0;
        elementi = new int[MAXELEMENTI];
    }
    /** Dice se la pila p e' vuota */
    public static boolean vuota (Pila p) {
        return (p.numElementi == 0);
    }
    /** Dice se la pila p e' piena */
    public static boolean piena (Pila p) {
        return (p.numElementi == p.MAXELEMENTI);
    }
}

```

## Pila.java (2/2)

```

/** push dell'elemento e sulla pila p */
public static Pila push (Pila p, int e) {
    if (!piena(p))
        p.elementi[p.numElementi++] = e;
    return p;
}

/** Restituisce il top della pila p */
public static int top(Pila p) {
    if (!vuota(p))
        return p.elementi[p.numElementi - 1];
    else return -1;
}

/** Elimina un elemento dalla cima della pila p */
public static Pila pop(Pila p) {
    if (!vuota(p))
        p.numElementi--;
    return p;
}
}

```

Stefano Mizzaro - TDA 2

34

## ProvaPila.java

```

/** Programma per la prova della classe Pila. */
class ProvaPila {
    public static void main (String[] args) {
        Pila p1 = new Pila();
        p1 = Pila.push(p1,1);
        p1 = Pila.push(p1,2);
        p1 = Pila.push(p1,3);
        p1 = Pila.push(p1,4);
        System.out.println(Pila.top(p1));
        p1 = Pila.pop(p1);
        p1 = Pila.pop(p1);
        System.out.println(Pila.top(p1));
        p1 = Pila.push(p1,5);
        System.out.println(Pila.top(p1));
    }
}

```

Stefano Mizzaro - TDA 2

35

## public o non public...

- Per ora non fa differenza
- Se non c'è il **private** attributi e metodi sono visibili dal resto del programma al di fuori della classe
  - (la differenza la vedremo più avanti: con il **public** sono "più visibili")
- L'unico che è necessario è quello del main
  - Esercizio: dato un programma funzionante a vostra scelta, provate a togliere il **public**, a compilare e ad eseguire
- Il **private** è importante

Stefano Mizzaro - TDA 2

36

## Funzionale e procedurale

- Nella versione precedente i metodi sono funzioni
- Alcuni potrebbero essere procedure
  - push, pop
- Lavorano per effetto collaterale (side effect) sui parametri
  - (Parametri che non sono tipi primitivi: viene passato il riferimento per valore => modifiche al parametro attuale si ripercuotono sul parametro formale)
  - Esercizio: rivedere la versione funzionale ed eliminare i side effect

Stefano Mizzaro - TDA 2

37

## PilaP.java (procedurale)

```
class PilaP {
    ...
    /** push dell'elemento e sulla pila p */
    static void push (PilaP p, int e) {
        if (!piena(p))
            p.elementi[p.numElementi++] = e;
    }

    /** Elimina un elemento dalla cima della pila
     * p */
    static void pop(PilaP p) {
        if(!vuota(p))
            p.numElementi--;
    }
}
```

Stefano Mizzaro - TDA 2

38

## ProvaPilaP.java (procedurale)

```
/** Programma per la prova della classe PilaP. */
class ProvaPilaP {
    public static void main (String[] args) {
        PilaP p1 = new PilaP();
        PilaP.push(p1,1);
        PilaP.push(p1,2);
        PilaP.push(p1,3);
        PilaP.push(p1,4);
        System.out.println(PilaP.top(p1));
        PilaP.pop(p1);
        PilaP.pop(p1);
        System.out.println(PilaP.top(p1));
        PilaP.push(p1,5);
        System.out.println(PilaP.top(p1));
    }
}
```

## Funzionale – procedurale (1/2)

- Definizione metodo

```
public static Pila push (Pila p, int e) {
    if (!piena(p))
        p.elementi[p.numElementi++] = e;
    return p;
}
```

```
public static void push (PilaP p, int e) {
    if (!piena(p))
        p.elementi[p.numElementi++] = e;
}
```

Stefano Mizzaro - TDA 2

40

## Funzionale – procedurale (2/2)

- Invocazione metodo

```
class ProvaPila {
    public static void main (String[] args) {
        Pila p1 = new Pila();
        p1 = Pila.push(p1,1);
        ...
        p1 = Pila.push(Pila.push(p1,1),2);
        p1 = Pila.pop(p1);
    }
}
```

```
class ProvaPilaP {
    public static void main (String[] args) {
        PilaP p1 = new PilaP();
        PilaP.push(p1,1);
        ...
        PilaP.pop(p1);
    }
}
```

Stefano Mizzaro - TDA 2

41

## Riassunto

- Riassunto
  - Programmazione strutturata
  - TDA (Tipo di dato astratto)
    - Altro meccanismo di astrazione
    - Evoluzione della programmazione strutturata (class, new, notazione puntata)
    - Occultamento delle informazioni (private, metodi come operazioni)
    - Costruttori
    - this
    - toString
    - Procedurale vs. funzionale
    - Sovraccarico
    - Esempi
- Prossima lezione:
  - Interazione fra TDA
  - TDA -> OO (Object Oriented, Orientato agli oggetti)

Stefano Mizzaro - TDA 2

42