



ASTRAZIONI SUI DATI : IMPLEMENTAZIONE DI TIPI DI DATO ASTRATTI IN JAVA

1



Abstract Data Types

- ✦ Insieme di **valori**
- ✦ Un insieme di **operazioni** che possono essere applicate in modo uniforme ai valori
- ✦ **NON** e' caratterizzato dalle rappresentazione dei dati (abstraction barrier)
 - La rappresentazione dei dati e' **privata** e **modificabile** senza effetto sul codice che utilizza il tipo di dato

2

Specificare un ADT



- ✎ La specifica di un ADT e' un contratto che definisce
 - **valori, operazioni** in termini di nome, parametri tipo, effetto osservabile
- ✎ *Separation of concerns*:
 - Progettazione e realizzazione del ADT
 - Progettazione applicazione che utilizza ADT

3

Formato della specifica



```
public class NuovoTipo {  
    // OVERVIEW: Gli oggetti di tipo NuovoTipo  
    // sono collezioni modificabili di ..  
  
    // costruttori  
    public NuovoTipo ()  
        // EFFECTS: ...  
  
    // metodi  
    // specifiche degli altri metodi  
}
```

4

IntSet



```
public class IntSet {
    // OVERVIEW: un IntSet è un insieme modificabile
    // di interi di dimensione qualunque
    // costruttore
    public IntSet ()
        // EFFECTS: inizializza this all'insieme vuoto
    // metodi
    public void insert (int x)
        // EFFECTS: aggiunge x a this
    public void remove (int x)
        // EFFECTS: toglie x da this
    public boolean isIn (int x)
        // EFFECTS: se x appartiene a this ritorna
        // true, altrimenti false
    ...}
```

5

IntSet



```
public class IntSet {
    ...
    // metodi
    ...
    public int size ()
        // EFFECTS: ritorna la cardinalità di this
    public int choose () throws EmptyException
        // EFFECTS: se this è vuoto, solleva
        // EmptyException, altrimenti ritorna un
        // elemento qualunque contenuto in this
}
```

6

Esempi di uso



```
myIntSet = new IntSet();  
:  
If myIntSet.IsIn(50) the system.out.println( ...)  
//Uso corretto  
:  
  
myIntSet = 50;  
//Uso scorretto
```

7

Astrazioni sui dati: implementazione



- ✚ scelta fondamentale è quella della rappresentazione (rep)
 - come i valori del tipo astratto sono implementati in termini di altri tipi
 - ✓ tipi primitivi o già implementati
 - ✓ nuovi tipi astratti che facilitano l'implementazione del nostro
 - metodologia: iterazione del processo di decomposizione basato su astrazioni
 - la scelta deve tener nel dovuto conto la possibilità di implementare in modo efficiente i costruttori e gli altri metodi
- ✚ poi viene l'implementazione dei costruttori e dei metodi

8

La rappresentazione



- ✦ i linguaggi che permettono la definizione di tipi di dato astratti hanno meccanismi molto diversi tra loro per definire come
 - i valori del nuovo tipo sono implementati in termini di valori di altri tipi
- ✦ in Java, gli oggetti del nuovo tipo sono semplicemente collezioni di valori di altri tipi
 - definite (nella implementazione della classe) da un insieme di variabili di istanza private
 - ✓ accessibili solo dai costruttori e dai metodi della classe

9

Definire un tipo in Java



- ✦ un insieme di variabili di istanza
 - **Private**: devono essere accessibili solo dai costruttori e dai metodi della classe
- ✦ i valori espliciti che si vedono sono solo quelli costruiti dai costruttori
 - più o meno i casi base di una definizione ricorsiva
- ✦ gli altri valori sono eventualmente calcolati dai metodi
 - rimane nascosta l'eventuale struttura ricorsiva

10

Usi “corretti” delle classi in Java



- ✎ nella definizione di astrazioni sui dati
 - le classi contengono essenzialmente metodi di istanza e variabili di istanza private
 - ✓ eventuali variabili statiche possono servire (ma è sporco!) per avere informazione condivisa fra oggetti diversi
 - ✓ eventuali metodi statici non possono comunque vedere l'istanza dell'oggetto e servono solo a manipolare le variabili statiche

11

I tipi record in Java



- ✎ Java non ha un meccanismo primitivo per definire tipi record (le struct di C)
 - ma è facilissimo definirli
 - anche se con una deviazione dai discorsi metodologici che abbiamo fatto
 - ✓ la rappresentazione non è nascosta (non c'è astrazione!)
 - ✓ non ci sono metodi
 - ✓ di fatto non c'è specifica separata dall'implementazione

12

Un tipo record



```
class Pair {
  // OVERVIEW: un tipo record
  int coeff;
  int exp;
  // costruttore
  Pair (int c, int n)
    // EFFECTS: inizializza il "record" con i
    // valori di c ed n
    { coeff = c; exp = n;}
}
```

- ✎ la rappresentazione non è nascosta
 - dopo aver creato un'istanza si accedono direttamente i "campi del record"
- ✎ la visibilità della classe e del costruttore è ristretta al package in cui figura
- ✎ non ci sono metodi diversi dal costruttore

13

Implementazione di IntSet



```
public class IntSet {
  // OVERVIEW: un IntSet è un insieme modificabile
  // di interi di dimensione qualunque
  private Vector els; // la rappresentazione
  // costruttore
  public IntSet ()
    // EFFECTS: inizializza this all'insieme vuoto
    {els = new Vector();}
  ...}
}
```

- ✎ un insieme di interi è rappresentato da un **Vector**
 - più adatto dell'Array, perché l'insieme ha dimensione variabile
- ✎ gli elementi di un Vector sono di tipo **Object**
 - non possiamo memorizzarci valori di tipo **int**
 - usiamo oggetti di tipo **Integer**
 - ✓interi visti come oggetti

14

Implementazione di IntSet



```
public void insert (int x)
    // EFFECTS: aggiunge x a this
    {Integer y = new Integer(x);
    if (getIndex(y) < 0) els.add(y); }
private int getIndex (Integer x)
    // EFFECTS: se x occorre in this ritorna la
    // posizione in cui si trova, altrimenti -1
    {for (int i = 0; i < els.size(); i++)
    if (x.equals(els.get(i))) return i;
    return -1; }
```

- ✦ non abbiamo occorrenze multiple di elementi
 - si semplifica l'implementazione di remove
- ✦ il metodo privato ausiliario getIndex ritorna un valore speciale e non solleva eccezioni
 - va bene perché è privato
- ✦ notare l'uso del metodo equals su Integer

15

Implementazione di IntSet



```
public void remove (int x)
    // EFFECTS: toglie x da this
    {int i = getIndex(new Integer(x));
    if (i < 0) return;
    els.set(i, els.lastElement());
    els.remove(els.size() - 1);}
public boolean isIn (int x)
    // EFFECTS: se x appartiene a this ritorna
    // true, altrimenti false
    { return getIndex(new Integer(x)) >= 0; }
```

- ✦ nella rimozione, se l'elemento è presente, ci scrivo sopra l'ultimo corrente ed elimino l'ultimo elemento

16

Implementazione di IntSet



```
public int size ()
// EFFECTS: ritorna la cardinalità di this
{return els.size(); }
public int choose () throws EmptyException
// EFFECTS: se this è vuoto, solleva
// EmptyException, altrimenti ritorna un
// elemento qualunque contenuto in this
{if (els.size() == 0) throw
    new EmptyException("IntSet.choose");
return
    ((Integer) els.lastElement()).intValue(); }
```

☛ anche se lastElement potesse sollevare un'eccezione, qui non può succedere. Perché?

17

I polinomi



```
public class Poly {
// OVERVIEW: un Poly è un polinomio a
// coefficienti interi non modificabile
// esempio:  $c_0 + c_1*x + c_2*x^2 + \dots$ 

// costruttori
public Poly ()
// EFFECTS: inizializza this al polinomio 0
public Poly (int c, int n) throws
    NegativeExponentExc
// EFFECTS: se  $n < 0$  solleva
    NegativeExponentExc
// altrimenti inizializza this al polinomio  $cx^n$ 

// metodi
...}
```

18

I polinomi



```
public class Poly {
    ...
    // metodi
    public int degree ()
        // EFFECTS: ritorna 0 se this è il polinomio
        // 0, altrimenti il più grande esponente con
        // coefficiente diverso da 0 in this
    public int coeff (int d)
        // EFFECTS: ritorna il coefficiente del
        // termine in this che ha come esponente d
    public Poly add (Poly q) throws
        NullPointerException
        // EFFECTS: q=null solleva
        NullPointerException
        // altrimenti ritorna this + q
    ...}

```

19

I polinomi 3



```
public class Poly {
    ...
    // metodi
    ...
    public Poly mul (Poly q) throws
        NullPointerException
        // EFFECTS: q=null solleva
        NullPointerException
        // altrimenti ritorna this * q
    public Poly sub (Poly q) throws
        NullPointerException
        // EFFECTS: q=null solleva
        NullPointerException
        // altrimenti ritorna this - q
    public Poly minus ()
        // EFFECTS: ritorna -this
    }

```

20

Prima implementazione di Poly

```
public class Poly {
    // OVERVIEW: un Poly è un polinomio a
    // coefficienti interi non modificabile
    // esempio:  $c_0 + c_1*x + c_2*x^2 + \dots$ 
    private int[] termini; // la rappresentazione
    private int deg; // la rappresentazione
```

- ✎ i polinomi non cambiano la dimensione
 - Array invece che Vector
 - l'elemento in posizione i contiene il coefficiente del termine che ha esponente i
 - va bene solo per polinomi non sparsi
- ✎ per comodità (efficienza) ci teniamo traccia nella rappresentazione del degree del polinomio
 - variabile di tipo int

21

Prima implementazione di Poly

```
// costruttori
public Poly ()
    // EFFECTS: inizializza this al polinomio 0
    {termini = new int[1]; deg = 0; }
public Poly (int c, int n) throws
    NegativeExponentExc
    // EFFECTS: se  $n < 0$  solleva NegativeExponentExc
    // altrimenti inizializza this al polinomio  $cx^n$ 
    if ( $n < 0$ ) throw new NegativeExponentExc ("Poly(int,int)
    constructor");
    if ( $c == 0$ )
        {termini = new int[1]; deg = 0; return; }
    termini = new int[n+1];
    for (int i = 0; i < n; i++) termini[i] = 0;
    termini[n] = c; deg = n; }
private Poly (int n)
    {termini = new int[n+1]; deg = n; }
```

- ✎ il polinomio vuoto è rappresentato da un array di un elemento contenente 0
- ✎ un costruttore privato di comodo

22

Prima implementazione di Poly

```
public int degree ()
// EFFECTS: ritorna 0 se this è il polinomio
// 0, altrimenti il più grande esponente con
// coefficiente diverso da 0 in this
{return deg; }
public int coeff (int d)
// EFFECTS: ritorna il coefficiente del
// termine in this che ha come esponente d
{if (d < 0 || d > deg) return 0;
 else return termini[d];}
public Poly minus ()
// EFFECTS: ritorna -this
{Poly y = new Poly(deg);
 for (int i = 0; i < deg; i++)
 y.termini[i] = - termini[i];
 return y;}
public Poly sub (Poly q) throws
    NullPointerException
// EFFECTS: q=null solleva NullPointerException
// altrimenti ritorna this - q
{return add(q.minus()); }
```

23

Prima implementazione di Poly

- le implementazioni di add e mul sono più complesse
 - ma solo negli aspetti algoritmici che non mostriamo
- se i polinomi sono sparsi, questa implementazione non è efficiente
 - arrays grandi e pieni di 0
 - un'implementazione alternativa in termini di Vector i cui elementi sono coppie (coefficiente, esponente)
 - ✓ esattamente il record type che abbiamo visto

```
class Pair {
int coeff; int exp;
Pair (int c, int n)
{ coeff = c; exp = n;}}
```

24

Seconda implementazione di Poly

```
public class Poly {  
  // OVERVIEW: un Poly è un polinomio a  
  // coefficienti interi non modificabile  
  // esempio:  $c_0 + c_1*x + c_2*x^2 + \dots$   
  private Vector termini; // la rappresentazione  
  private int deg; // la rappresentazione
```

☛ gli oggetti contenuti in termini sono Pair che rappresentano i termini con coefficiente diverso da 0

☛ un esempio di operazione

```
public int coeff (int d)  
  // EFFECTS: ritorna il coefficiente del  
  // termine in this che ha come esponente d  
  {for (int i = 0; i < termini.size(); i++)  
    {Pair p = (Pair) termini.get(i);  
      if (p.exp == d) return p.coeff;}  
  return 0;}
```

- notare il casting