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

Forme di astrazione

- ***Procedural abstraction***: separazione delle proprieta' logiche di una azione computazionale dai dettagli implementativi
  - Programmazione I + Logica per la programmazione
- ***Data abstraction***: separazione delle proprieta' logiche dei dati dei dettagli della rappresentazione dei dati.

Abstract data types

- abstract data type (ADT) una collezione di elementi il cui comportamento logico e' definito da un dominio di valori e dalle operazioni su quel dominio.
- ADT
  - Nome
  - Valori
  - Operazioni
  - Semantica delle operazioni

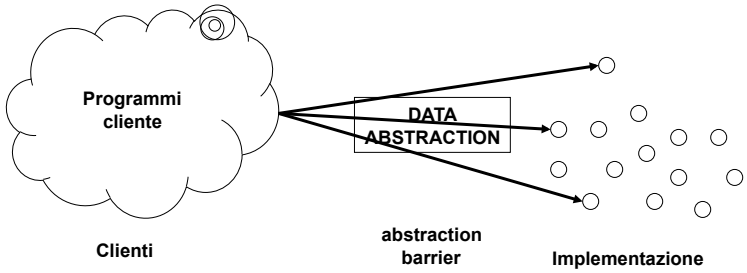
3

Benefici

I clienti dell'astrazione devono "rispettare" o per meglio dire "sono costretti a rispettare" dell'astrazione …

Questa cosa viene comunemente indicata con il nome di *"abstraction barrier"*

- Una caratteristica importante da avere nei linguaggi



4

Esempio (ADT via Assiomi)

- Nome: *Stack (of Item)*
- *Item* (parametro) il tipo degli elementi che possono essere inserito nello stack

5

ADT

- Operazioni (Segnatura)
  - ***Constructors**:*
    - ✓*create: unit -> Stack*
  - ***Mutators**:*
    - ✓*push: (Stack, Item) -> Stack*
    - ✓*pop: Stack -> Stack*
  - ***Accessors**:*
    - ✓*top: Stack -> Item*
    - ✓*empty: Stack -> boolean*
    - ✓*Size: Stack -> int*
  - ***Destructors**:*
    - ✓*destroy: Stack -> unit*

6

ADT STACK: assiomi

Assiomi: regolano il comportamento delle operazioni (semantica intesa della astrazione)

top(push(s,x)) = x
pop(push(s,x)) = s
empty(create()) = True
empty(push(s,x)) = False

Notazione in stile funzionale

7

Stack: assiomi

S.size(), S.empty(), S.push(t) sono sempre definite (operazioni totali)

S.pop() e S.top() sono definite iff S.empty() = false

S.empty(), S.size(), S.top() lasciano S immutato
S.empty() = true iff 0 = S.size()
S.push(t) ; S.pop() = S
S.push(t) ; S.top() = t
S.push(t) ; S.push(t).sizie() = S.size() + 1
S.pop(); S.pos().size = S.size() - 1

NOTAZIONE STILE OOP

8

Cosa servono gli assiomi?

Teorema.
Assumiamo che (n = S.size()) e che S' sia
lo stack ottenuto dopo avere eseguito
k operazioni di push allora S'.size() = S.size() + k

9

Operazioni parziali

- Precondition of pop(Stack S):
  not empty(S)
- Precondition of top(Stack S)
  not empty(S)

10

ADT: visione costruttiva

- Operazioni sono caratterizzate da una precondizione e da una poscondizione.
- ***precondition*** *formula logica che caratterizza le proprieta' e il valore degli argomenti.*
- ***Postcondition*** *formula logica che caratterizza il risultato calcolato dall'operazione rispetto al valore degli argomenti.*

11

Esempio

- push(Stack S, Item I) -> Stack S'
  Precondition: S e' una istanza valida si stack && not full(S)
- Postcondition: S' e' una istanza valida di stack && S' = S esteso con I come elemento top

12

Astrazione sui dati via specifica

- con la specifica, astraiamo dall'implementazione del tipo di dato
- avendo gli oggetti insieme alle operazioni, l'astrazione diventa possibile
  - la rappresentazione è nascosta all'utente esterno, mentre è visibile all'implementazione delle operazioni
  - se una rappresentazione viene modificata, devono essere modificate le implementazioni delle operazioni, ma non le astrazioni che la utilizzano

13

Gli ingredienti della specifica di un tipo di dato astratto

- ✓ Java (parte sintattica della specifica)
  - classe o interfaccia
    - ✓per ora solo classi
  - nome per il tipo (la classe)
  - operazioni
    - ✓metodi di istanza, incluso il(i) costruttore(i)
- ✓ la specifica del tipo
  - fornita dalla clausola OVERVIEW che descrive i valori astratti degli oggetti ed alcune loro proprietà per esempio la modificabilità
- ✓ per il resto la specifica è una specifica dei metodi
  - strutturata tramite precondizioni (clausola require) e postcondizioni (clausola effect)

14

Formato della specifica

```
public class NuovoTipo {
  // OVERVIEW:  Gli oggetti di tipo NuovoTipo
  // sono collezioni modificabili di ..

  // costruttori
  public NuovoTipo ()
     // REQUIRES
     // EFFECTS: ...

  // metodi
  // specifiche degli altri metodi
}
```

15

ADT: specifica

ADT Immutable

1. **overview**
2. **abstract state**
3. **creators**
4. **observers**
5. **producers**
6. ~~**mutators**~~

ADT mutable

1. **overview**
2. **abstract state**
3. **creators**
4. **observers**
5. **producers (rare)**
6. **mutators**

- Creators: restituiscono nuovi valori dell'astrazione (Java: costruttori)
- Mutators: Modificano I valori dell'astrazione
- Observers: Restituiscono informazione sull'astrazione

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
  ...}
```

17

IntSet 2


```
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
}
```

18

IntSet: analisi


```
public class IntSet {
 // OVERVIEW: un IntSet è un insieme modificabile
 // di interi di dimensione qualunque
  ....
}
```

- i valori astratti degli oggetti della classe sono descritti nella specifica in termini di concetti noti
  - gli insiemi matematici
  - Uso di una notazione matematica (in seguito)
- gli stessi concetti sono anche utilizzati nella specifica dei metodi
  - aggiungere, togliere elementi
  - appartenenza, cardinalità

19

IntSet: analisi


```
public class IntSet {
 // OVERVIEW: ...
 // costruttore
  public IntSet ()
    // EFFECTS: inizializza this all'insieme
   vuoto
...}
```

- un solo costruttore (senza parametri)
  - inizializza this (l'oggetto nuovo)
  - non è possibile vedere lo stato dell'oggetto tra la creazione e l'inizializzazione

20

IntSet: analisi

```
public class IntSet {
...
 // metodi
  public void insert (int x)
     // EFFECTS: aggiunge x a this
  public void remove (int x)
     // EFFECTS: toglie x da this
...}
```

- modificatori
  - modificano lo stato del proprio oggetto
  - notare che nè insert nè remove sollevano eccezioni
    - ✓se si inserisce un elemento che c'è già
    - ✓se si rimuove un elemento che non c'è

21

IntSet: analisi

```
 public boolean isIn (int x)
     // EFFECTS: se x appartiene a this ritorna
    // true, altrimenti false
 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...}
```

- osservatori
  - non modificano lo stato del proprio oggetto: choose può sollevare un'eccezione (se l'insieme è vuoto)
    - ✓EmptyException può essere unchecked, perché l'utente può utilizzare size per evitare di farla sollevare
    - ✓choose è sottodeterminata (implementazioni corrette diverse possono dare diversi risultati)

22

Specifica di un tipo "primitivo"

- le specifiche sono ovviamente utili **anche** per capire ed utilizzare correttamente i tipi di dato "primitivi" di Java
- vedremo, come esempio, il caso dei vettori
  - Vector
  - arrays dinamici che possono crescere e accorciarsi
  - sono definiti nel package java.util

Vector 1


```
public class Vector {
 // OVERVIEW: un Vector è un array modificabile
 // di dimensione variabile i cui elementi sono
 // di tipo Object: indici tra 0 e size - 1
 // costruttore
  public Vector ()
    // EFFECTS: inizializza this a vuoto
 // metodi
  public void add (Object x)
     // EFFECTS: aggiunge una nuova posizione a
    // this inserendovi x
  public int size ()
     // EFFECTS: ritorna il numero di elementi di
    // this
  ...}
```

Vector 2

```
...
 public Object get (int n) throws
    IndexOutOfBoundsException
      // EFFECTS: se n<0 o n>= this.size solleva
     // IndexOutOfBoundsException, altrimenti
     // ritorna l'oggetto in posizione n in this
 public void set (int n, Object x) throws
    IndexOutOfBoundsException
      // EFFECTS: se n<0 o n>= this.size solleva
     // IndexOutOfBoundsException, altrimenti
     // modifica this sostituendovi l'oggetto x in
     // posizione n
 public void remove (int n) throws
    IndexOutOfBoundsException
      // EFFECTS: se n<0 o n>= this.size solleva
     // IndexOutOfBoundsException, altrimenti
     // modifica this eliminando l'oggetto in
     // posizione n            }
```


Vector: analisi

```
public class Vector {
 // OVERVIEW: un Vector è un array modificabile
 // di dimensione variabile i cui elementi sono
 // di tipo Object: indici tra 0 e size - 1
  ....
}
```

- gli oggetti della classe sono descritti nella specifica in termini di concetti noti: gli arrays
- gli stessi concetti sono anche utilizzati nella specifica dei metodi
  - indice, elemento identificato dall'indice
- il tipo è modificabile come l'array
- notare che gli elementi sono di tipo Object
  - non possono essere int, bool e char

Vector: analisi


```
public class Vector {
 // OVERVIEW: un Vector è un array modificabile
 // di dimensione variabile i cui elementi sono
 // di tipo Object: indici tra 0 e size - 1
 // costruttore
  public Vector ()
    // EFFECTS: inizializza this a vuoto
...}
```

- un solo costruttore (senza parametri)
  - inizializza this (l'oggetto nuovo) ad un "array" vuoto

Vector: analisi


```
public void add (Object x)
     // EFFECTS: aggiunge una nuova posizione a
     // this inserendovi x
public void set (int n, Object x) throws
    IndexOutOfBoundsException
     // EFFECTS: se n<0 o n>= this.size solleva
     // IndexOutOfBoundsException, altrimenti modifica
     // this sostituendovi l'oggetto x in posizione n
public void remove (int n) throws
    IndexOutOfBoundsException
      // EFFECTS: se n<0 o n>= this.size solleva
      // IndexOutOfBoundsException, altrimenti modifica
     // this eliminando l'oggetto in posizione n
```

- sono modificatori
  - modificano lo stato del proprio oggetto
  - set e remove possono sollevare un'eccezione primitiva unchecked

Vector: analisi

```
public int size ()
     // EFFECTS: ritorna il numero di elementi di
    // this
public Object get (int n) throws
   IndexOutOfBoundsException
     // EFFECTS: se n<0 o n>= this.size solleva
    // IndexOutOfBoundsException, altrimenti
    // ritorna l'oggetto in posizione n in this
public Object lastElement ()
     // EFFECTS: ritorna l'ultimo oggetto in this
```

- sono osservatori
  - non modificano lo stato del proprio oggetto
  - get può sollevare un'eccezione primitiva unchecked

29

JAVA DOCS

- Andiamo a vdere cosa dice la documentazione standard di Java
- http://docs.oracle.com/javase/7/docs/api/java/util/Vector.html

30

I polinomi

```
public class Poly {
 // OVERVIEW: un Poly è un polinomio a
 // cofficienti interi non modificabile
 // esempio: c0 + c1*x + c2*x^2 + ...

 // 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

  ...}
```


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
...}
```

I polinomi

```
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
}
```


Poly: Analisi

```
public class Poly {
 // OVERVIEW: ...
 // 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
...}
```

- due costruttori overloaded
  - stesso nome (quello della classe)
  - diverso numero o tipo di parametri
  - la scelta tra metodi overloaded viene effettuata in base al numero e tipo di parametri
    - ✓ eventualmente a run time scegliendo il più specifico

Poly: analisi

```
public class Poly {
 // OVERVIEW: ...
 // 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
...}
```

- l'eccezione NegativeExponentExc non è definita qui
  - nello stesso package di Poly?
  - può essere unchecked, perché non è probabile che l'utente usi esponenti negativi

35

Poly: analisi

```
// 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
...
```

- non ci sono modificatori
  - il tipo è non modificabile!
  - degree e coeff sono osservatori
  - add, mul, sub e minus ritornano nuovi oggetti di tipo Poly

36

Aspetti metodologici

- per prima cosa si definisce la specifica
  - "scheletro" formato da headers, overview, pre e post condizioni di tutti i metodi
  - mancano la rappresentazione degli oggetti ed il codice dei corpi dei metodi
    - ✓che possono essere sviluppati in un momento successivo ed indipendentemente dallo sviluppo dei "moduli" che usano il nuovo tipo di dato
    - ✓è molto importante riuscire a differire le scelte relative alla rappresentazione

37

Aspetti metodologici

- se aggiungiamo il codice dei metodi ben-tipati alla specifica dei metodi
  - ✓la specifica può essere compilata
  - ✓possono essere compilate implementazioni di moduli che la utilizzano (errori rilevati subito dall'analisi statica)
  - ✓Possono essere progettati i test di analisi
  - ✓Esempio: Junit e' basato su questa idea.

38

Un cliente di `IntSet`

```
public static IntSet getElements (int[] a) throws
    NullPointerException
     // EFFECTS: a=null solleva NullPointerException
    // altrimenti, restituisce un insieme che contiene
    // tutti e soli gli interi presenti in a
{IntSet s = new IntSet();
 for (int i = 0; i < a.length; i++)
      s.insert(a[i]);
 return s;
}
```

- scritta solo conoscendo la specifica di IntSet
  - non accede all'implementazione
    - ✓Ora non esiste ancora. Ma anche se ci fosse non potrebbe "vederla"
  - costruisce, accede e modifica l'oggetto solo attraverso i metodi (incluso il costruttore)

39

Verifiche

```
public static IntSet getElements (int[] a) throws
    NullPointerException
     // EFFECTS: a=null solleva NullPointerException
    // altrimenti, restituisce un insieme che contiene
    // tutti e soli gli interi presenti in a
{IntSet s = new IntSet();
 for (int i = 0; i < a.length; i++)
      s.insert(a[i]);
 return s;
}
```

- insert non ha precondizione
  - se ci fosse una precondizione bisognerebbe
    - ✓inserire in *getElements* il codice che la verifichi (run time check), oppure
    - ✓dimostrare che la precondizione è sempre verificata (verifica "statica")

40

Quale e' il punto?

- *Client—Supplier*
- *Supplier: fornisce il servizio con un ADT*
- *Client: utente del servizio (a sua volta fornitore di altri servizi)*
- *Visione moderna che si ritrova*
  - *Service oriented computing*
  - *Cloud computing*

41

Aspetti significativi (ACM SIG Soft. Eng)

- Supplier:
  - efficient and reliable algorithms and data structures,
  - convenient implementation
  - easy maintenance.
- Clients:
  - using the supplier services without effort to understand its internal details,
  - having a sufficient, but not overwhelming, set of operations.

42