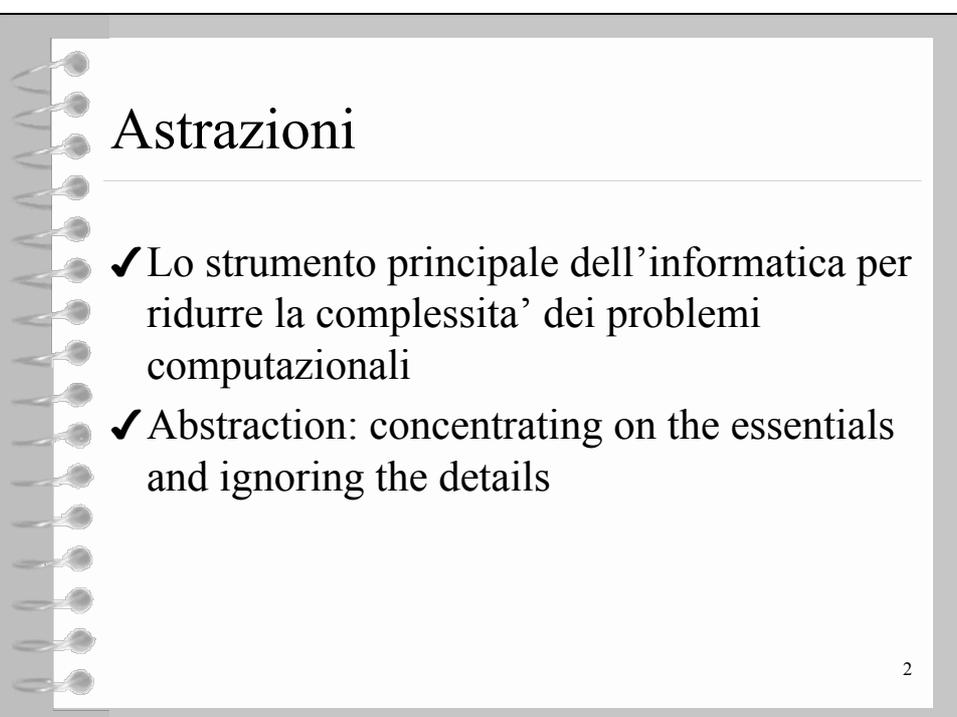
A graphic of a spiral-bound notebook with a grey cover and a white page. The spiral binding is on the left side. The text is written on the page.

Astrazioni sui dati : Specifica di Tipi di Dato Astratti in Java

1

A graphic of a spiral-bound notebook with a grey cover and a white page. The spiral binding is on the left side. The text is written on the page.

Astrazioni

- ✓ Lo strumento principale dell'informatica per ridurre la complessità dei problemi computazionali
- ✓ Abstraction: concentrating on the essentials and ignoring the details

2

Forme di astrazione

- ✓ **Procedural abstraction**: separazione delle proprietà logiche di una azione computazionale dai dettagli implementativi
- ✓ **Data abstraction**: separazione delle proprietà logiche dei dati dai dettagli della rappresentazione dei dati.

3

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

4

Esempio (ADT via Assiomi)

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

5

ADT

- ✓ *Elementi (Insiemi)*
- ✓ *Item*
- ✓ *Bool = { False, True }*
- ✓ *Stack (insieme di tutti gli stack)*

6

ADT

✓ Operazioni (Segnatura)

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

7

ADT: assiomi

- ✓ $\text{top}(\text{push}(s,x)) = x$
- ✓ $\text{pop}(\text{push}(s,x)) = s$
- ✓ $\text{empty}(\text{create}()) = \text{True}$
- ✓ $\text{empty}(\text{push}(s,x)) = \text{False}$

8

Operazioni parziali

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

9

Descructors?

- ✓ Sono operazioni che dipendono dall'ambiente operativo in cui viene utilizzato il tipo di dato astratto

10

ADT: visione costruttiva

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

11

Esempio

- ✓ $\text{push}(\text{StackB } S, \text{Item } I) \rightarrow \text{StackB } S'$
Precondition: S is a valid StackB instance
&& $\text{not full}(S)$
- ✓ Postcondition: S' is a valid StackB instance
&& $S' = S$ with I added as the new top element

12

Specifica ed Implementazione di Tipi di Dato Astratti in Java

- ✓ cos'è un tipo di dato astratto
- ✓ specifica di tipi di dati astratti
 - un tipo modificabile (`IntSet`)
 - qualche tipo primitivo
 - un tipo non modificabile (`Poly`)
- ✓ implementazione di tipi di dati astratti
 - definire la rappresentazione
 - le restrizioni nell'uso di Java
 - un record type (l'eccezione!)
 - l'implementazione di `IntSet` e `Poly`
- ✓ alcuni metodi “ereditabili”

13

Perché l'astrazione sui dati

- ✓ il più importante tipo di astrazione
- ✓ il nucleo (non esclusivo) della programmazione orientata ad oggetti
- ✓ lo scopo è quello di estendere il linguaggio
 - nel nostro caso, Javacon nuovi tipi di dato
- ✓ quali?
 - dipende dall'applicazione
 - interprete: stacks e tabelle di simboli
 - applicazione bancaria: conti
 - applicazioni numeriche: matrici

14

Cos'è un'astrazione sui dati

- ✓ in ogni caso è
 - un insieme di oggetti
 - stacks, conti, matrici
 - + un insieme di operazioni
 - per crearli e manipolarli
- ✓ i nuovi tipi di dato dovrebbero incorporare i due meccanismi di astrazione
 - parametrizzazione
 - simile al caso delle astrazioni procedurali
 - quella più importante si realizza con il polimorfismo
 - specifica
 - porta a vedere necessariamente insieme oggetti ed operazioni

15

Astrazione sui dati via specifica

- ✓ con la specifica, astraiamo dall'implementazione del tipo di dato
 - dalla sua rappresentazione
- ✓ se avessimo solo la rappresentazione degli oggetti non sarebbe possibile
 - o l'utente opera direttamente sulla rappresentazione
 - non ci sarebbe astrazione
 - se la rappresentazione viene modificata, la modifica si ripercuote su tutti gli utenti
 - oppure non si rende possibile la manipolazione degli oggetti del nuovo tipo
- ✓ 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
 - è il tipo di modifica più comune durante la manutenzione

16

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
 - nome della classe
 - operazioni
 - metodi di istanza, incluso il(i) costruttore(i)
- ✓ la specifica del tipo
 - fornita dalla clausola OVERVIEW
 - 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 come già abbiamo visto per le astrazioni procedurali
 - l'oggetto su cui i metodi operano è indicato nella specifica da this

17

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  
}
```

18

L'insieme di interi 1

```
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  
    ...}
```

19

L'insieme di interi 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  
}
```

20

IntSet: commenti 1

```
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
 - in questo caso, gli insiemi matematici
- ✓ gli stessi concetti sono anche utilizzati nella specifica dei metodi
 - aggiungere, togliere elementi
 - appartenenza, cardinalità

21

IntSet: commenti 2

```
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

22

IntSet: commenti 3

```
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

23

IntSet: commenti 4

```
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)

24

Specifica di un tipo “primitivo”

- ✓ le specifiche sono ovviamente utili 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`

25

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  
    ...}
```

26

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 }
```

27

Vector: commenti 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
    ....
}
```

- ✓ gli oggetti della classe sono descritti nella specifica in termini di concetti noti
 - in questo caso, 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
 - per esempio, non possono essere int, bool e char

28

Vector: commenti 2

```
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

29

Vector: commenti 3

```
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

30

Vector: commenti 4

```
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
```

31

I polinomi 1

```
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
    ...}
```

32

I polinomi 2

```
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
    ...}

```

33

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
    }

```

34

Poly: commenti 1

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

- ✓ gli oggetti della classe sono descritti nella specifica in termini di concetti noti
 - in questo caso, i polinomi
- ✓ gli stessi concetti sono anche utilizzati nella specifica dei metodi
 - operazioni di +, *, e -

35

Poly: commenti 2

```
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
 - se no, errore di compilazione
 - la scelta tra metodi overloaded viene effettuata in base al numero e tipo di parametri
 - eventualmente a run time scegliendo il più specifico

36

Poly: commenti 3

```
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 cxn  
    ...}
```

- ✓ 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

37

Poly: commenti 4

```
// 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`

38

Proprietà della Specifica

- ✓ 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
 - se aggiungiamo corpi contenenti return 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)

39

Implementazione di un metodo che usa 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
 - non esiste ancora
 - anche se ci fosse non potrebbe “vederla”
 - costruisce, accede e modifica l’oggetto solo attraverso i metodi (incluso il costruttore)

40

Verifiche nel metodo che usa 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;
}
```

✓ insert non ha preconditione

- se ci fosse una preconditione c bisognerebbe
 - inserire in getElements il codice che verifichi c (run time check), oppure
 - dimostrare che c è sempre verificata (verifica "statica")

41

Implementazione di un metodo che usa Poly

```
public static Poly diff (Poly p) throws
    NullPointerException
    // EFFECTS: p=null solleva NullPointerException
    // altrimenti, restituisce il polinomio che risulta
    // dalla differenziazione di p
{Poly q = new Poly();
  for (int i = 0; i <= p.degree(); i++)
    q = q.add(new Poly(p.coeff(i)*i, i-1));
  return q;
}
```

✓ scritta solo conoscendo la specifica di Poly

- non accede all'implementazione
 - non esiste ancora
 - anche se ci fosse non potrebbe "vederla"
- costruisce ed accede l'oggetto solo attraverso i metodi (incluso il costruttore)
- nessuna preconditione -> nessuna verifica

42

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*

43

Aspetti significativo

- ✓ **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.

44

client-supplier contract

- ✓ Client: the conditions under which the supplier must deliver results -- when the *preconditions of the operations are satisfied*
- ✓ Supplier: *the benefits the supplier must deliver -- make the postconditions hold at the end of the operation (i.e., the operations deliver the correct results).*