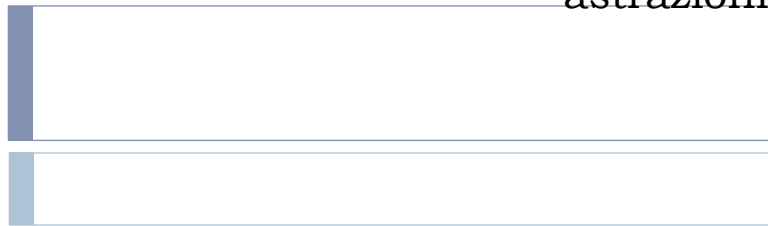


Programmazione
=
decomposizione basata su
astrazioni



1

Decomposizione in “moduli”

- ▶ necessaria quando si devono sviluppare programmi abbastanza grandi
 - ▶ decomporre il problema in sotto-problemi
 - ▶ i moduli che risolvono i sotto-problemi devono riuscire a cooperare nella soluzione del problema originale
 - ▶ Viviamo in un mondo di API
- ▶ metodologie
 - ▶ si deve poter lavorare in modo indipendente (ma coerente) nello sviluppo dei diversi moduli
 - ▶ deve essere possibile eseguire “facilmente” modifiche e aggiornamenti (manutenzione)
 - ▶ a livello dei singoli moduli, senza influenzare il comportamento degli altri
- ▶ i programmi devono essere decomposti in moduli, in modo che sia facile capirne le interazioni

▶ 2

Decomposizione e astrazione

- ▶ la decomposizione può essere effettuata in modo produttivo ricorrendo all'**astrazione**
 - ▶ cambiamento del livello di dettaglio, nella descrizione di un problema, limitandosi a “considerare” solo alcune delle sue caratteristiche
 - ▶ perché si spera di semplificare l'analisi, separando gli attributi che si ritengono rilevanti da quelli che si ritiene possano essere trascurati
 - ▶ la rilevanza dipende dal contesto

▶ 3

Meccanismi di astrazione

- ▶ Quali sono i meccanismi di astrazione legati alla programmazione
- ▶ lo strumento fondamentale è l'utilizzazione di **linguaggi ad alto livello**
 - ▶ enorme semplificazione per il programmatore
 - ▶ usando direttamente i costrutti del linguaggio ad alto livello invece che una delle numerosissime sequenze di istruzioni in linguaggio macchina “equivalenti”

▶ 4

I linguaggi non bastano

```
// ricerca all'insù
found = false;
for (int i = 0; i < a.length; i++)
    if (a[i] == e) {
        z = i; found = true;}
// ricerca all'ingiù
found = false;
for (int i = a.length - 1; i >= 0; i--)
    if (a[i] == e) {
        z = i; found = true;}
```

- ▶ sono diversi
 - ▶ possono dare risultati diversi
 - ▶ potrebbero essere stati scritti con l'idea di risolvere lo stesso problema
 - ▶ verificare se l'elemento è presente nell'array e restituire una posizione in cui è contenuto

▶ 5

Migliori astrazioni nel linguaggio?

- ▶ il linguaggio potrebbe avere delle potenti operazioni sull'array del tipo `isIn` e `indexOf`

```
// ricerca indipendente dall'ordine
found = a.isIn(e);
if found z = a.indexOf(e);
```

- ▶ l'astrazione è scelta dal progettista del linguaggio
 - ▶ quali e quante?
 - ▶ quanto complicato diventa il linguaggio?
- ▶ meglio progettare linguaggi dotati di **meccanismi che permettano di definire le astrazioni che servono**

▶ 6

Il più comune tipo di astrazione

- ▶ **l'astrazione procedurale**
 - ▶ presente in tutti i linguaggi di programmazione
- ▶ **la separazione tra “definizione” e “chiamata” rende disponibili nel linguaggio i due meccanismi fondamentali di astrazione**
 - ▶ **l'astrazione attraverso parametrizzazione**
 - ▶ si astrae dall'identità di alcuni dati, rimpiazzandoli con parametri
 - ▶ si generalizza un modulo per poterlo usare in situazioni diverse
 - ▶ **l'astrazione attraverso specifica**
 - ▶ si astrae dai dettagli dell'implementazione del modulo, per limitarsi a considerare il comportamento che interessa a chi utilizza il modulo (ciò che fa, non come lo fa)
 - ▶ si rende ogni modulo indipendente dalle implementazioni dei moduli che usa

▶ 7

Astrazione via parametrizzazione

- ▶ **l'introduzione dei parametri permette di descrivere un insieme (anche infinito) di computazioni diverse con un singolo programma che le astrae tutte**

`x * x + y * y`

- ▶ **descrive una computazione**

`λx,y:int.(x * x + y * y)`

- ▶ **descrive tutte le computazioni che si possono ottenere chiamando la procedura, cioè applicando la funzione ad una opportuna coppia di valori**

`λx,y:int.(x * x + y * y) (w, z)`

- ▶ **ha la stessa semantica dell'espressione `w * w + z * z`**

▶ 8

Astrazione via specifica

- ▶ la procedura si presta a meccanismi di astrazione più potenti della parametrizzazione
- ▶ possiamo astrarre dalla specifica computazione descritta nel corpo della procedura, associando ad ogni procedura una **specifica**
 - ▶ semantica intesa della procedura
- ▶ e derivando la semantica della chiamata dalla specifica invece che dal corpo della procedura
- ▶ non è di solito supportata dal linguaggio di programmazione
 - ▶ se non in parte (vedi specifiche di tipo)
- ▶ si realizza con opportune annotazioni
 - ▶ Esempio JML

▶ 9

JML: Esempio

```
public class BankingExample {
    public static final int MAX_BALANCE = 1000;
    private /*@ spec_public @*/ int balance;
    private /*@ spec_public @*/ boolean isLocked = false;

    /*@ public invariant balance >= 0 && balance <= MAX_BALANCE;
    /*@ assignable balance;
    /*@ ensures balance == 0;
    public BankingExample() { balance = 0; }

    /*@ requires 0 < amount && amount + balance < MAX_BALANCE;
    /*@ assignable balance;
    /*@ ensures balance == \old(balance + amount);
    public void credit(int amount) { balance += amount; }

    /*@ requires 0 < amount && amount <= balance;
    /*@ assignable balance;
    /*@ ensures balance == \old(balance) - amount;
    public void debit(int amount) { balance -= amount; }

    /*@ ensures isLocked == true;
    public void lockAccount() { isLocked = true; }

    /*@ requires !isLocked;
    /*@ ensures \result == balance;
    /*@ also
    /*@ requires isLocked;
    /*@ signals_only BankingException;
    public /*@ pure @*/ int getBalance() throws BankingException {
        if (!isLocked) { return balance; }
        else { throw new BankingException(); }
    }
}
```

▶ 10

Un esempio

```
float sqrt (float coef) {  
  // REQUIRES: coef > 0  
  // EFFECTS: ritorna una approssimazione  
  // della radice quadrata di coef  
  float ans = coef / 2.0; int i = 1;  
  while (i < 7) {  
    ans = ans - ((ans*ans-coef)/(2.0*ans));  
    i = i+1;  }  
  return ans;  }
```

- **precondizione (asserzione requires)**
 - deve essere verificata quando si chiama la procedura
- **postcondizione (asserzione effects)**
 - tutto ciò che possiamo assumere valere quando la chiamata di procedura termina, se al momento della chiamata era verificata la precondizione

▶ 11

Il punto di vista di chi usa la procedura

```
float sqrt (float coef) {  
  // REQUIRES: coef > 0  
  // EFFECTS: ritorna una approssimazione  
  // della radice quadrata di coef  
  ... }  
}
```

- **gli utenti della procedura non si devono preoccupare di capire cosa la procedura fa, astruendo le computazioni descritte dal corpo**
 - cosa che può essere molto complessa
- **gli utenti della procedura non possono osservare le computazioni descritte dal corpo e dedurre da questo proprietà diverse da quelle specificate dalle asserzioni**
 - **astruendo dal corpo (implementazione), si “dimentica” informazione evidentemente considerata non rilevante**

▶ 12

Tipi di astrazione

- ▶ parametrizzazione e specifica permettono di definire vari tipi di astrazione
 - ▶ **astrazione procedurale**
 - ▶ si aggiungono nuove operazioni a quelle della macchina astratta del linguaggio di programmazione
 - ▶ **astrazione di dati**
 - ▶ si aggiungono nuovi tipi di dato a quelli della macchina astratta del linguaggio di programmazione
 - ▶ **iterazione astratta**
 - ▶ permette di iterare su elementi di una collezione, senza sapere come questi vengono ottenuti
 - ▶ **gerarchie di tipo**
 - ▶ permette di astrarre da specifici tipi di dato a famiglie di tipi correlati

▶ 13

Astrazione procedurale

- ▶ fornita da tutti i linguaggi ad alto livello
- ▶ **aggiunge nuove operazioni a quelle della macchina astratta del linguaggio di programmazione**
 - ▶ per esempio, `sqrt` sui `float`
- ▶ la specifica descrive le proprietà della nuova operazione

▶ 14

Astrazione sui dati

- ▶ fornita da tutti i linguaggi ad alto livello moderni
- ▶ aggiunge nuovi tipi di dato e relative operazioni a quelli della macchina astratta del linguaggio
 - ▶ tipo `MultiInsieme` con le operazioni `vuoto`, `inserisci`, `rimuovi`, `numeroDi` e `dimensione`
 - ▶ la rappresentazione dei valori di tipo `MultiInsieme` e le operazioni sono realizzate nel linguaggio
 - ▶ l'utente non deve interessarsi dell'implementazione, ma fare solo riferimento alle proprietà presenti nella specifica
 - ▶ le operazioni sono astrazioni definite da asserzioni come
$$\text{dimensione}(\text{inserisci}(s,e)) = \text{dimensione}(s) + 1$$
$$\text{numeroDi}(\text{vuoto}(),e) = 0$$
- ▶ la specifica descrive le relazioni fra le varie operazioni
 - ▶ per questo, è cosa diversa da un insieme di astrazioni procedurali

▶ 15

Iterazione astratta

- ▶ non è fornita da nessun linguaggio di uso comune
 - ▶ Java ha costrutti che permettono una simulazione relativamente semplice
- ▶ permette di iterare su elementi di una collezione, senza sapere come questi vengono ottenuti
- ▶ evita di dire cose troppo dettagliate sul flusso di controllo all'interno di un ciclo
 - ▶ per esempio, potremmo iterare su tutti gli elementi di un `MultiInsieme` senza imporre nessun vincolo sull'ordine con cui vengono elaborati
- ▶ astrae (nasconde) il flusso di controllo nei cicli

▶ 16

Gerarchie di tipo

- ▶ fornite da alcuni linguaggi ad alto livello moderni
 - ▶ per esempio, Java
- ▶ permettono di astrarre gruppi di astrazioni di dati (tipi) a famiglie di tipi
- ▶ i tipi di una famiglia condividono alcune operazioni
 - ▶ definite nel supertype, di cui tutti i tipi della famiglia sono subtypes
- ▶ una famiglia di tipi astrae i dettagli che rendono diversi tra loro i vari tipi della famiglia
- ▶ in molti casi, il programmatore può ignorare le differenze

▶ 17

Astrazione e programmazione orientata ad oggetti

- ▶ il tipo di astrazione più importante per guidare la decomposizione è l'astrazione sui dati
 - ▶ gli iteratori astratti e le gerarchie di tipo sono comunque basati su tipi di dati astratti
- ▶ l'astrazione sui dati è il meccanismo fondamentale della **programmazione orientata ad oggetti**
 - ▶ anche se esistono altre tecniche per realizzare tipi di dato astratti
 - ▶ per esempio, all'interno del paradigma di programmazione funzionale

▶ 18