Programmazione II

3. Astrazione Procedurale e Specifiche (PDJ 3, 9)

Dragan Ahmetovic

Astrazione

Meccanismi di astrazione

Permettono di trascurare **dettagli implementativi** e concentrarsi sulle **caratteristiche** del problema

- Astrazione per **parametrizzazione**
  - Trascura i **valori effettivi** utilizzati nei calcoli e si concentra sul loro **tipo**
  - Potrò sostituire i **parametri** con i **valori effettivi**, mantenendo valido il calcolo
  - L'astrazione per parametrizzazione è un'**astrazione sui dati**
- Astrazione per **specifica**
  - Trascura il **modo** in cui viene risolto, si concentra sulle **specifiche del problema**
  - Posso mantenere la validità del programma anche se **implementato in maniera diversa**
  - L'astrazione per specifica è un'**astrazione sui vincoli dell'operazione**

Astrazione

Caratteristiche vantaggiose fornite dall'utilizzo di astrazioni

- **Località**
  - posso considerare (definire e implementare) una data astrazione in maniera **disgiunta dal resto**
  - chi usa l'implementazione dell'astrazione dovrà guardare **solo la specifica**
- **Modificabilità**
  - posso **vincolare l'uso dell'astrazione**, ma non la sua implementazione
  - Bug e problemi di performance si possono sistemare in un **secondo momento**
  - Modifiche future si possono **prevedere** rendendo il codice facilmente modificabile

Tipi di astrazione

- Astrazione Procedurale
- Astrazione sui tipi di dati
- Astrazione di iterazione
- Gerarchie di tipi

Astrazione Procedurale

In cosa consiste l'astrazione procedurale?

Nell'astrazione procedurale, la meccanica dell'astrazione per parametrizzazione permette:

- Definizione della procedura come **blocco di codice indipendente**
- Parametrizzazione degli **ingressi e uscite**
  - In questo modo il codice si può richiamare per tutti i problemi simili

Invece, l'astrazione per specifica è resa mediante:

- Intestazioni delle procedure (**header** o signature)
- Specifiche rese attraverso **commenti** con l'obiettivo di:
  - Indicare **prerequisiti** sui parametri
  - Spiegare i possibili **risultati**
  - Segnalare eventuali **effetti collaterali**

Linguaggio di specifica

Come deve essere un linguaggio di specifica?

È necessario avere un modo di scrivere specifiche, **formale** o **informale**, ma comunque:

- **chiaro** - che sia noto quello che significano i **termini** utilizzati
- **coerente** - che **non muti** nel tempo e che tutti siano **concordi sulla semantica**
  - Può essere complesso ottenere **coerenza** usando un linguaggio informale.
  - Di contro, aiuta ad avere una **chiarezza** immediata.

Il linguaggio di specifica **NON è un linguaggio di programmazione**

- Ve ne sono diversi: Liskov, javadoc
- Evitare **specifiche operazionali** (non dire come ma cosa fare)
  - In alcuni casi possono essere utili (se serve un metodo ben noto per le sue proprietà)

Linguaggio di specifica

Specifiche di procedura (Liskov)

Sono definite attraverso 3+1 principali clausole di specifica:

- **requires** (pre-condizioni) - specifica i (possibili) vincoli sugli input della procedura
  - si omette se non vi sono vincoli su input (a parte il tipo di dato)
- **modifies** (effetti collaterali) - specifica gli input modificati dalla procedura
  - si omette se la procedura non modifica input
- **effects** (post-condizioni) - specifica gli effetti della procedura (output, modifiche causate...)
  - **deve essere presente**, utile confrontare lo stato pre e post esecuzione
- **overview** (su classe) - specifica l'obiettivo della classe e possibili funzionalità chiave

ATTENZIONE: La specifica deve essere il primo passo nella scrittura del codice e serve per progettare più agevolmente la struttura del codice

Linguaggio di specifica

Esempio:

```
1  public class Arrays {
2  // OVERVIEW: Provides a number of standalone procedures for manipulating arrays of ints
3
4    public static int search(int[] a, int x)
5    // EFFECTS: If x is in a, returns an index where x is stored; otherwise, returns -1
6
7    public static int searchSorted(int[] a, int x)
8    // REQUIRES: a is sorted in ascending order
9    // EFFECTS: If x is in a, returns an index where x is stored; otherwise, returns -1
10
11   public static void sort(int[] a)
12   // MODIFIES: a
13   // EFFECTS: Rearranges the elements of a into ascending order
14   //          e.g., if a= [3, 1, 6, 1] before the call, on return a= [1, 1, 3, 6]
15   //          e.g., if a= [3, 1, 6, 1], a_post = [1, 1, 3, 6]. (forma alternativa)
16 }
```

Le modifiche possono accadere anche negli input **impliciti** (ambiente, variabili di classe, ...)

```
1    public static void copyLine()
2    // REQUIRES: System.in contains a line of text
3    // MODIFIES: System.in and System.out
4    // EFFECTS: Reads a text line from System.in (advancing the cursor to the end of the line)
        and writes it on System.out
```

Linguaggio di specifica

Esempio JavaDOC (Si può generare HTML usando “javadoc Test.java”)

```java
/**
 * Description of the class
 * <p>
 * Longer description of the class
 *
 * @author    Firstname Lastname address@example.com
 * @version   1.6 (current version number of program)
 * @since     1.2 (the version of the package this class was first added to)
 */
public class Test {
  /**
   * Short description of the method.
   * <p>
   * Longer description, may include whatever we would put in MODIFIES.
   * Possibly also EFFECTS on implicit input and other collateral effects.
   *
   * @param  parameter Description of the parameter, i.e.: REQUIRES.
   * @return Description of the returned values, i.e.: EFFECTS.
   */
  public int methodName (int parameter) {
    // method body with a return statement
  }
}
```

Linguaggio di specifica

Proprietà delle specifiche

Una buona specifica ha le seguenti proprietà:

- **restrittività** - dovrebbe escludere le implementazioni non corrette
- **minimalità** - dovrebbe porre il meno possibile vincoli sull'implementazione
  - Questo ha come effetto una maggiore **generalità** delle procedure
  - inoltre potrebbe permettere la **sottodeterminazione**
  - ovvero **comportamento non determinato** - le implementazioni potrebbero differire nei risultati
  - ma le procedure dovrebbero sempre avere un comportamento **deterministico**
- **chiarezza** - dovrebbe essere chiara semanticamente (anche considerando altre persone)
  - **ridondanza** - dovrebbe chiarire il significato mediante ulteriori spiegazioni o esempi

Linguaggio di specifica

Implementazione:

```
1  public static int searchSorted (int[] a, int x) {
2  // REQUIRES: a is sorted in ascending order
3  // EFFECTS: If x is in a, returns an index where x is stored; otherwise, returns -1.
4  // uses linear search
5    if(a == null)
6      return -1;
7
8    for(int i = 0; i < a.length; i++)
9      if(a[i] == x)
10       return i;
11     else if(a[i] > x)
12       return -1;
13
14   return -1;
15 }
```

Linguaggio di specifica

Implementazione:

```
1  public static int searchSorted (int[] a, int x) {
2  // REQUIRES: a is sorted in ascending order
3  // EFFECTS: If x is in a, returns an index where x is stored;
4  //          otherwise, returns -1.
5  // uses binary recursive search
6    if(a == null)
7      return -1;
8    else
9      return binarySearch(a, x, 0, a.size());
10 }
11
12 private static int binarySearch(int[] a, int x, int low, int high) {
13 // REQUIRES: a is not null and sorted, and 0 <= low & high < a.length
14 // EFFECTS: If high < low return -1;
15 //          if x < (>) mid value, repeat search in lower (higher) half range;
16 //          otherwise return its index;
17   int mid = low + ((high - low) / 2);
18
19   if(high < low)
20     return -1;
21   else if(x < a[mid])
22     return binarySearch(a, x, low, mid - 1);
23   else if(x > a[mid])
24     return binarySearch(a, x, mid + 1, high);
25
26   return mid;
27 }
```

Proprietà delle procedure

Semplicità

- Deve essere **chiaro** cosa fa una procedura.
- Se fatico a **trovarle un nome** efficace, non è sufficientemente semplice.
- Può accadere se faccio **diversi compiti distinti** in una sola procedura.

Generalità

Proprietà che determina il **contesto di applicabilità** di una procedura.

- Una procedura è **più generale** se accetta un'**insieme più ampio di valori di input**
- Può essere utilizzata per **risolvere una gamma più ampia di problemi**
- Si può **rendere più generale** mediante parametrizzazione delle variabili o delle assunzioni
  - Auspicabile se una procedura più generale ha una sua utilità

Proprietà delle procedure

Procedure **totali**

- Ovvero **senza vincoli sul dominio** - tutti gli input sono validi
- A causa di questo, le procedure totali **non richiedono** la clausola **requires**
- **Compilatore stesso controlla** l'unico vincolo di validità, ovvero il tipo di input
- Qualsiasi input fornito produce un risultato **valido e prevedibile**

Procedure **parziali**

- Ovvero **con vincoli** sui valori dei parametri (sul **dominio**)
- Richiedono la clausola **requires**
- Bisogna specificare su **quali parametri** la procedura può funzionare
- In tutti gli altri casi il **comportamento non è definito**
  - Ovvero può succedere qualunque cosa (crash, hang, output di qualche genere)
  - Il comportamento però deve essere coerente a parità di input (**deterministico**)

Proprietà delle procedure

Esempio di procedura parziale:

```
1  public static double square(double n){
2  //REQUIRES: n > 0
3  //EFFECTS: returns square root of n
4    double t = n / 2;
5    double r = (t + (n / t)) / 2;
6
7    while ((t - r) != 0) {
8      t = r;
9      r = (t + (n / t)) / 2;
10   }
11
12   return r;
13 }
```

Proprietà delle procedure

Perchè costruire procedure parziali?

- Le procedure **totali** sono più **sicure**
- Le procedure **parziali** possono essere più **performanti**
- Inoltre possono essere più **facili da scrivere**

Quando fare una procedura **parziale** e quando fare una procedura **totale**?

- **Totale** per **ridurre errori** e rendere **più generale** l'utilizzo
- **Parziale** se il **contesto d'uso è limitato** e ben noto, e per grandi benefici di **performance**
- È possibile **rendere la procedura totale** gestendo tutti gli input.
- Non è sempre possibile capire se l'input è valido oppure no dal principio

Proprietà delle procedure

Esempio di procedura resa totale:

```
1  public static double square(double n){
2  //EFFECTS: if n <= 0 returns 0, otherwise returns square root of n.
3    if(n <= 0)
4      return 0;
5
6    double t = n / 2;
7    double r = (t + (n / t)) / 2;
8
9    while ((t - r) != 0) {
10     t = r;
11     r = (t + (n / t)) / 2;
12   }
13
14   return r;
15 }
```

Programmazione II

3. Astrazione Procedurale e Specifiche (PDJ 3, 9)

Dragan Ahmetovic