LE GERARCHIE DI TIPI: IMPLEMENTAZIONI MULTIPLE E PRINCIPIO DI SOSTITUZIONE

1

utilizzare una gerarchia di tipi

- implementazioni multiple di un tipo
  - i sottotipi non aggiungono alcun comportamento nuovo
  - la classe che implementa il sottotipo implementa esattamente il comportamento definito dal supertipo il sottotipo estende il comportamento del suo supertipo fornendo nuovi metodi
- dal punto di vista semantico, supertipo e sottotipo sono legati dal principio di sostituzione
  - che definisce esattamente il tipo di astrazione coinvolto nella definizione di gerarchie di tipo

2

Implementazioni multiple

- il tipo superiore della gerarchia definisce una famiglia di tipi tale per cui
  - tutti i membri hanno esattamente gli stessi metodi e la stessa semantica che forniscono l'implementazione di tutti i metodi astratti, in accordo con le specifiche del supertipo
  - gli oggetti dei sottotipi vengono dall'esterno tutti visti come oggetti dell'unico supertipo
  - dall'esterno si vedono solo i costruttori dei sottotipi

3

`IntList`

- il supertipo è una classe astratta
- usiamo i sottotipi per implementare i due casi della definizione ricorsiva
  - lista vuota
  - lista non vuota
- la classe astratta ha alcuni metodi non astratti
  - comuni alle due sottoclassi
  - definiti in termini dei metodi astratti
- la classe astratta non ha variabili di istanza e quindi nemmeno costruttori

4

supertipo IntList

```
public abstract class IntList {
 // OVERVIEW: un IntList è una lista modificabile
 // di Integers.
 // Elemento tipico [x1,...,xn]
 public abstract Integer first () throws EmptyException;
  // EFFECTS: se this è vuoto solleva EmptyException,
  // altrimenti ritorna il primo elemento di this
 public abstract IntList rest () throws EmptyException;
  // EFFECTS: se this è vuoto solleva EmptyException,
  // altrimenti  ritorna la lista ottenuta da this]
  //togliendo il primo elemento
 public abstract IntList addEl (Integer x);
  // EFFECTS: aggiunge x all'inizio di this
 public abstract int size ();
  // EFFECTS: ritorna il numero di elementi di this
 public abstract boolean repOk ();
 public String toString ();
 public boolean equals (IntList o);
}
```

5

Implementazione EmptyIntList

```
public class EmptyIntList extends IntList {
 public EmptyIntList () {}
 public Integer first () throws EmptyException
 { throw new EmptyException
   ("EmptyIntList.first"); }
 public IntList rest () throws EmptyException
 { throw new EmptyException
   ("EmptyIntList.rest"); }
 public IntList addEl (Integer x)
 {return new FullIntList(x);}
 public int size () {....}
 public boolean repOk () {....}
}
```

6

Implementazione `FullIntList`

```
public class FullIntList extends IntList {
 private int sz;
 private Integer val;
 private IntList next;
 public FullIntList (Integer x)
 {sz = 1; val = x;
  next = new EmptyIntList(  ); }
 public Integer first () {return val; }
 public IntList rest () { return next;  }
 public IntList addEl (Integer x)
 {FullIntList n = new FullIntList(x);
 n.next = this; n.sz = this.sz + 1; return n; }
 public int size () {....}
 public boolean repOk () {....}
}
```


Principio di sostituzione

- un oggetto del sottotipo può essere sostituito ad un oggetto del supertipo senza influire sul comportamento dei programmi che utilizzano il tipo
  - i sottotipi supportano il comportamento del supertipo
  - per esempio, un programma scritto in termini del tipo Intlist può lavorare correttamente su oggetti del tipo FullIntList
- il sottotipo deve soddisfare le specifiche del supertipo
- astrazione via specifica per una famiglia di tipi
  - astraiamo diversi sottotipi a quello che hanno in comune: la specifica del loro supertipo

Principio di sostituzione

- devono essere supportate
  - la regola della segnatura
    - ✓ gli oggetti del sottotipo devono avere tutti i metodi del supertipo
    - ✓ le segnature dei metodi del sottotipo devono essere compatibili con le segnature dei corrispondenti metodi del supertipo
  - la regola dei metodi
    - ✓ le chiamate dei metodi del sottotipo devono comportarsi come le chiamate dei corrispondenti metodi del supertipo
  - la regola delle proprietà
    - ✓ il sottotipo deve preservare tutte le proprietà che possono essere provate sugli oggetti del supertipo
- tutte le regole riguardano la semantica!

Regola della segnatura

- se una chiamata è type-correct per il supertipo lo è anche per il sottotipo
  - garantita dal compilatore Java
    - ✓ che permette solo che i metodi del sottotipo sollevino meno eccezioni di quelli del supertipo
    - ✓ da Java 5 un metodo della sottoclasse puo' sovrascrivere un metodo della superclasse con la stessa firma fornendo un return type piu' specifico.
      - http://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html#jls-8.4.8.3
    - ✓ e altre due regole non possono essere garantite dal compilatore Java
    - ✓ hanno a che fare con la specifica della semantica! ...

Covariante vs controvariante

- Una sottoclasse puo' riscrivere un metodo restituendo come risultato un valore di un sottotipo di quello previsto dal metodo della super-classe (covariant return type)
- Una nozione piu' liberale potrebbe avere argomenti contravarianti
  - Un tipo e' detto Covariante se mantiene l'ordinamento ≤ dei tipi
  - contravariante se inverte l'ordinamenti
  - invariante se non dipende dall'ordinamento

11

Classi vs Tipi

- Una classe definisce il comportamento degli oggetti (istanza)
  - Il meccanismo di ereditarieta puo' modificare i comportanti riscrivendo i metodi
- Un tipo definisce i comportamenti in termini del tipo dei metodi
- Sono concetti differenti e devono essere usati in modo coerente
  - Java/C# confondono le due nozioni: il nome di una classe e' il tipo degli oggetti
  - Praticamente questa confuzione e' utile ma si deve comprendere che sono due nozioni differenti

12

Regola dei metodi

- si può ragionare sulle chiamate dei metodi usando la specifica del supertipo anche se viene eseguito il codice del sottotipo
- garantito che va bene se i metodi del sottotipo hanno esattamente le stesse specifiche di quelli del supertipo
- come possono essere diverse?
  - se la specifica nel supertipo è nondeterministica (comportamento sottospecificato) il sottotipo può avere una specifica più forte che risolve (in parte) il nondeterminismo

13

Regola dei metodi

- in generale un sottotipo può indebolire le precondizioni e rafforzare le post condizioni
- per avere compatibilità tra le specifiche del supertipo e quelle del sottotipo devono essere soddisfatte le regole
  - regola delle precondizione
    - ✓pre_{super} ==> pre_{sub}
  - regola delle postcondizione
    - ✓pre_{super} && $post_{sub}$ ==> $post_{super}$

14

Regola dei metodi

- indebolire la precondizione
  - pre $_{super}$ ==> pre $_{sub}$

  ha senso, perché il codice che utilizza il metodo è scritto per usare il supertipo
  - ne verifica la precondizione
  - verifica anche la precondizione del metodo del sottotipo
- esempio: un metodo in IntSet

```
public void addZero (  )
// REQUIRES: this non è vuoto
// EFFECTS: aggiunge 0 a this
```

potrebbe essere ridefinito in un sottotipo

```
public void addZero (  )
// EFFECTS: aggiunge 0 a this
```

15

Regola dei metodi

- rafforzare la post condizione
  - pre $_{super}$ && post $_{sub}$ ==> post $_{super}$

  ha senso, perché il codice che utilizza il metodo è scritto per usare il supertipo
  - assume come effetti quelli specificati nel supertipo
  - gli effetti del metodo del sottotipo includono comunque quelli del supertipo (se la chiamata soddisfa la precondizione più forte)
- esempio: un metodo in IntSet

```
public void addZero (  )
// REQUIRES: this non è vuoto
// EFFECTS: aggiunge 0 a this
```

potrebbe essere ridefinito in un sottotipo

```
public void addZero (  )
// EFFECTS: se this non è vuoto aggiunge 0 a
   this altrimenti aggiunge 1 a this
```

16

Regola dei metodi: violazioni

consideriamo `insert` in `IntSet`

```
public class IntSet {
 public void insert (int x)
  // EFFECTS: aggiunge x a this
```

supponiamo di definire un sottotipo di `IntSet`
con la seguente specifica di `insert`

```
public class StupidoIntSet extends IntSet {
public void insert (int x)
// EFFECTS: aggiunge x a this se x è pari,
// altrimenti non fa nulla
```

17

Regola delle proprietà

- il ragionamento sulle proprietà degli oggetti basato sul supertipo è ancora valido quando gli oggetti appartengono al sottotipo
- proprietà degli oggetti (non proprietà dei metodi)
- da dove vengono le proprietà degli oggetti?
  - dal modello del tipo di dato astratto
    - ✓ le proprietà degli insiemi matematici, etc.
    - ✓ le elenchiamo esplicitamente nell'overview del supertipo
      - un tipo astratto potrebbe avere un numero infinito di proprietà
- proprietà invarianti
  - un FatSet non è mai vuoto
- proprietà di evoluzione
  - il grado di un Poly non cambia

18

Regola delle proprietà

- per mostrare che un sottotipo soddisfa la regola delle proprietà dobbiamo mostrare che preserva le proprietà del supertipo
- per le proprietà invarianti
  - bisogna provare che creatori e produttori del sottotipo stabiliscono l'invariante (solita induzione sul tipo)
  - che tutti i metodi (anche quelli nuovi, inclusi i costruttori) del sottotipo preservano l'invariante
- per le proprietà di evoluzione
  - bisogna mostrare che ogni metodo del sottotipo le preserva

Regola delle proprietà: una proprietà invariante

il tipo `FatSet` è caratterizzato dalla proprietà che i suoi oggetti non sono mai vuoti

```
// OVERVIEW: un FatSet è un insieme modificabile di interi la
// cui dimensione è sempre almeno 1
```

assumiamo che `FatSet` non abbia un metodo `remove`
ma invece abbia un metodo `removeNonEmpty`

```
public void removeNonEmpty (int x)
// EFFECTS: se x è in this e this contiene altri elementi
// rimuovi x da this
```

e abbia un costruttore che crea un insieme con almeno un elemento

possiamo provare che gli oggetti `FatSet` hanno dimensione maggiore di zero

Regola delle proprietà: una proprietà invariante

consideriamo il sottotipo ThinSet che ha tutti i metodi di FatSet con identiche specifiche e in aggiunta il metodo

```
public void remove (int x)
// EFFECTS: rimuove x da this
```

ThinSet non è un sottotipo legale di FatSet

(1) perché il suo extra metodo può svuotare l'oggetto

(2) l'invariante del supertipo non sarebbe conservato

21

Regola delle proprietà: una proprietà di evoluzione (non modificabilità)

- tipo SimpleSet che ha i due soli metodi insert e isIn
  - oggetti SimpleSet possono solo crescere in dimensione
  - IntSet non può essere un sottotipo di SimpleSet perché il metodo remove non conserva la proprietà

22