Abstract Data Types: API, ADT

Sommario: 24 Maggio, 2019

- Un Tipo Astratto in OCaml: `'a Set`
- Usiamo `'a Set`
- Estensione e Riuso in Java: `SetE<A>` Mutable
- Usiamo `Set<A>`
- Esaminiamo, Usiamo, e Ridefiniamo `APIMuInt`.
- Esercizi

Un Tipo Astratto in OCaml: Struttura dei Moduli

- **Tipo di Dato Astratto**
  Una Collezione di Valori definita dall'insieme delle sole operazioni che possono essere usate per introdurli e operare con essi.

- **Tipo di Dato**: In OCaml (funzionale) sono valori IMMUTABLE.
  - Solo operazioni: Costruttori, Produttori, Osservatori.

- **API**: Un modulo Signature:

```
module type APINAME =
  sig
    {type type-expression}^1
    {val operation-signature}^1
  end
```

- **ADT**: Un modulo Struct:

```
module ADTName =
  struct
    {type type-definition}^1
    {let operation-definition}^1
  end: APINAME
```

Estensione e Riuso in Java

- **Tipo di Dato Astratto**
  Una Collezione di Valori definita dall'insieme delle sole operazioni che possono essere usate per introdurli e operare con essi.

- **Estensione e Riuso: Il Problema**
  - È nota l'API Java di un Tipo Astratto
    (i.e. definizione di un interfaccia Java per tale Tipo).
  - È dato il nome di un ADT per tale tipo
    (i.e. nome di una classe che implementa l'API di tale Tipo).
  - Vogliamo Estendere i valori:
    - aggiungendo ulteriori proprietà e/o operazioni

- **Estensione e Riuso: Il Procedimento**
  - Definiamo una nuova classe che estende l'ADT dato
  - Lo stato della nuova classe estende lo stato dell'ADT
    - ed è private (in tutti i nuovi componenti)
  - AF ed I: sono ridefinite in accordo all'estensioni
  - Operazioni dell'ADT: Ereditate o anche Overridden
  - Nuove Operazioni: Definite
    - Overriding e Nuove Operazioni definite con riuso
      (i.e. uso dei metodi super)

Gli Esercizi di Oggi: Modellare Variabili C

Esercizio (1)

Un `APIMuInt` *è un Tipo Astratto per valori MUTABLE della forma* `[k]` *con* `k` *un valore si tipo* `int`*. Su questi valori sono definite le operazioni* `add` *e* `mul` *con la segnatura data in* `APIMuInt`*.*
Ogni implementazione provvede a definire:

- *Un costruttore che crea e inizializza un APIMuInt ad un intero* `k` *dato come argomento ed assegnato ad una variabile* `n`*, fornisce un comportamento analogo ad una dichiarazione C:* `int n = k;`
- *metodo toString per la presentazione, in aggiunta agli additionals che ritiene opportuni.*

Nel Listing trovate un file `APIMuInt.java` *con la API, un file MainMI.java da completare ed un folder* `NewObjets` *contenente il codice oggetto di* `APIMuInt.java`*,* `APIMuInt.class`*, e il codice oggetto, MuInt1.class, di un'implementazione di cui si omette il sorgente.*
Si utilizzi questo materiale per:

1. *Completare il file MainMI.java con una sequenza di operazioni su valori* `APIMuInt` *che esprima il comportamento della sequenza C:*
   ```
   int n1 = 3; int n2 =- 12; int n3 = 7; n1 *= n2; n2 += n1; n3 *= n2;
   ```
   e che mostri i valori ottenuti per `n1, n2, n3`*.*
2. *Compilare ed eseguire il file risultante, mostrando l'effettiva soluzione.*

Gli Esercizi di Oggi: Un Testo Semplificato

Esercizio (2)

Un insieme finito è un tipo di dato strutturato, `'a set`*, della forma* {*x1,...,xn*}*, per* `n>=0`*, a componenti* `xi` *omogenei, e contenente al più un'occorrenza di uno stesso valore,* `xi=xj iff i=j`*.*
I valori `'a set` *sono valori IMMUTABLE, sui quali possiamo operare mediante le sole operazioni indicate nell'API* `SET`*, vedi file "PolySet.ml" nel Listing allegato.*
Si dia un ADT, con nome `Set1`*. Allo scopo, si fornisca:*

1. *Stato della Rappresentazione, AF ed I: Non si sovraccarichi la rappresentazione con duplicati di elementi;*
2. *Si completi il modulo Set1 e la sequenza di espressioni da valutare, indicate da "..." nel file* `PolySet.ml`*.*
3. *Si esegua il file* `PolySet.ml` *mostrando l'effettiva soluzione data nei punti precedenti. check*

Esercizio (3)

Sia `APISet<A>` *l'API java per un Tipo Astratto di insiemi finiti, come in esercizio1 ma MUTABLE. Si assuma già disponibile un ADT* `Set1<A>` *per tale API. Il folder* `ApiSet` *nel listing allegato, contiene i codici oggetto di* `APISet<A>` *e di* `Set1<A>`*, da utilizzare nella compilazione delle classi da definire sotto. Si chiede:*

1. *Mostrare la definizione di* `APISet<A>`*, completando l'allegato file* `APISet.java` *(non compilare: check);*
2. *Estendere, in una nuova classe* `SetIS<A>`*, con il metodo* `is`*, i valori di* `APISet<A>`*, implementati in* `Set1`*. Il metodo* `is`*, applicato ad un oggetto di tipo* `A`*, calcola true se tale oggetto è contenuto in this. Calcola false, altrimenti.*
   Si fornisca la `SetIS<A>` *completando e compilando il file* `SetIS.java` *nel Listing allegato check.*
3. *Si completi, compili ed esegua il file* `MainSet.java` *con un caso di uso della soluzione data nel punto precedente.*

Gli Esercizi di Oggi: Modellare Variabili C

Esercizio (4)

Si riprenda l'esercizio1.

1. *Si fornisca una file* `MuInt2.java` *con package* `NewObjects` *che fornisca una seconda implementazione per* `APIMuInt.java`.
2. *Si discuta come la nuova implementazione può essere usata al posto di* `MuInt1` *per risolvere l'esercizio1.*
3. *Si compili e si esegua in modo da mostrare effettivamente i risultati trattati nei 2 punti precedenti.*