Correzione Compito Primo Appello del 16/6/14

Ivano Salvo
Sapienza Università di Roma
email: salvo@di.uniroma1.it

Anno Accademico 2013-14

Esercizio 1.1

Questo esercizio era molto insidioso perchè occorreva capire bene due cose: la scelta del metodo (inteso come nome+tipo di parametri e tipo di ritorno) avviene *staticamente* sulla base dei tipi delle variabili. La scelta di una particolare *implementazione* del metodo scelto (possono convivere più di una implementazione a causa di *overriding*) avviene *dinamicamente* sulla base di chi sia l'oggetto ricevente.

Detto questo, `ic` è di tipo statico `I` e poi punterà ad un oggetto di tipo `C`. Viceversa `cc` è di tipo statico `C`. Di conseguenza, le chiamate `ic.m(ic)` e `ic.m(cc)` chiameranno invariabilmente il metodo `m(I x)`, perchè quello è l'*unico* metodo `m` previsto nell'interfaccia del tipo `I`. Analogamente le invocazioni di `n` su `ic` chiameranno sempre e solo il metodo `n(I x, I y)`. Poi la `toString()` causerà *sempre* la stampa di `C`, perchè, dinamicamente, gli oggetti sono tutti di tipo `C`. E questo, grossomodo risolve i primi sei punti.

La chiamata di `cc.m(ic)` chiameranno ancora il metodo `m(I x)` perchè il tipo statico di `ic` è `I`, mentre la chiamata di `cc.m(cc)` causerà l'esecuzione del metodo `m(C x)`.

La chiamata `cc.n(ic,ic)` eseguirà `n(I x, I y)`, la chiamata `cc.n(ic,cc)` eseguirà `n(I x, I y)` (qui per *best match*, visto che non esiste un metodo `n(I x, C y)`), la chiamata `cc.m(cc,ic)` eseguirà `n(C x, I y)` e infine la chiamata `cc.m(cc,cc)` eseguirà `n(C x, C y)`. Riassumendo, gli output sono i seguenti:

```
m(I:C)
m(I:C)
n(I:C, I:C)
n(I:C, I:C)
n(I:C, I:C)
n(I:C, I:C)
m(I:C)
m(C:C)
```

```
n(I:C, I:C)
n(I:C, I:C)
n(C:C, I:C)
n(C:C, C:C)
```

Osservazioni ed Errori Tipici

Che dire, si è visto un po' di tutto. Le due cose più "intollerabili" sono state: 1) vedere stampe di `I` a destra di `:` (per esempio: `m(I:I)`), soprattutto perchè nessuna `toString()` restituisce la stringa `I`; 2) vedere nelle stampe cose come `ic` e `cc` che manifestano di non aver capito che quando un oggetto viene convertito in stringa, viene restituito l'output di `toString()`.

Esercizio 1.2

Sarebbe stato bello vedere qualche discussione accompagnare la soluzione di questo esercizio. Purtroppo non se ne sono viste.

La soluzione di gran lunga più gettonata è stata quella di ordinare il vettore e poi creare un nuovo vettore in cui caricare i primi k elementi del vettore ordinato. Si tratta di una soluzione corretta e relativamente efficiente, a patto, ovviamente che l'ordinamento sia eseguito con un metodo ottimo (di complessità $n \log n$). Certo che se k è molto minore di n, conviene calcolare k minimi successivi (avendo cura di non riconsiderare i minimi già calcolati): questo costa kn che è meglio di $n \log n$ se $k < \log n$.

Esistono per la verità metodi ancora migliori di estrarre k minimi, ma coinvolgono strutture dati (una forma particolare di albero binario, detto heap) che probabilmente non conoscete. E questo, ovviamente, non era preteso.

Osservazioni ed Errori Tipici

Molti programmi partivano con idee molto buone, ma non sempre l'implementazione era all'altezza. È da osservare che chi riordinava il vettore (senza appoggiarsi a qualche collezione) *modifica* il vettore di ingresso, e questo, quanto meno andava dichiarato nelle specifiche con la clausola MODIFIES.

Analogamente, è scorretto, ad esempio, ritornare il vettore `v` stesso nel caso in cui `k=v.length`. Infatti, si crea un pericoloso *alias* tra due strutture dati. Il vettore tornato, in ogni caso, andava creato ex-novo.

Le precondizioni sono spesso state ignorate, ma `0<=k<=v.length` a me pare una precondizione ragionevole.

Infine, uno sparuto gruppo di studenti non ha capito il problema ed ha restituito un vettore con tutti gli elementi di `v` minori di `k`.

Esercizio 1.3

La soluzione di questo esercizio era esattamente uguale a quella dell'analogo esercizio dell'esonero (direi un piccolo regalo ai reduci delusi dall'esonero). L'unica differenza è che nella variabile `d` piuttosto che i primi n numeri dispari venivano memorizzate le prime n potenze di due $1, 2, 4, 8, 16$, la cui somma è chiaramente $2^n - 1$ (è di fatto il numero rappresentato dalla stringa binaria di tutti 1). Cosa che ogni buon informatico, anche in erba, dovrebbe ormai sapere.

Esercizio 2.1

Qui occorreva capire che la `merge` generica sostanzialmente deve richiedere un tipo che *estende* `Comparable`. Ecco una elegante versione ricorsiva:

```
public static <A extends Comparable<A>> List<A> merge(List<A> l, List<A> m){
    if (l.isEmpty()) return m.copia();
    if (m.isEmpty()) return l.copia();
    if (l.head().compareTo(m.head())<0) return merge(l.tail(),m).add(l.head());
    return merge(l,m.tail()).add(m.head());
}
```

Notate bene quali siano i tipi corretti: si astrae sul tipo `A` dentro le parentesi amgolate, richiedendo che estenda `Comparable`: sarebbe come dire: "per tutti i tipi che sono sottotipo di `Comparable`".

Notate che, per mantenere l'immutabilità occorre, nei casi base, restituire una *copia* delle code e non semplicemente le code stesse.

Osservazioni ed Errori Tipici

Un errore tipico di molti è una volta verificato che la testa di una lista è minore dell'altra lista, inserire entrambi gli elementi (nel giusto ordine) e mandare avanti entrambe le liste. Questo non produce risultati corretti, perchè a volte ci sono intere sottosequenze minori in una delle due liste.

Esercizio 2.2

Senza entrare in grandi dettagli, la mia soluzione preferita sarebbe stata avere una variabile di istanza di tipo `Vector` (o altra collezione) e usare le funzioni `top, pop, push` come interfacce per l'inserimento e estrazione di elementi dal `Vector`. Assolutamente sconsigliato l'uso di un array (ha dimensione fissa e ci sono anche problemi nel creare array generici – secondo me, una delle incongruenze dei generics in Java). Osservo anche, che questa soluzione, dal mio punto di vista, è migliore di quella usata nelle Collections di Java, dove gli `Stack` sono *sottotipi* di `Vector` e come tali si può accedere a oggetti di tipo `Stack` in modo incoerente rispetto alla disciplina *last-in-first-out* che caratterizza la struttura dati pila.