Metodologie di Programmazione
Secondo Esonero

docenti: Ivano Salvo, Flavio Chierichetti
Sapienza Università di Roma, 12 giugno 2014

Parte Prima

Le soluzioni corrette a questa parte garantiscono una comoda sufficienza.

Esercizio 1.1: Classi Generiche. Definire la classe `Coppia<A,B>` che memorizza coppie di elementi di due tipi diversi (qualsiasi). Definire il costruttore, un metodo `fst` e un metodo `snd` che restituiscono rispettivamente il primo e il secondo elemento. Ridefinire (facendo overriding del metodo definito in `Object`) il metodo `equals` tenendo presente che due coppie sono uguali se hanno ordinatamente gli elementi uguali. Ridefinire il metodo `toString` producendo in output una coppia nella usuale notazione matematica.

Ad esempio per la coppia che contiene la stringa `sono arrivato` e l'intero `1` la `toString` dovrà restituire la stringa `(Sono arrivato, 1)`.

Esercizio 1.2: Metodi Generici. Considerare il tipo di dato `Lista` generica vista a lezione. Scrivere un metodo generico `zip` che invocato su una `Lista<A>` con un paramentro di tipo `Lista<B>` restituisce una lista di oggetti di tipo `Lista<Coppia<A,B>>`.

Il metodo `zip` costruisce una *nuova* lista in cui il primo elemento contiene una coppia formata dal primo elemento della lista ricevente ed il primo elemento della lista parametro, il secondo elemento contiene una coppia formata dal secondo elemento della lista ricevente ed il secondo elemento della lista parametro, e così via...

Il metodo `zip` deve restituire un'eccezione se la lista ricevente e la lista parametro non hanno la stessa lunghezza.

Esercizio 1.3: Eccezioni. Considerare il seguente codice.

```
class E0 extends Exception{};

class En extends Exception{};

class Eg extends Exception{
    int x;
    public Eg(int y){x=y;}
    public String toString(){return ""+x;}
  }

public class TestEccezioni{

    static void f(int n)
                  throws E0,En{
        if (n==0) throw new E0();
        else throw new En();
    }

    static int g(int n, int x, int d)
                  throws Eg {
        try { f(n); }
           catch (En e) { g(n-1,x+d,d*2);}
           catch (E0 e) { throw new Eg(x);}
        return x;
    }

    public static void main(String[] args){
         try { g(4,0,1); }
             catch (Eg e){
                  System.out.println(e);
                  }
    }
 }
```

Rispondere alle seguenti domande, motivando **brevemente** le risposte:

1. Qual è l'output del programma?
2. Più in generale, qual è il risultato stampato da una una invocazione `g(n,0,1)` dentro il `try ... catch` nel `main`?
3. Cosa risponde il compilatore se viene tolta l'istruzione `return x` alla fine del metodo `g`?
4. Se il compilatore passasse il programma senza `return`, quale output si otterrebbe?

Parte Seconda

Esercizio 2.1: Iteratori. Arricchire l'implementazione delle liste generiche `List<A>` definendo un iteratore che restituisce gli elementi di una lista nello stesso ordine in cui sono memorizzati nella lista.

Usare tale iteratore per scrivere un metodo `zip` che si comporta come il metodo dell'esercizio 1.2, ma in questo caso è un metodo statico con due parametri.

★ **Esercizio 2.2: Strutture Dati Generiche.** Un albero binario può convenientemente rappresentare un espressione aritmetica, dove i nodi interni sono operazioni, mentre le foglie sono numeri o variabili. Progettare una struttura dati generica adatta a memorizzare una espressione aritmetica con variabili. Inserire nella struttura dati un oggetto di tipo `Lista<Coppie<String,Number>>` che assegna un valore a ciascuna variabile (rappresentata come stringa).

Scrivere un metodo `valuta` che calcola il valore dell'espressione. Sollevare opportune eccezioni nelle possibili anomalie che individuate (ad esempio che una foglia contenga una variabile non presente nella lista).