Linguaggi di Programmazione (III Anno), Settembre 1998 - OOP

Esercizio OOP.1 (8 punti) Si consideri il seguente programma Java:

```
class Zero {
  public int f (Zero zero) { return 0;}
}
class One extends Zero {
  public int m () { return 1;}
  public int f (One one) { return one.m();}
}
class Two extends One {
  public int m () { return 2;}
}
```

Si dica qual'è l'effetto (errore statico, errore dinamico, produzione di un valore e quale) della valutazione delle espressioni (1) – (5): seguenti casi:

```
Zero zero = new Two();
One one = new One();
Two two = new Two();
(1) print(zero.m());
(2) print(two.m());
(3) print(one.f(zero));
(4) print(one.f(two));
(5) print(((Two)one).m());
```

Esercizio OOP.2 (8 punti) Si implementino le sottoclassi *somma*, *identita* e *costante* della classe astratta *funzione* (funzione su interi). Si puo' usare una sintassi sia C++ che Java-like (ma si consiglia l'uso di Java). Per le parti di codice che sono indipendenti dagli aspetti object-oriented di tali linguaggi si consiglia di usare pseudo-istruzioni o una descrizione a parole.

```
abstract class funzione {
abstract public int val (int x);    valore nel punto x
}
class somma extends funzione {
... PARTE PRIVATE
public somma(funzione f, funzione g)       DA IMPLEMENTARE (COSTRUTTORE)
METODI DI funzione DA IMPLEMENTARE
}
class costante extends funzione {
... PARTE PRIVATE
public costante(int c)       DA IMPLEMENTARE (COSTRUTTORE)
METODI DI funzione DA IMPLEMENTARE
}
class identita extends funzione {
... PARTE PRIVATE
public identita()       DA IMPLEMENTARE (COSTRUTTORE)
METODI DI funzione DA IMPLEMENTARE
}
```

Linguaggi di Programmazione (III Anno), Settembre 1998 - FP

Esercizio FP.1 (8 punti) Si implementino le operazioni del tipo di dato `polinomio`, i cui elementi rappresentano i polinomi in x a coefficienti interi

```
type polinomio = int list;
fun somma (p,q) = somma dei polinomi p e q
fun costante c = polinomio costante p(x)=c
val identita = polinomio identita' p(x)=x
fun derivata p = il polinomio dp/dx (* un po' difficile *)
fun prodotto (p,q) = prodotto dei polinomi p e q (* difficile *)
```

Si supponga che la lista $[c_i | i \in n]$ rappresenti il polinomio $p(x) = \Sigma_{i \in n} c_i * x^i$. Si consiglia di seguire l'ordine indicato.

Esercizio FP.2 (8 punti) Dato il tipo "alberi binari", definito come segue

```
datatype 'a btree = empty | node of 'a * 'a btree * 'a btree;
```

si consideri la seguente funzione higher-order:

```
fun btree_rec f a empty = a
  | btree_rec f a (node(b,t1,t2)) = f(b, t1, t2, btree_rec f a t1,
btree_rec f a t2);
```

- Si indichi il tipo piu' generale di `btree_rec`.
- Si definiscano istanziazioni di `btree_rec` che calcolano:
  - il prodotto delle etichette dei nodi di un albero binario con etichette intere, sollevando l'eccezione `Zero` se almeno una etichetta e' 0, cioe' $prod : int\ btree \rightarrow int$
  - la lista dei sottoalberi propri, cioe' $sub_trees :' a\ btree \rightarrow ('a\ btree)\ list$, in particolare $sub_trees\ empty = nil$.

```
(* OOP.1 *)

class Zero {
public int f (Zero zero) { return 0;}
}

class One extends Zero {
public int m () { return 1;}
public int f (One one) { return one.m();}
}

class Two extends One {
public int m () { return 2;}
}

public class Settembre98 {
public static void main(String argv[]) {
funzione f = new costante (1);
System.out.println(f.val(0));
funzione g = new identita ();
System.out.println(g.val(0));
funzione h = new somma (f,g);
System.out.println(h.val(0));

Zero zero = new Two();
One one = new One();

//errore statico System.out.println(zero.m());
System.out.println(((One)zero).m());//scrive 2
System.out.println(one.f(zero)); // scrive 0
System.out.println(one.f((One)zero));//scrive 2
System.out.println(((Two)one).m());//errore dinamico
}
};


(* OOP.2 *)

abstract class funzione {
abstract public int val (int x);
}

class somma extends funzione {
private funzione f,g;
public somma (funzione f, funzione g) {
this.f = f; this.g = g;
}
public int val (int x) { return f.val(x) + g.val(x); }
}

class costante extends funzione {
private int c;
public costante (int c) { this.c = c; }
public int val (int x) { return c; }
```

```
}

class identita extends funzione {
public identita () {}
public int val (int x) { return x; }
}

(* FP.1 *)

type polinomio = int list;
fun somma (nil,p) = p
  | somma (p,nil) = p
  | somma (c1::p1,c2::p2):polinomio = (c1+c2)::somma(p1,p2);

local
    fun moltiplica c q = map (fn x:int=>c*x) q;
    fun shift q = 0::q;
in
    fun prodotto(nil,q) = nil
      | prodotto(c::p,q) = somma(moltiplica c q,prodotto(p,shift q));
end;
fun costante c = [c];
val identita = [0,1];
local
    fun d n nil = nil
      | d n (c::p) = (n*c)::d(n+1)p;
in
    fun derivata nil = nil
      | derivata (c::p) = d 1 p;
end;

(* FP.2 *)

datatype 'a btree = empty | node of 'a * 'a btree * 'a btree;
fun btree_rec f a empty = a
  | btree_rec f a (node(b,t1,t2)) = f(b, t1, t2, btree_rec f a t1, btree_rec f a t2);

exception Zero;
local
    fun f (n:int,_,_,n1,n2) = if n=0 then raise Zero else n*n1*n2;
in
    val prod = btree_rec f 1
end;
local
    fun f (_,t1,t2,l1,l2) = t1::t2::(l1@l2);
in
    val sub_trees = btree_rec f nil
end;
```