Paradigmi di Programmazione
I esonero: Gerarchia di alberi in Java

Stefano Guerrini

A.A. 2002/03 – Corso di Laurea in Informatica

Versione del 23 luglio 2003

Lo scopo del progetto è implementare una gerarchia di classi Java per l'implementazione di alberi.

Alla radice della gerarchia si troverà una classe per la rappresentazione di tutti i tipi di alberi. Si devono scrivere almeno due sottoclassi per rappresentare:

1. alberi binari.
2. alberi generici con un numero di figli variabile.

Per la rappresentazione degli alberi generici si consiglia di ricorrere ad una rappresentazione ricorsiva di tipo *alberi e foreste*. Si ricorda che una foresta è un insieme ordinato di alberi. Quindi, a partire da una foresta, un albero può essere costruito aggiungendo un nodo radice alla foresta, che diviene così la sequenza ordinata dei figli del nodo radice.

Per la visita degli alberi si deve utilizzare una opportuna implementazione della interfaccia `java.lang.Enumeration`. Come esempio si veda l'implementazione della classe `LinkedStack` riportata in appendice.

In pratica, si dovranno fornire opportune implemetazioni di tale classe per scrivere gli algoritmi di visita in preordine, in postordine e simmetrica (quando questa ha senso, ovvero per gli alberi binari).

Oltre ai costruttori, si dovranno scrivere alcune funzioni di base sugli alberi: conteggio dei nodi dell'albero, calcolo della profondità, conteggio delle foglie; massima cardinalità di un nodo (numero dei figli), etc. (altre funzioni di base possono essere aggiunte a discrezione dello studente).

Lo studente docrà cercare di implementare tali funzioni nel modo più generale possibile, ovvero, se possibile nella radice della gerachia, oppure nella sottoclasse più generale possibile.

Altre operazioni da implementare sono quelle che elimano o rimpiazzano sottoalberi.

Si raccomanda di scriver uno o più main che utilizzano e verificano le classi implementate.

Estensioni

Quanto sopra scritto è la parte obbligatoria del progetto. Facoltativamente, si possono scrivere una o più delle seguenti classi.

- Alberi binari di ricerca con le rispettive operazioni di ricerca ed inserimento.
- Espressioni aritmetiche. Supponendo di raprresentare le espressioni mediante alberi si scrivano le funzioni che calcolano il valore dell'espressione, che la leggono e stampano in forma polacca prefissa e postfissa o in forma infissa con le parentesi.
- Termini di un'algebra. Supponendo di avere dei simboli di funzione denotati da identificatori che cominciano con lettere minuscole e dei simboli di variabile denotati da identificatori che cominciano con lettera maiuscola, si possono costruire espressioni del tipo X, dove X è una variabile, oppure, $f(e_0, \ldots, e_k)$ dove f è un simbolo di funzione ed $e_0, \ldots, e_k$ sono espressioni. Si devono scrivere le funzioni di lettura e stampa delle espressioni.

A L'esempio della classe `LinkedStack`

```
public class LinkedStack {
    // Our static member interface; body omitted here...
    public static interface Linkable { ... }

    // The head of the list
    private Linkable head;

    // Method bodies omitted here
    public void push(Linkable node) { ... }
    public Linkable pop() { ... }

    // This method returns an Enumeration object for this LinkedStack
    public java.util.Enumeration enumerate() { return new Enumerator();
    }

    // Here is the implementation of the Enumeration interface,
    // defined as a member class.
    protected class Enumerator implements java.util.Enumeration {
        Linkable current;
        // The constructor uses the private head field of the containing class
        public Enumerator() { current = head; }
        public boolean hasMoreElements() {  return (current != null); }
        public Object nextElement() {
            if (current == null) throw new java.util.NoSuchElementException();
            Object value = current;
            current = current.getNext();
            return value;
        }
    }
}
```