LISTE di Interi

- null è la lista vuota
- se l è una lista e n è un numero intero allora (n::l) è una lista
- niente altro è una lista
- **L** denota l'insieme delle liste

Notazione

[] è la lista vuota null

$[n_1;...;n_k]$ è la lista $(n_1 :: ...(n_k :: \text{null})...)$

Le operazioni principali associate alle liste sono:

head: **L** - {null} → int
tail : **L** - {null} → **L**

head(n :: l) = n
tail(n :: l) = l

Una classe per le liste

```
public class L {
    private int h;
    private L t;
    public L(int h, L t) {
        this.h = h;
        this.t = t;
    }

    public int head() {
        return h;
    }

    public L tail() {
        return t;
    }

    public String toString(){
        return "[" + convert() + "]";
    }
    private String convert(){
        if (t == null) return "" + h;
        else return h + ":" + t.convert();
    }

    public L clona() {
        if (t == null) return new L(h,null);
        else return new L(h,t.clona());
    }
```

h: head, t: tail

qualche metodo di istanza

```
// concatena questa lista con other e restituisce il risultato
  public L append(L other) {
        if (t == null) return new L(h,other);
        else return new L(h,t.append(other));
  }
  // inverte questa lista e restituisce il risultato
  public L reverse() {
        if (t == null) //return this;
          return new L(h,null);
        else return t.reverse().append(new L(h,null));
  }
  // inverte questa lista e restituisce il risultato, senza usare append()
  public L reverse_acc() {
        return reverse_acc(null);
  }
  private L reverse_acc(L acc) {
        if (t == null) return new L(h,acc);
        else return t.reverse_acc(new L(h,acc));
  }
  // prende un elemento no e uno si' da una lista
  // e restituisce il risultato
  public L alternate() {
        if (t == null || t.t == null) return new L(h,null);
        else return new L(h,t.t.alternate());
  }
// estrae  gli elementi pari
public L pari() {
    if (t== null){
        if ((h / 2) * 2 == h) {
            return  new L (h,null);
        }
        else return null;
    }
    else if ((h / 2) * 2 == h) {
        return new L(h, t.pari());
    }
    else return t.pari();
```

Alberi Binari etichettati con Interi (visti come espressioni)

- null è l'albero binario vuoto
- se t_1 e t_2 sono due alberi binari e n è un intero allora Tree(n,t_1,t_2) è un albero binario (n è l'etichetta della radice, t_1 è detto sottoalbero sinistro e t_2 è detto sottoalbero destro)
- niente altro è un albero binario

T denota l'insieme degli alberi binari etichettati con interi

In un albero t diverso da null lo stesso numero (etichetta) n può occorrere più volte.
Ad esempio in Tree(2,Tree(3,null),Tree(2,null))
abbiamo due occorrenze dell'etichetta 2
Tree(2,Tree(3,null),Tree(2,null))

Per tale motivo parleremo di "occorrenze" delle etichette.
Le occorrenze delle etichette in t sono dette nodi.
L'occorrenza dell'etichetta della radice è detta semplicemente radice.
Una nodo che ha null come sottoalbero sinistro e destro è detta foglia.

Le operazioni principali associate agli alberi sono:

root: **T** -{null} → int

left : **T**-{null} → **T**

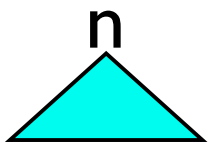
right: **T**-{null} → **T**

root(Tree(n,t_1,t_2))= n

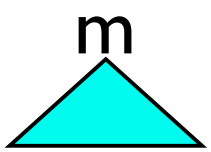
left (Tree(n,t_1,t_2)) = t_1

right (Tree(n,t_1,t_2)) = t_2

Spesso gli alberi binari sono rappresentati in forma grafica. null è rappresentato con un "disegno vuoto"
Se t_1 (con radice n) è rappresentato graficamente da



e t_2 (con radice m) è rappresentato da



e r è un numero allora le rappresentazioni di
Tree(r,null,null), Tree(r,t_1,null), Tree(r,null,t_2), Tree(r,t_1,t_2)
sono rispettivamente