Alberi Binario in Java

Realizzare un albero binario di ricerca.

L'albero binario è di **ricerca** se esiste una relazione di ordinamento tra i valori dei nodi (valori comparabili). In particolare, dato un nodo, il sottoalbero sinistro ha nodi i cui valori sono più piccoli di quello del nodo d'origine, mentre il sottoalbero destro ha nodi con valori più grandi

Deve essere possibile inserire, cancellare e cercare valori.

Modellazione: quali entità?

1. **Albero**
2. **Nodo**

Nodo

```
public class Nodo {
private MioOggetto value;
private Nodo left;   // figlio sinistro
private Nodo right; // figlio destro

public Nodo(MioOggetto valore) {
     value = valore;
     left = null;
     right = null;
}

/**
 * Permette di leggere il valore associato al nodo
 * @return un intero corrispondente al valore del nodo
 */
public MioOggetto getValue() { return value; }
```

Nodo

```
public void setLeftChild(MioOggetto child) { left = child; }

public void setRightChild(MioOggetto child) { right = child; }

public MioOggetto getLeftChild() { return left; }

public MioOggetto getRightChild() { return right; }
```

Nota: nell'ottica di implementare un albero binario di ricerca, è necessario poter stabilire una **relazione d'ordine** tra il valore degli oggetti associati ai nodi. In altre parole, le istanze di MioOggetto devono essere **comparabili**

L’interfaccia Comparable

Tra le diverse interfacce incluse nel Java Development Kit, l’interfaccia Comparable (package java.lang) identifica gli oggetti per i quali è possibile definire una relazione d’ordine.
Tra le classi del JDK che implementano tale interfaccia, si segnalano le classi wrapper (Integer, Double, Char, …), e le classi String e File.

Comparable richiede l’implementazione di un unico metodo, compareTo, la cui signature è la seguente:

```
int compareTo(Object o)
```

Il metodo deve ritornare un valore

- negativo, se l’oggetto (this) è minore del parametro “o”
- nullo, se l’oggetto è uguale al parametro “o”
- positivo, se l’oggetto è uguale al parametro “o”

Nel nostro esempio, la classe MioOggetto dovrà fornire una implementazione di tale metodo per consentire l’ordinamento delle istanze

Albero Binario

```
public class AlberoBinario {
 private Nodo root;

 public AlberoBinario() {
       root = null;
 }
```

Le funzionalità da implementare sono così dichiarabili:

```
 public void inserisciValore(MioOggetto valore) {…}

 public boolean ricercaValore(MioOggetto valore) {…}

 public void eliminaValore(MioOggetto valore) throws TreeException {…}
```

Inserisci valore – soluzione iterativa

```
public void inserisciValore(MioOggetto valore) {
    Nodo nodoCorrente = root;
    while (nodoCorrente != null) {
        if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
            nodoCorrente = nodoCorrente.getLeftChild();
        else {
            if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
                nodoCorrente = nodoCorrente.getRightChild();
            else
                return;
        }
    }
    nodoCorrente = new Nodo(valore);
}
```

Inserisci valore – soluzione ricorsiva

```
public void inserisciValore(MioOggetto valore) {
    insert(valore, root);
}

protected void insert(MioOggetto valore, Nodo nodoCorrente) {
    if (nodoCorrente == null) {
        nodoCorrente = new Nodo(valore);
    }
    else {
        if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
            insert(valore, nodoCorrente.getLeftChild());
        else {
            if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
                insert(valore, nodoCorrente.getRightChild());
        }
    }
}
```

Ricerca valore – soluzione iterativa

```
public boolean ricercaValore(MioOggetto valore) {
   Nodo nodoCorrente = root;
   boolean trovato = false;
   while ((nodoCorrente != null) && (!trovato)) {
      if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
         nodoCorrente = nodoCorrente.getLeftChild();
      else {
         if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
            nodoCorrente = nodoCorrente.getRightChild();
         else
            trovato = true;
      }
   }
   return trovato;
}
```

Ricerca valore – soluzione ricorsiva

```
public boolean ricercaValore(MioOggetto valore) {
    return search(valore, root);
}

protected boolean search(MioOggetto valore, Nodo nodoCorrente) {
    if (nodoCorrente == null) {
        return false;
    }

    // else opzionale
    if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
        return search(valore, nodoCorrente.getLeftChild());
    else {
        if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
                return search(valore, nodoCorrente.getRightChild());
        else
                return true;
    }
}
```

Elimina valore – soluzione iterativa

```
/**
 * Cancella un valore dall'albero
 * @param valore il valore da cancellare
 * @throws TreeException se il valore non è presente nell'albero
 */
public void eliminaValore(MioOggetto valore) throws TreeException {
    if (root == null) throw new TreeException("Albero Nullo");
    if (valore.compareTo(root.getValue()) == 0) {
        insertTree(root.getRightChild(), root.getLeftChild());
        return;
    }    <continua>
```

```
private void insertTree(Nodo treeRoot, Nodo addedTreeRoot) {
    Nodo parent = null;
    while (treeRoot != null) {
            parent = treeRoot; treeRoot = treeRoot.getLeftChild();
    }
    if (parent != null) parent.setLeftChild(addedTreeRoot);
}
```

Elimina valore – soluzione iterativa

```
// Cerca il nodo
Nodo nodoCorrente = root, parent = null;
int parentLink = 0; // vale -1 se percorro il ramo di sx, +1 se percorro quello di dx

while ( (nodoCorrente != null) && // Attenzione! Shortcut
     (valore.compareTo(nodoCorrente.getValue())!=0) ) {
          parent = nodoCorrente;
          if (valore.compareTo(nodoCorrente.getValue()) < 0) {
               nodoCorrente = nodoCorrente.getLeftChild();
               parentLink = -1;
          }
          else {
               nodoCorrente = nodoCorrente.getRightChild();
               parentLink = 1;
          }
}
if (nodoCorrente == null) throw new TreeException("Valore inesistente");
<continua>
```

Elimina valore – soluzione iterativa

```
if (nodoCorrente.getRightChild() == null) {
    if (parentLink == -1)
        parent.setLeftChild(nodoCorrente.getLeftChild());
    else
        parent.setRightChild(nodoCorrente.getLeftChild());
}
else {
    if (parentLink == -1)
        parent.setLeftChild(temp);
    else
        parent.setRightChild(temp);
    insertTree( nodoCorrente.getRightChild(), nodoCorrente.getLeftChild() );
}
```

Elimina valore – soluzione ricorsiva

```
public void cancellaValore(MioOggetto valore) throws TreeException {
    if (valore.compareTo(root.getValue()) == 0 )
        insertTree(root.getRightChild(), root.getLeftChild());
    else {
        if ( valore.compareTo(root.getValue()) < 0 )
            delete(valore, root, root.getLeftChild());
        else
            delete(valore, root, root.getRightChild());
    }
}

protected void delete(MioOggetto valore, Nodo parent, Nodo
nodoCorrente)
    throws TreeException {
        if (nodoCorrente == null)
            throw new TreeException("Valore inesistente"); < continua >
```

Elimina valore – soluzione ricorsiva

```
if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
   delete(valore, nodoCorrente, nodoCorrente.getLeftChild());
else {
   if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
      delete(valore, nodoCorrente, nodoCorrente.getRightChild());
   else { // nodoCorrente è il nodo che voglio cancellare
      Nodo temp;
      if ( nodoCorrente.getRightChild() == null )
         temp = nodoCorrente.getLeftChild();
      else {
         temp = nodoCorrente.getRightChild();
         insertTree( nodoCorrente.getRightChild(), nodoCorrente.getLeftChild() );
      }
      if (parent.getLeftChild() == nodoCorrente)
         parent.setLeftChild(temp);
      else
         parent.setRightChild(temp);
   }
}
```

Nodo – una diversa rappresentazione

```
public class Nodo2 {
protected Nodo2[] child;
protected MioOggetto value;

public Nodo(MioOggetto valore) {
    value = valore;
    child = new Nodo[2];
    child[0] = null; // figlio sinistro
    child[1] = null; // figlio destro
}

/**
 * Permette di leggere il valore associato al nodo
 * @return un intero corrispondente al valore del nodo
 */
public int getValue() { return value; }
```

Nodo2

```
public static final short SX_CHILD = 0;
public static final short DX_CHILD = 1;
/**
 * Restituisce uno dei nodi figli del nodo
 * @param selezione vale Nodo.SX_CHILD per il nodo di sinistra, Nodo.DX_CHILD per il
            nodo di destra
 * @return il riferimento al nodo figlio selezionato
 */
public Nodo2 getChild(short selezione) { return child[selezione]; }

/**
 * Inserisce uno specifico nodo come figlio dell'istanza di Nodo corrente
 * @param selezione vale Nodo.SX_CHILD se il figlio è a sinistra, Nodo.DX_CHILD se il
            figlio è a destra
 */
public void setChild(Nodo2 n, short selezione) { child[selezione] = n; }
```

Nota: non c'è controllo sul parametro selezione perchè si suppone l'uso di una delle costanti definite; in teoria è possibile provocare la generazione di un'eccezione non controllata ArrayIndexOutOfBoundsException

Inserisci valore – soluzione ricorsiva

```
public void inserisciValore(MioOggetto valore) {
    insert(valore, root);
}

protected void insert(MioOggetto valore, Nodo2 nodoCorrente) {
    if (nodoCorrente == null) {
        nodoCorrente = new Nodo2(valore);
    }
    else {
        if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
            insert(valore, nodoCorrente.getChild(Nodo2.SX_CHILD));
        else {
            if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
                insert(valore, nodoCorrente.getChild(Nodo2.DX_CHILD));
        }
    }
}
```

Ricerca valore – soluzione ricorsiva

```
public boolean ricercaValore(MioOggetto valore) {
    return search(valore, root);
}

protected boolean search(MioOggetto valore, Nodo2 nodoCorrente) {
    if (nodoCorrente == null) {
        return false;
    }

    // else opzionale
    if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
        return search(valore, nodoCorrente.getChild(Nodo2.SX_CHILD));
    else {
        if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
            return search(valore, nodoCorrente.getChild(Nodo2.DX_CHILD));
        else
            return true;
    }
}
```

Elimina valore – soluzione ricorsiva

```
public void cancellaValore(MioOggetto valore) throws TreeException {
if (valore.compareTo(root.getValue()) == 0 )
   insertTree(root.getChild(Nodo2.DX_CHILD),root.getChild(Nodo2.SX_CHILD));
else {
   if ( valore.compareTo(root.getValue()) < 0 )
      delete(valore, root, root.getChild(Nodo2.SX_CHILD));
   else
      delete(valore, root, root.getChild(Nodo2.DX_CHILD));
}
}
```

Elimina valore – soluzione ricorsiva

```
protected void delete(MioOggetto valore, Nodo2 parent, Nodo2 nodoCorrente)
   throws TreeException {
if (nodoCorrente == null) throw new TreeException("Valore inesistente");

if ( valore.compareTo(nodoCorrente.getValue()) < 0 )
   delete(valore, nodoCorrente, nodoCorrente.getChild(Nodo2.SX_CHILD));
else {
   if ( valore.compareTo(nodoCorrente.getValue()) > 0 )
      delete(valore, nodoCorrente, nodoCorrente.getChild(Nodo2.DX_CHILD));
   else { // nodoCorrente è il nodo che voglio cancellare
      short parentLink;
      if (parent.getChild(Nodo2.SX_CHILD) == nodoCorrente )
         parentLink = Nodo2.SX_CHILD;
      else
         parentLink = Nodo2.DX_CHILD; < continua >
```

Elimina valore – soluzione ricorsiva

```
if ( nodoCorrente.getChild(Nodo2.DX_CHILD) == null )
   parent.setChild( nodoCorrente.getChild(Nodo2.SX_CHILD), parentLink );
else {
   parent.setChild( nodoCorrente.getChild(Nodo2.DX_CHILD), parentLink );

   // per evitare una chiamata a vuoto
   if ( nodoCorrente.getChild(Nodo2.SX_CHILD) != null )
      insertTree( nodoCorrente.getChild(Nodo2.DX_CHILD),
   }
}

// Inserimento del sottoalbero a sinistra
private void insertTree(Nodo2 treeRoot, Nodo2 addedTreeRoot) {
   Nodo2 parent = null;
   while (treeRoot != null) {
      parent = treeRoot;
      treeRoot = treeRoot.getChild(Nodo2.SX_CHILD);
   }
   if (parent != null) parent.setChild(addedTreeRoot,Nodo2.SX_CHILD);
}
```

Eclipse Tips

Eclipse ha parecchi strumenti per automatizzare alcune attività di routine nella scrittura del codice o per semplificarne la gestione.
In particolare, il menu **“Source”** elenca numerose funzioni di utilità relative alla **creazione dei sorgenti**.
Si segnalano:

1. **Format**: formattazione automatica (e ragionevolmente corretta) del codice [tasto di scelta rapida CRTL + SHIFT + F]
2. **Generate Getter and Setter**: generazione selezione dei metodi di get e set una volta definiti gli attributi di una classe
3. **Surround with try/catch Block**: racchiude la porzione di codice selezionata in un blocco try catch

Attraverso il menu **“Refactor”** si accede a strumenti di modifica dei sorgenti. Ad esempio, lo strumento di **rename** permette di rinominare classi/attributi/ecc. **propagando le modifiche** a tutte le parti del progetto interessate