Alberi Binari 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 CLASSI?

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) {
//costruttore per instanziare un nodo che contenga una nuova informazione
     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; }
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;
 }
```

I metodi 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 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 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 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);
   }
}
```