Non-Binary Trees:

class Node {
    ElementType data;
    Node parent, leftmostChild, rightSibling;
    // singly-linked list of each node’s children
}
class Tree {
    Node root;
}

Node:

<table>
<thead>
<tr>
<th>data</th>
<th>parent</th>
<th>leftC</th>
<th>rightS</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node:

<table>
<thead>
<tr>
<th>data</th>
<th>parent</th>
<th>leftC</th>
<th>rightS</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alternative representation of Non-Binary Trees:

class Node {
    ElementType data;
    Node parent, leftmostChild, rightmostChild,
        leftSibling, rightSibling;
    // doubly-linked list of each node's children
}
class Tree {
    Node root;
}

Applications:

Contents of a book
Traversals of a non-binary tree:

Preorder traversal

```java
void preorder (Tree T) {
    preorder (T.root);
}
void preorder (Node p) {
    if (p==null) return;
    visit (p);
    preorder (p.leftmostChild);
    preorder (p.rightSibling);
}
```

A B C D E F G H J
Postorder traversal

```c
void postorder (Tree T) {
    postorder (T.root);
}
void postorder (Node p) {
    if (p==null) return;
    postorder (p.leftmostChild);
    visit (p);
    postorder (p.rightSibling);
}
```

Inorder traversal

```
No natural definition of inorder for non-binary trees
```
Level-order traversal

```java
void levelOrder (Tree T) {
    Queue Q( );
    Q.enqueue (T.root);
    while (not Q.isEmpty( )) {
        Node p = Q.dequeue( );
        visit (p);
        for (Node c=p.leftmostChild;  c!=null;  c=c.rightSibling)
            Q.enqueue (c);
    }
}
```

Analysis: Let \( n \) = number of nodes in the tree. Each kind of traversal spends \( \Theta(1) \) time at each node of the tree, so each traversal has \( \Theta(n) \) total running time. [same as for binary trees]