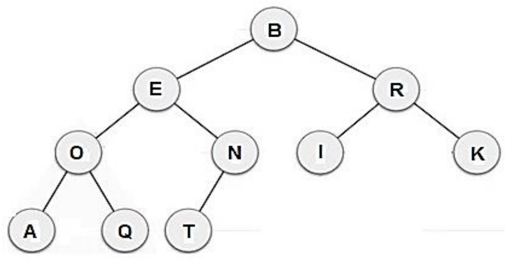
Trees

Fundamentals

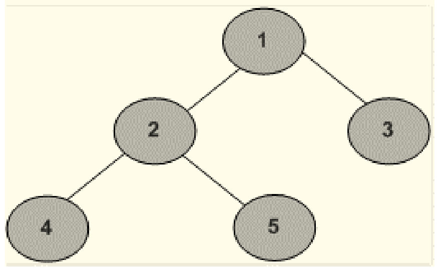
- A **tree** represents a hierarchical nature of a structure in a graphical form. It consists of elements or **nodes**, with each node linked to its successors.
- The best example of a tree is the computer's file system: **C:\Users\bpena\Desktop\TreeDemo.java**
- The top of a tree is called its **root**. The links from a node to its successors are called **branches**, **edges**, **lines**, or **paths**.
- The successors of a node are called its **child nodes** or children. The predecessor of a node is called its **parent**.
- A tree is considered a **binary tree** if all its nodes have two (2) child nodes at most.
- Each node in a tree has exactly one (1) parent except for the root node, which has no parent.
- Nodes that have the same parent are **siblings**.
- A node that has no child nodes is a **leaf node** or **external node**. Nodes that have children are known as **internal nodes**.
- A tree within a tree is considered a **subtree**.
- The **level** of a node is a measure of its distance from the root.
- The **depth** of the tree is its highest level.
- The **degree** is the number of child nodes in a subtree.
- *Example:*



Parts	**Value(s)**
Root node	B
Child nodes	E, R, O, N, I, K, A, Q, T
Parent nodes	B, E, R, O, N
Siblings	E and R O and N I and K A and Q
Leaf nodes	I, K, A, Q, T
One-level subtrees	B-E/R E-O/N R-I/K O-A/Q N-T
Nodes per level	Level 0 – B Level 1 – E, R Level 2 – O, N, I, K Level 3 – A, Q, T
Depth	3
Degree of each one-level subtree	Subtree B – 2 Subtree E – 2 Subtree R – 2 Subtree O – 2 Subtree N – 1

Tree Traversal

- **Traversal** is the process of visiting all the nodes in a specific order. The following are the different traversal types:



  - **Breadth-First or Level Order**: Nodes are visited by level. 1, 2, 3, 4, 5
  - **Depth-First**
    - **Inorder** (Left, Root, Right): 4, 2, 5, 1, 3
      Start with the bottommost left subtree. Once the root in Level 0 is visited, proceed to the bottommost right subtree.

- **Preorder** (Root, Left, Right): 1, 2, 4, 5, 3
  Start with the root in Level 0 then continue with the left subtree.
- **Postorder** (Left, Right, Root): 4, 5, 2, 3, 1
  Start with the bottommost left subtree then proceed to the other subtrees. The root in Level 0 is the last node visited.

Programming Trees

- The **`JTree`** is a Java Swing component that displays a set of hierarchical data as an outline. It is included in the **javax.swing** package.
- The Java class, **`DefaultMutableTreeNode`**, is used to represent a general-purpose node in a tree data structure. It is included in the **javax.swing.tree** package.
- The **`add()`** method removes a node from its parent and makes it a child of another node by adding it to the end of that node's child array.
- Other Java methods used in retrieving values from a tree are:

Method	**Description**
`getRoot()`	Returns the root of the tree that contains the node
`children()`	Creates and returns a forward-order enumeration of a node's children
`getChildCount()`	Return the number of children that a node has
`getParent()`	Returns a node's parent or `null` if it has no parent
`isNodeSibling()`	Returns `true` if a node is a sibling of the other node
`getPreviousSibling()`	Returns the previous sibling of a node in the parent's children array
`getNextSibling()`	Returns the next sibling of a node in the parent's children array
`getSiblingCount()`	Returns the number of siblings of a node
`isLeaf()`	Returns `true` if a node has no children
`getLeafCount()`	Returns the total number of leaves that are descendants of a node
`getLevel()`	Returns the number of levels above a node
`getDepth()`	Returns the highest level of the tree
`getChildCount()`	Returns the number of children (degree) of a node
`breadthFirstEnumeration()`	Creates and returns an enumeration that traverses the subtree rooted at a node in breadth-first order.
`preorderEnumeration()`	Creates and returns an enumeration that traverses the subtree rooted at a node in preorder
`postorderEnumeration()`	Creates and returns an enumeration that traverses the subtree rooted at this a in postorder.

Sample Codes:

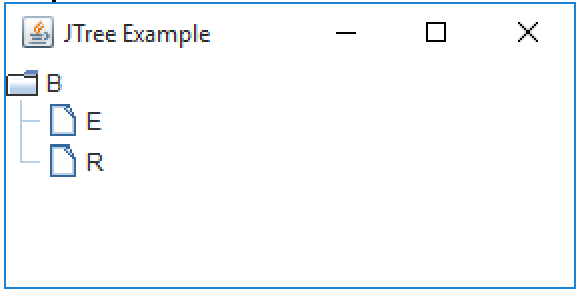
1. To create an empty tree
   ```
   JTree tree = new JTree();
   ```
2. To create a node:
   ```
   DefaultMutableTreeNode root = new
   DefaultMutableTreeNode("B");
   ```

3. To create a tree with identified root node:
   `JTree tree = new JTree(root);`
4. To add a child node
   `root.add(n1);` `//n1 is another node`
5. To display the children of a node:
```
//Convert the Enumeration type to List
List childNodes = Collections.list(root.children());
System.out.print(childNodes);
//or simply
System.out.print(Collections.list(root.children()));
```
6. To display a traversed tree:
```
//Convert the Enumeration type to List
List preTree =
Collections.list(root.preorderEnumeration());
System.out.print(preTree);
```

Sample Program to Display a `JTree` in a `JFrame`:

```
public class TreeSample extends JFrame {
    JTree tree;
    public TreeSample()
    {
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("B");
        DefaultMutableTreeNode n1 = new DefaultMutableTreeNode("E");
        DefaultMutableTreeNode n2 = new DefaultMutableTreeNode("R");
        root.add(n1);
        root.add(n2);
        tree = new JTree(root);
        add(tree);
        this.setTitle("JTree Example");
        this.setSize(300,300);
        this.setVisible(true);
    }
    public static void main(String[] args)
    {
        new TreeSample();
    }
}
```

Output:



References:
Koffman, E. and Wolfgang, P. (2016). *Data structures: Abstraction and design using Java.* Hoboken: John Wiley & Sons, Inc.
Oracle Docs (n.d.). *Citing sources.* Retrieved from https://docs.oracle.com/javase/7/docs/api/javax/swing/tree/DefaultMutableTreeNode.html