17. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E>* Traversals

Continuing with the *BinaryTree<E>* traversal methods:

`#traverse(pWhich: int, pRoot: Node<E>, pVisitor: BinaryTreeVisitor<E>): void`

Performs the type of traversal specified by *pWhich* starting at the subtree rooted by *pRoot*. If *pRoot* is the root *Node* of a *BinaryTree* then we traverse the entire tree. Otherwise, if *pRoot* is the root *Node* of a subtree, then we traverse only the subtree.

```
protected void traverse(int pWhich, Node<E> pRoot, BinaryTreeVisitor<E> pVisitor) {
  if (pRoot == null) return;
  switch (pWhich) {
    case INORDER:
      traverse(pWhich, pRoot.getLeft(), pVisitor);  // Traverse left subtree of pRoot
      pVisitor.visit(pRoot.getData());              // Visit pRoot
      traverse(pWhich, pRoot.getRight(), pVisitor); // Traverse right subtree of pRoot
      break;
    case POSTORDER:
      traverse(pWhich, pRoot.getLeft(), pVisitor);  // Traverse left subtree of pRoot
      traverse(pWhich, pRoot.getRight(), pVisitor); // Traverse right subtree of pRoot
      pVisitor.visit(pRoot.getData());              // Visit pRoot
      break;
    case PREORDER:
      pVisitor.visit(pRoot.getData());              // Visit pRoot
      traverse(pWhich, pRoot.getLeft(), pVisitor);  // Traverse left subtree of pRoot
      traverse(pWhich, pRoot.getRight(), pVisitor); // Traverse right subtree of pRoot
    break;
}
```

17. Trees :: Binary Trees :: Java Implementation :: *BinaryTree<E>* Traversals

`#traverseLevelOrder(pRoot: Node<E>, BinaryTreeVisitor<E> pVisitor): void`

Performs a level order traversal of the tree rooted at *pRoot*.

In *Trees : Section 4* we discussed the pseudocode for a level order traversal and saw that it used a queue to store the *Node*s to be visited.

By modifying both the *DList* and *Queue* classes to generify them (i.e., specify the type of each element by a type parameter *E*), then we use the *Queue* class we discussed earlier to store the *Nodes*. Note that type of the elements added to the *Queue* (and subsequently to the *DList<E>* since *Queue<E>* extends *DList<E>*) is a *Node<E>*:

```
protected void traverseLevelOrder(Node<E> pRoot, BinaryTreeVisitor<E> pVisitor) {
  Queue<Node<E>> nodeQueue = new Queue();
  nodeQueue.enqueue(pRoot);
  while (!nodeQueue.isEmpty()) {
    Node<E> root = nodeQueue.dequeue();
    pVisitor.visit(root.getData());
    if (root.hasLeft()) nodeQueue.enqueue(root.getLeft());
    if (root.hasRight()) nodeQueue.enqueue(root.getRight());
  }
}
```