Lab 12: Binary Trees – Two by Two

Purpose

This lab explores the design and implementation of an array implementation of a Binary Tree. At the completion of the lab you will have:

1. Created a binary tree package.
2. Implemented a binary tree using a linked structure.
3. Printed the binary tree using an in-order traversal.

Creating an Array Implementation of a Binary Tree

Do the following:

1. Clone the lab repo.
2. The design of the BianryTreeADT<T> is given Figure 1. The code for the interface is complete in the lab repo. Your task is to clean up the documentation for each method. ***Do this before moving to the next step.***

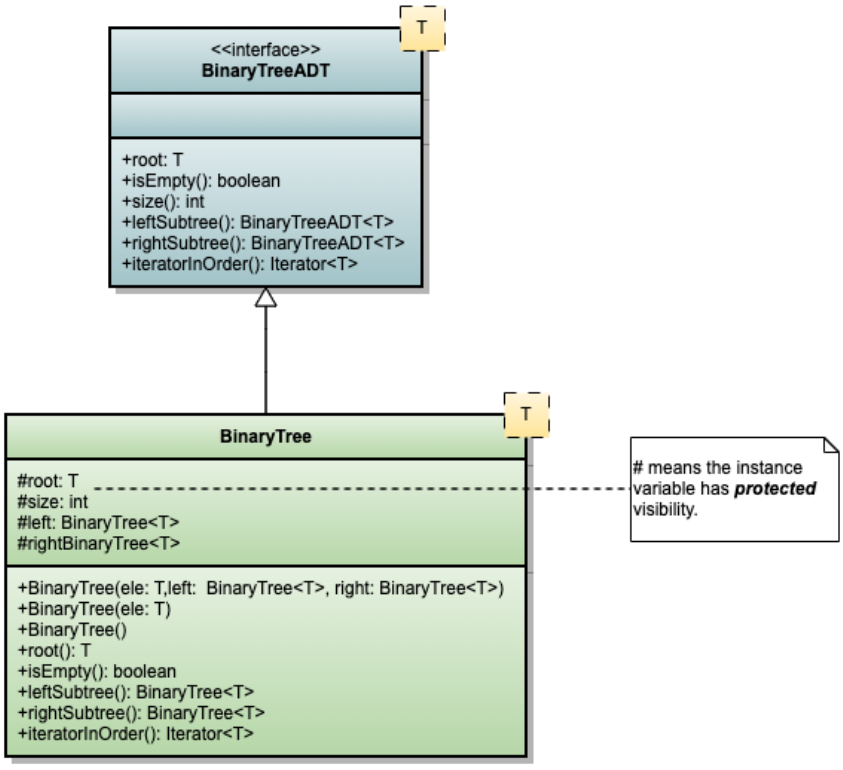


Figure 1: Binary Tree Design

3. Complete the BinaryTree<T> class as defined in Figure 1 by replacing the *TODO* comments with the appropriate code or comments. Note the declaration of the class uses protected visibility for instance variables as follows:

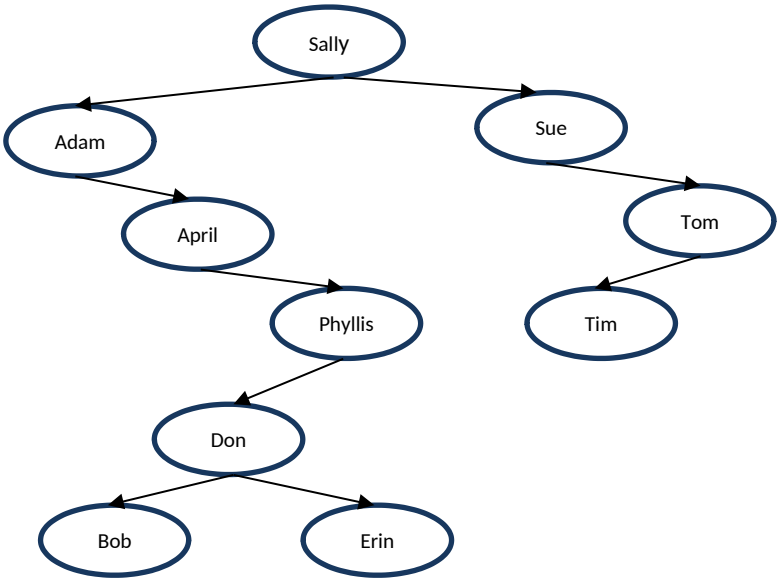
```
public class BinaryTree<T> implements BinaryTreeADT<T> {
        protected T root;          // root data in the tree
        protected int size;        // the number of nodes in the tree
        protected BinaryTree<T> left;    // left subtree
        protected BinaryTree<T> right: // right subtree
    . . .

}
```

To implement the TreeIterator class you will need to complete the iteratorInOrder method. We use the approach of creating a class to house the iterator, save the nodes in the tree in the correct order in an array, and maintaining the index of the current array element. The code for this class is included in the repo. You will need to code the heart of the inorder traversal as directed in the *TODO* comment in the inOrderTrav method.

Testing the Binary Tree

Complete the code for the BinaryTreeTest class as indicated in the *TODO* comments in the class. The idea in the testing class is to create the tree given in the figure on the top of the next page. You will have to create the tree from the leaves up to the root using the appropriate constructors to combine trees.



Use the iteratorInOrder to traverse the tree and print the names as they are retrieved from binTree. The easy way to do this is to use a foreach loop. When your program is working call your instructor over and demonstrate it.

Submit your completed code to your repository.

(Code for the TreeIterator<T> class on the following page.)

```
// This class provides an iterator for Binary Trees.
public class TreeIterator<T> implements Iterator<T> {
      // current iterator
      private int current;

      // collection size
      private int count;

      // Linear collection of nodes
      private T[] collectionArray;

      // The binary tree collection;
      private BinaryTree<T> collection;

      // Constructs a TreeIterator that iterates over the tree

      public TreeIterator(BinaryTree<T> collection, int size) {
            // set the initial iterator state
            current = 0;
            count = size;
            this.collection = collection;

            // make the collection
            collectionArray = (T []) new Object[count];

            // load the collectionArray
            inOrderTrav(this.collection);

            // reset current
            current = 0;

      } //end constructor

      // Method to check if more elements remain in the iteration
      public boolean hasNext() {
            return current < count;
      } // end hasNext

      // Method to return the next element in the iteration.
      public T next() {
            T retVal = collectionArray[current];
            current++;
            return retVal;
      } // end next

      // Not implemented but must be included.
      public void remove() {
      } // end remove
```

```
	// Private method to traverse the tree inorder, storing the
	// nodes visited in the collectionArray.
	private void inOrderTrav(BinaryTree<T> tree) {
		if (tree == null)
			return;
		else {
			// TO DO: add the code to implement the LVR
			//        traversal algorithm.  Note: The visit
			//        amounts to adding the root to the
			//        current element in collectionArray
			//        and incrementing current.
			//  Traversal is recursive:
			//    if (tree is empty)
			//       return
			//    Traverse the left subtree
			//    Visit =>Insert the root into collectionArray at
			//        current & increment current
			//    Traverse the right subtree
		}
	} // end inOrderTrav
} // end TreeIterator
```