CS 445: Data Structures
Fall 2018

Assignment 5

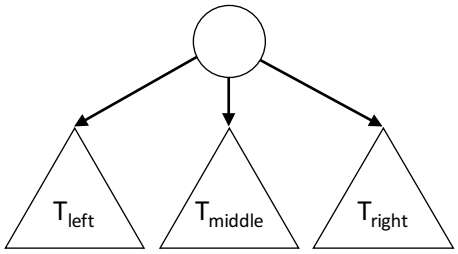
Assigned: Tuesday, November 20 **Due:** Thursday, December 6 11:59 PM

---

1 Motivation

In this assignment, you will be implementing a data structure as described in its ADT (represented using a Java interface). Specifically, you are provided with `TernaryTreeInterface`, an interface describing a tree where each node can have up to 3 children. Such a tree is defined as either:

1. Empty; or
2. of the following form, where $\mathsf{T}_{\mathsf{left}}$, $\mathsf{T}_{\mathsf{middle}}$, and $\mathsf{T}_{\mathsf{right}}$ are ternary trees.



When implementing this class, you may use the `BinaryTree` source code provided (package `cs445.binary`) as a starting point, or you can start from scratch. You are also allowed to use any classes from the Java Collections Framework such as `java.util.Stack` or `java.util.LinkedList` (these may be useful, e.g., in implementing tree traversal iterators), but you **cannot** use any Java-provided tree-like structures (e.g., `javax.swing.tree.TreeNode`).

2 Provided files

First, carefully read the provided files, in the usual place on Pitt Box.

The `cs445.binary` package contains the textbook's implementation of a binary tree.

The `cs445.StackAndQueuePackage` provides the textbook's implementations of Stack and Queue ADTs, which are used in the binary tree implementation (in particular, for the tree iterators).

`TernaryTreeInterface` and its superinterfaces are provided in the `cs445.a5` package. Do not modify these provided files.

3 Tasks

You must develop a class named TernaryTree that implements the interface TernaryTreeInterface. You may want to develop a TernaryNode class to represent the nodes of the tree, similar to the BinaryNode class discussed in the textbook and in lecture. As noted above, you may (but are not required to) use the source code provided with the textbook as a starting point.

When storing the children of a node, **you must use an array of node references**, *not* three stand-alone instance variables. You can create such an array, e.g., using the following line of code. Please review type erasure if this line is unclear.

```
@SuppressWarnings("unchecked")
TernaryNode<T>[] children = (TernaryNode<T>[])new TernaryNode<?>[3];
```

In this example, children[0] represents the left child, children[1] the middle child, and children[2] the right child.

The TernaryTreeInterface declares the abstract methods that are specific to the ternary tree, and also inherits the abstract methods of TreeInterface and TreeIteratorInterface. Thus, when implementing TernaryTreeInterface, you will also need to override the abstract methods declared in both of these superinterfaces.

TreeInterface declares basic tree operations such as getHeight, getNumberOfNodes, etc.

TreeIteratorInterface declares methods that create and return iterators for performing various traversals of the tree. Your TernaryTree class **must include the following inner classes that implement these iterators:** PreorderIterator, PostorderIterator, and LevelOrderIterator.

Your iterators *do not* need to support the remove() operation (recall that this operation is optional). That is, the remove() methods should throw java.lang.UnsupportedOperationException as with the examples in the book.

Your iterators are required to be efficient. That is, you may *not* do a full traversal of the tree when initializing the iterator, in order to determine the node order in advance. In addition, you may not restart the traversal every time the next() method is called (each iterator is required to store where it left off). This means that the depth-first traversals will need to use a stack to simulate recursion, and the level-order traversal should use a queue.

Furthermore, **you do not need to implement an inorder iterator.** Instead, getInorderIterator() should throw a java.lang.UnsupportedOperationException. In addition, **include in the comments** for this method a short explanation of why TernaryTree does not support inorder traversal.

The final interface, TernaryTreeBonus, declares two additional abstract methods that you may implement in your TernaryTree class for up to 8 bonus points. If you choose to complete this extra credit, your TernaryTree class should implement both TernaryTreeBonus and TernaryTreeInterface. You must also include a file named Bonus.txt **in the root of your Box folder** whose contents state that you completed the extra credit.

In addition to the methods required by the interfaces, your class *must* have the following constructors:

- public TernaryTree(): initializes an empty tree
- public TernaryTree(E rootData): initializes a tree whose root node contains rootData

- `public TernaryTree(E rootData, TernaryTree<E> leftTree, TernaryTree<E> middleTree, TernaryTree<E> rightTree)`: initializes a tree whose root node contains `rootData` and whose subtrees are `leftTree`, `middleTree`, and `rightTree`, respectively.

> **Note:** You are *highly encouraged* to write a test client that allows you to test the functionality of your implementation of `TernaryTree` prior to submission. You will be graded on each method's functionality as it compares to the descriptions in the interfaces, so make sure you test each one to ensure it behaves as expected!

4 Grading

Your grade for this assignment will be based on each method's functionality as it compares to the descriptions in both the interfaces and this document.

> **Note:** Code that cannot be compiled and tested as described will be given a grade of 0.

Method	Points
`TernaryTree()`	2
`TernaryTree(E)`	4
`TernaryTree(E, tree, tree, tree)`	6
`getRootData()`	2
`isEmpty()`	2
`getHeight()`	6
`getNumberOfNodes()`	6
`getLevelOrderIterator()`	16
`getPreorderIterator()`	18
`getPostorderIterator()`	20
`getInorderIterator()`	6
`clear()`	2
`setTree(E)`	4
`setTree(E, tree, tree, tree)`	6

4.1 Extra credit

In addition to the above methods from `TernaryTreeInterface` and its superinterfaces, you may also choose to make your `TernaryTree` implement `TernaryTreeBonus`. In this case, your `TernaryTree` class will need to override the two abstract methods in this bonus interface. If you choose to complete this extra credit, you must include a file named `Bonus.txt` **in the root of your Box folder** whose contents state that you completed the extra credit. **If you fail to include this file, your grader will not know that you completed the bonus, and you will not receive the extra credit.**

5 Submission

Upload your java files in the provided Box directory as additions to the provided code.

All programs will be tested on the command line, so if you use an IDE to develop your program, you must export the java files from the IDE and ensure that they compile and run on the command line. Do not submit the IDE's project files. Your TA should be able to download your `cs445-a5-abc123` directory from Box, and compile and test your code. Specifically, `javac cs445/a5/TernaryTree.java` must compile your program, when executed from the root of your `cs445-a5-abc123` directory.

In addition to your code, you may wish to include a `README.txt` file that describes features of your program that are not working as expected, to assist the TA in grading the portions that do work as expected. This file should be uploaded into the root of your `cs445-a5-abc123` directory, not in the package directory.

Your project is due at 11:59 PM on Thursday, December 6. You should upload your progress frequently, even far in advance of this deadline: **No late submissions will be accepted.**