

CS/COE 0445 Spring 2018 Assignment 4

Online Sunday, April 1, 2018

Due: All files (both those provided and the ones you have written, including directories for the packages) zipped into a single .zip file and submitted properly on the submission site by **11:59PM on Monday, April 16, 2018.**

Late Due Date: 11:59PM on Saturday, April 21, 2018.

Purpose and Goal: We have discussed (or will discuss) binary trees (BTs) and binary search trees (BSTs) in detail, in terms of both functionality and implementation. In this assignment you will further your understanding of BTs and BSTs by writing your own version of the author's TreePackage, called MyTreePackage. Most of MyTreePackage will be identical to the author's TreePackage. However, you will be adding to the functionality of the package by modifying some of the classes / interfaces already there, and by adding a new class, ComparableBinaryTree.

Details: The text author has provided the implementation of a BT and BST through the BinaryNode, BinaryTree and BinarySearchTree classes (plus numerous interfaces). However, there are some methods that could be useful that are not provided in the classes. It could also be useful to have a class that can store Comparable objects but that does not require them to be organized as in a BST. In this assignment you will modify the interfaces and classes as specified below, and you will also create a new class, ComparableBinaryTree, which will be a subclass of BinaryTree and the superclass of BinarySearchTree. You will test your resulting MyTreePackage using a driver program, [Assig4.java](#), which will be provided for you. Your output should exactly match that shown in file [A4Out.txt](#).

BinaryTreeInterface: Will now include the additional methods below

```
public boolean isFull();    // If the tree is a full tree, return true
                           // Otherwise, return false.  See notes for
                           // definition of full.
```

```
public boolean isBalanced(int k); // Return true if 1) the difference
    // in height between the left and right subtrees is at most k,
    // and 2) the left and right subtrees are both recursively
    // k-balanced; return false otherwise
```

```
public void saveInorder(String fileName) // Save the data in the BT to
    // file "fileName" using an inorder traversal.  Format the file
    // in the following way: It first contains an int representing
    // the number of nodes in the tree, followed by the actual
    // objects from the tree (inorder).  Use the writeObject()
    // method to write the objects using ObjectOutputStream
```

```
public void buildInorder(String filename) // Build a balanced BT
    // from the file "fileName".  Assume the first line of the file has
    // an integer, N, indicating the number of values to follow.  The
    // remaining N lines of the file contain N values.  The order of
    // the values in the file should be preserved by the tree (i.e. an
    // inorder traversal should show the data in the order stored in the
    // file).  Also if  $N = 2^k - 1$  for some K, the tree MUST be a FULL tree.
    // If N is any other value the tree should be optimally balanced for
    // that value of N but it will not necessarily be a complete tree.
    // This method MUST build the tree recursively (hint: have it call a
    // recursive method).  Use the readObject() method to read the
    // objects using an ObjectInputStream (so the file can contain any
    // type of Serializable objects).
```

ComparableTreeInterface: This is a new interface that extends **TreeInterface** and adds the following methods:

```
public T getMax() // If the tree is not empty, return the maximum
                  // value in the tree; otherwise return null
public T getMin() // If the tree is not empty, return the minimum
                  // value in the tree; otherwise return null
public boolean isBST() // Return true if the tree meets the
                      // recursive definition of a BST; else
                      // return false. See notes for BST definition.
public int rank(T data) // Return the rank of data in the tree with
                       // 0 being the smallest answer and N being the largest answer
                       // (if data is greater than all of the items in the tree).
                       // If there are duplicates, the rank of data should be minimized.
                       // data does not actually have to be present in the tree.
public T get(int i) // Return the value in the tree with rank equal
                   // to i. i should be in range 0 to N-1 (where the N is the number
                   // of nodes in the tree). If i is out of range, thrown an
                   // IndexOutOfBoundsException.
```

Note: The above interfaces have already been written for you – see the files **BinaryTreeInterface.java** and **ComparableTreeInterface.java** on the CS 0445 Assignments page (and read the comments). Below are the classes you must modify / implement.

BinaryNode class

- I already changed the class to "public" so that it can be accessed outside the package
- Add the following recursive methods [note that they MUST be recursive]

```
public boolean isFull(); // If the tree is a full tree, return true
                        // Otherwise, return false. See notes for
                        // definition of full.
public boolean isBalanced(int k); // Return true if 1) the difference
                                  // in height between the left and right subtrees is at most k,
                                  // and 2) the left and right subtrees are both recursively
                                  // k-balanced; return false otherwise
```

- These are the same methods specified in the **BinaryTreeInterface** above.
- As a starting point use the file **BinaryNode.java**, which is provided for you.

BinaryTree class

- Add the implementations of the methods above, so that **BinaryTree** still correctly implements **BinaryTreeInterface**. Note that because **isFull()** and **isBalanced()** will be implemented in the **BinaryNode** class, they will be trivial to implement in **BinaryTree**. Think why this is so. The other two new methods in **BinaryTreeInterface** will require more thought.
- To allow for easier testing, **one method specification will be changed**. The method **setRootNode()** must be changed from protected to **public**. [Note: This does not affect the real functionality of the class, and somewhat violates the principle of data hiding. However, it makes the test program a LOT easier to devise, as you will note in **Assig4.java**]
- As a starting point, use the file **BinaryTree.java**, which is provided for you.

ComparableBinaryTree class

- This new class should be a subclass of **BinaryTree**. Its data will be **Comparable** but they will not be in any particular order. The class header will be:

```
public class ComparableBinaryTree<T extends Comparable<? super T>>
    extends BinaryTree<T>
    implements ComparableTreeInterface<T>
```

- Include the implementations of the methods in `ComparableTreeInterface`. Since this class extends `BinaryTree` it will inherit the methods in `BinaryTreeInterface`.

BinarySearchTree class

- This class should now be a subclass of `ComparableBinaryTree` (rather than `BinaryTree`). It will have all of the functionality of the author's original BST, but will also override the methods (from `ComparableBinaryTree`) as specified below.

```
public T getMax()
public T getMin()      // override ComparableBinaryTree versions of
                      // these methods to tailor them to a BST (i.e.
                      // make them more efficient)

public boolean isBST() // override to always return true
public int rank(T data) // override to tailor implementation to BST
public T get(int i)    // override to tailor implementation to BST
```

- The idea here is that these methods can (perhaps) be done more efficiently given that the underlying tree is a BST. The `isBST()` method is an obvious example.
- **IMPORTANT NOTE / RESTRICTION:** You may not use any of the Iterators that are implemented in the `BinaryTree` class in any of your methods for the assignment.

Files:

You will need a number of files for this assignment – most of which are provided for you. These are necessary because of the many interfaces and classes used in the overall implementations. These files are all linked from the Assignments page in directory A4Files. [Note: The permissions are set on these files so that only computers with Pitt IP addresses may access them, so you need to download them from a computer that is within Pitt's network, or you must be connected via the VPN]. Within this directory are two subdirectories – `MyTreePackage` and `StackAndQueuePackage`. Set up your program in the following way:

- Create a directory in which you will put your main program (`Assig4.java`). Assume this directory is called `assig4`.
- Make two subdirectories in this directory, `MyTreePackage` and `StackAndQueuePackage`
- Copy all of the files from the A4Files site to the correct corresponding directory on your computer.
- The `StackAndQueuePackage` can be used as is – you do not have to modify it in any way.
- Many files in the `MyTreePackage` can be used as is. However, the following will need to be done to get the package to work:
 - `BinaryNode.java` must be modified as specified above
 - `BinaryTree.java` must be modified as specified above
 - `ComparableBinaryTree.java` must **be created** as specified above
 - `BinarySearchTree.java` must be modified as specified above

Hints, Notes and Extra Credit:

- To help you get started, I have implemented the `saveInorder()` method for you. See the `BinaryTree.java` file for this implementation.
- The `buildInorder()` method is tricky – think about it carefully before implementing it. As a hint, note that the recursive method you call should include a parameter that indicates the number of nodes to be stored in the current subtree. If you cannot get this method working, comment it out of `Assig4.java` program and indicate this in your Assignment Information Sheet.
- The `get()` method for the `ComparableBinaryTree` is also tricky – think about what the method is doing and what you know about the data in the `ComparableBinaryTree` and you will understand why it is tricky. Think creatively to figure out how to implement this method.
- As usual, write and test the methods one at a time – comment out the other parts of the program as you are testing specific methods.

- As an extra credit option, add a method to the `BinaryNode` class to draw it graphically and add a method to the `BinaryTree` class to draw the tree graphically. This is worth 10 points. For 5 points you can provide a method that will show the tree structure but that does not use any graphics.