Binary Search Trees

Overview

For this assignment you will be implementing the Binary Search Tree ADT and a number of associated functions.
Due: April 18th, 2014 @ 5pm

Files to complete

You are expected to write an implementation for each of the interfaces listed in the classes presented in the **config** package provided. As with the last assignment, you must specify which implementation you would like us to grade in this file.

In addition to this, you **MUST** include a String field in the **config.Configuration** file with your UMass student ID number. This is the 8 digit value found on your student ID card and is used to identify you in moodle. This will allow us to get your grade and feedback to you quickly. We recommend that you enter this ID value immediately after importing the project.

It will almost certainly be useful for you to write additional class files. Any class file you write that is used by your solution **MUST** be in the provided **src** folder. When we test your assignment, all files not included in the **src** folder will be ignored.

Note: When you submit your solution, be sure to remove **ALL** compilation errors from your project. Any compilation errors in your project may cause the autograder to fail and you will receive a zero for your submission.

Test files

In the test folder, you are provided with several JUnit test cases that will help you keep on track while completing this assignment. We recommend you run the tests often and use them as a checklist of things to do next. You are not allowed to remove anything from these files. If you have errors in these files, it means the structure of the files found in the **src** folder have been altered in a way that will cause your submission to lose points. We highly recommend that you add new @Test cases to these files. However, before submitting, make sure that your program compiles with the original test folder provided.

Support Code API

The Support Code's comments have been generated into a nicely formatted API that can be found here:
http://people.cs.umass.edu/~jcollard/cs187/s14/assignments/8/doc/

It is highly recommended that you spend some time simply reading over the comments in each of the interfaces and classes provided.

Part One: Importing Project into Eclipse

Begin by downloading the provided project from Moodle and importing it into your workspace.

You should now have a project called **bst-student.** It is very important that you not rename this project as it is used during the grading process. If the project is renamed, your assignment may not be graded.

By default, your project should have no errors and contain the following root items:
src - The source folder where all code you are submitting must go. You can change anything you want in this folder, you can add new files, etc...
support - This folder contains support code that we encourage you to use (and must be used to pass certain tests). You are not allowed to change or add anything in this folder
test - The test folder where all of the public unit tests are available
JUnit 4 - A library that is used to run the test programs
JRE System Library - This is what allows java to run

If you are missing any of the above or errors are present in the project, seek help immediately so you can get started on the project right away.

Once you have your project imported, add your 8 digit student ID number to the **config.Configuration** class.

Part Two: Implement the BinaryTreeNode Interface

A BinaryTreeNode represenst a node in a binary tree. It stores data of generic type T and may have a right and a left child, each a reference to another BinaryTreeNode. The BinaryTreeNode interface includes standard getters and setters for a BinaryTreeNode's right and left children as well as its data.

Part Three: Implement the BinaryTreeUtility Interface

The BinaryTreeUtility interface provides basic functions for working with a binary tree.

Depth -- The depth of the tree is the maximum level of any leaf node in the tree. Recall that the level of a node begins at zero for the root of a tree with no children. A child of the root is at level 1.

Related posts:
https://piazza.com/class/hqgqcfm0chh1r2?cid=667

Balance -- The balance of a tree measures how close it is to a full or complete tree. You will implement the method isBalanced(BinaryTreeNode<T> root, int tolerance) which determines whether the maximum difference in the depth of any two children is no larger than the given tolerance value.

Related posts:
https://piazza.com/class/hqgqcfm0chh1r2?cid=636

Testing the BST property -- Recall that a BST is a binary tree that also satisfies a special sorting property: if all elements in left subtree of any node X are less than or equal to node X and all nodes in the right subtree of X are greater than X. You will write a function isBST(root) which returns true if root is the root of a valid binary search tree. Your function need not explicitly test basic requirements of the binary tree (e.g. that there is a unique path from the root to every node in the tree). Testing the BST property has a nice recursive solution, but it requires some thought. If you'd like a hint, please see the following wikipedia page:

http://en.wikipedia.org/wiki/Binary_search_tree#Determining_whether_a_tree_is_a_BST_or_not

Iterators -- You will also provide three methods getPreOrderIterator(BinaryTreeNode<T> root), getInOrderIterator(BinaryTreeNode<T> root), getPostOrderIterator(BinaryTreeNode<T> root) each returning an iterator that follows the stated traversal over a binary tree.

For Extra Credit: If you choose to implement an iterator method that returns in O(1) time, you will receive extra credit. This means you should not compute the traversal in its entirety when creating and returning the iterator. Instead, the iterator should be initialized and returned as a result of the function call, with the next element in the traversal computed with each call to next().

Related Posts:
https://piazza.com/class/hqgqcfm0chh1r2?cid=642
https://piazza.com/class/hqgqcfm0chh1r2?cid=653
https://piazza.com/class/hqgqcfm0chh1r2?cid=652

Hint: A PreOrderIterator class has been provided as an example to get started.
Tip: The iterative versions of Post / In order traversal can be found here: http://en.wikipedia.org/wiki/Tree_traversal

Feeling ambitious? See if you can make a Level-Order iterator. http://en.wikipedia.org/wiki/File:Sorted_binary_tree_breadth-first_traversal.svg

Part Four: Implement the BinarySearchTree Interface

Next you will implement methods underlying the BST structure.

Transformers -- Add and remove are the main transformers of a BST. These must add or remove elements while maintaining the BST property. You may follow the textbook's implementation of these methods (which was discussed in class). You may also find variations of the methods that work.

Observers -- You will implement an isEmpty() method and a size() method. In addition, you will implement the getMinimum() and getMaximum() functions, which return the smallest and largest values stored in the BST.

Iterator -- This function returns an iterator that supports in-order access to the nodes of the BST. Recall that an in-order traversal of a BST results in a sorted order due to the BST property. The method returning an iterator should complete in O(1) time, so it should not compute the entire sequential order ahead of time.

Related Posts:

https://piazza.com/class/hqgqcfm0chh1r2?cid=647
https://piazza.com/class/hqgqcfm0chh1r2?cid=654

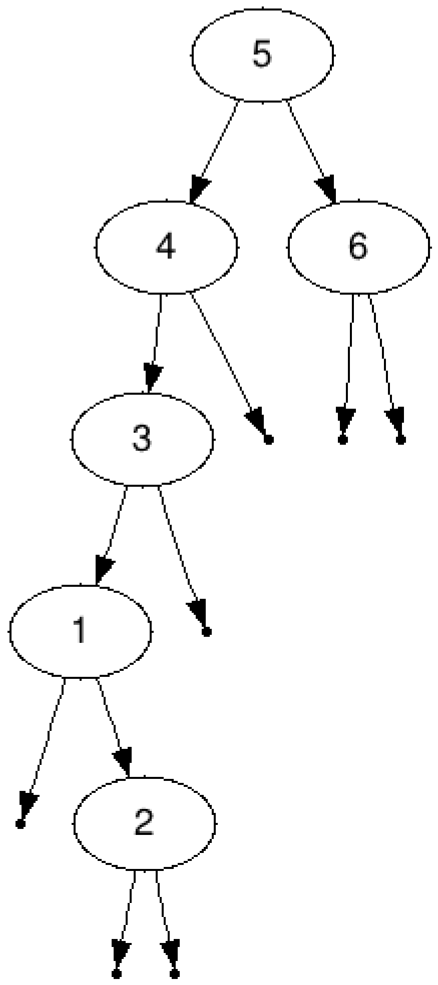
Part Five: Visualize your Binary Tree

We have provided you with a function that traverses a binary tree and constructs a special string which is in the "dot" format of a graph visualization package called GraphViz. Using the TreeViewer class you can create a binary tree (or BST) and print its representation in the dot format. You may then copy this string to one of the following websites and you will be able to visualize the structure of your tree. This is simply a tool that may help you when developing your code. You do not have to do anything for this part of the assignment.

http://sandbox.kidstrythisathome.com/erdos
http://graphviz-dev.appspot.com

If you add values 5, 6, 4, 3, 1, 2 to an initially empty BST, the dot representation produced by dotFormat() is below. You don't have to understand the details. Simply copy the text into one of the websites above and it will produce the image shown below:

```
digraph G {
graph [ordering="out"];
5 -> 4;
5 -> 6;
4 -> 3;
node0 [shape=point];
4 -> node0;
node1 [shape=point];
6 -> node1;
node2 [shape=point];
6 -> node2;
3 -> 1;
node3 [shape=point];
3 -> node3;
node4 [shape=point];
1 -> node4;
1 -> 2;
node5 [shape=point];
2 -> node5;
node6 [shape=point];
2 -> node6;
};
```



Part Six: Export and Submit

When you have finished and are ready to submit, export the entire project. Be sure that the project is named **bst-student**. Save the exported file with the zip extension (any name is fine).

Log into Moodle and submit the exported zip file.