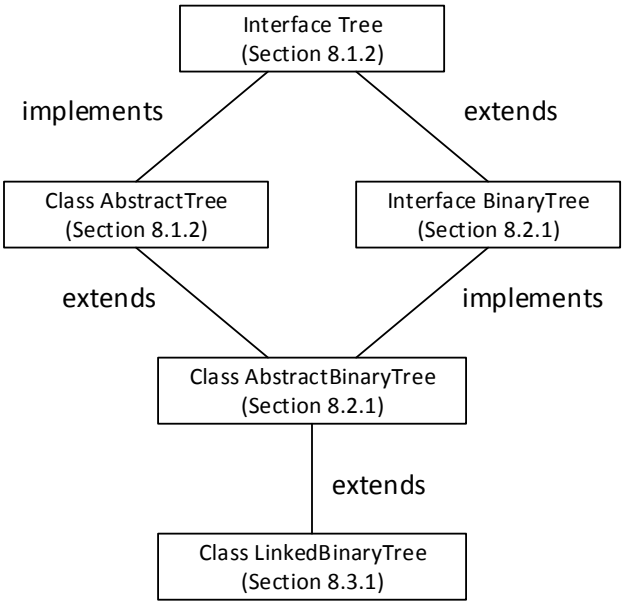
CS526
Homework Assignment 3

This assignment is an implementation of a ***generic binary search tree***, which extends the *LinkedBinaryTree* class. Binary trees are discussed in Section 8.2 and Section 8.3.1 in the textbook. Binary search tree is discussed in page 338 of the textbook.

The *LinkedBinaryTree.java*, which comes with the textbook, is a concrete implementation of a binary tree that uses a linked structure. It extends the *AbstractBinaryTree* class. The relevant class hierarchy is shown below:



The *LinkedBinaryTree* inherits variables and methods from its superclasses and it also implements its own methods.

The requirements of this assignment is given below:

(1) Create a generic binary search tree class named *MyBST.java* and implement an *add* method within the class. An incomplete code is posted on Blackboard and you must complete the code.
(2) Perform an experiment that determines an average height of binary search trees.

Detailed requirements are given below.

Create a generic class named *MyBST* as a subclass of *LinkedBinaryTree*. The *MyBST* is an implementation of a binary search tree using a linked tree structure.

A *binary search tree* is a binary tree that satisfies the following *binary search tree property*.

For each internal position *p*:

- Elements stored in the left subtree of *p* are less than *p*'s element.
- Elements stored in the right subtree of *p* are greater than *p*'s element.

You are required to implement an *add* method using the following pseudocode:

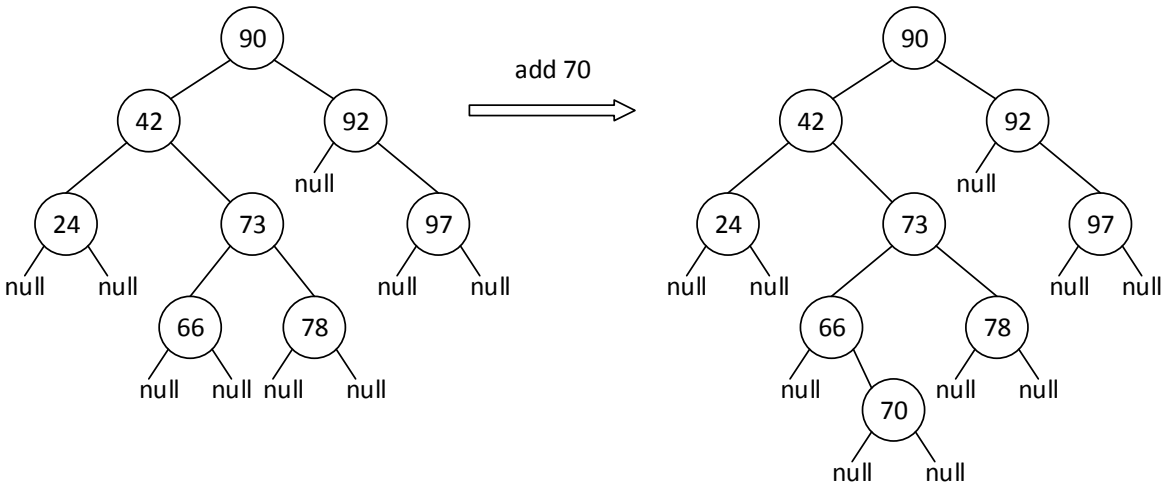
```
Algorithm add(p, e)
  Input parameters:
    p: The position of the root of the tree (or subtree) to which
       a new node is added
    e: The element of the new node to be added
  Output: Returns the position of the new node that was added.
    If there already is a node with e in the tree, returns null.

  if p == null // this is an empty tree
    create a new node with e and make it the root of the tree
    increment size // size is the number of nodes currently in the tree
    return the root

  x = p
  y = x
  while (x is not null) {
    if (the element of x) is the same as e, return null
    else if (the element of x) > e {
      y = x
      x = left child of x
    }
    else {
      y = x
      x = right child of x
    }
  } // end of while

  temp = new node with element e
  y becomes the parent of temp
  if (the element of y) > e
    temp becomes the left child of y
  else
    temp becomes the right child of y
  increment size // size is the number of nodes currently in the tree
  return temp
```

The following figure illustrates adding a node to a binary search tree. If you add a node with 70 to the tree on the left, the result is the tree on the right.



You may want to read and study the codes of *LinkedBinaryTree* and its superclasses and super interfaces carefully before writing a code.

An incomplete code of *MyBST.java* is posted on Blackboard. You need to complete this code as indicated and test your *add* method in the main method.

Then, perform the following experiment.

This experiment determines an average height of binary search trees when integers are added in a random order.

You are required to write a code segment in your main method to experimentally determine an average height of binary search trees.

First, you need to generate 1,000 distinct random integers in the range [0, 999999] and add them, one at a time, to an initially empty binary tree, using the *add* method you implemented. **Make sure that the resulting tree has 1,000 distinct integers**. Then, determine the height of the tree. Repeat this 100 times and calculate the average height of these 100 binary search trees. A sample code that generates a random integer in [0, 999999] is shown below:

```
int e;
Random r = new Random();
r.setSeed(System.currentTimeMillis());
e = r.nextInt(1000000);
```

You may want to refer to Java's manual for more information about the *Random* class.

Output: For each iteration, your program must print the height of the tree and the number of nodes of the tree. Then, after 100 iterations, your program must print the average height of 100 trees. A sample output is shown below:

```
Height = xx, Size = 1000
Height = xx, Size = 1000
```

```
Height = xx, Size = 1000
Height = xx, Size = 1000
Height = xx, Size = 1000
Height = xx, Size = 1000

. . .

Average height = xx.xx // this is the average of 100 heights
```

Documentation

No separate documentation is needed. However, you must include sufficient inline comments within your program.

Grading

Your facilitator will test your *add* method by adding a sequence of integers. They will repeat this three times with three different sequences of integers and 10 points will be deducted for each sequence that generates an incorrect result or an error.

If the average height of binary search trees determined by your program is outside the generally expected range, 10 points will be deducted.

Points will be deducted up to 20 points if you do not include sufficient inline comments.

Deliverables

You need to complete the *MyBST.java* file. Combine this file with all other files, which are necessary to compile and execute your program, into a single archive file, such as a *zip* file or a *rar* file, and name it *LastName_FirstName_hw*3.*EXT*, where *EXT* is an appropriate file extension (such as *zip* or *rar*). Upload it to Blackboard.