COMP1006/1406 – Winter 2022

Submit a single file called `assignment4-P2.zip` to the cuLearn with all of your `.java` files. Do not include your `.class` files.

This assignment has 10 marks.
All marks will be based on correctness and efficiency of your code.
48-hour grace period for submissions allowed.

1 BST (Binary Search Tree) [10 marks]

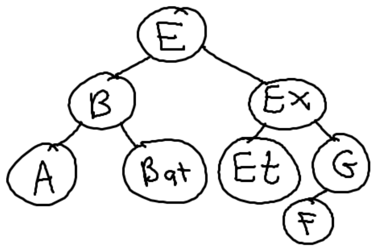
A **Binary Search Tree** (**BST**) can be used to efficiently implement a sorted set. It stores unique values and offers an efficient membership method (contains).

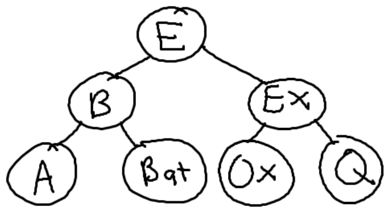
A binary search tree (BST) is a binary tree with the additional binary search tree property. The binary search tree property states that for every node `n` in the tree

1. `n.data` is greater than the `data` value of all nodes in the sub-tree rooted with `n.left`.
2. `n.data` is less than the `data` value of all nodes in the sub-tree rooted with `n.right`.

As we are implementing a set, all data (strings) in the tree will be unique. Every node in the tree will have its own distinct string. Strings will be compared using the canonical way (as defined by the `compareTo()` method in the String class).

For example, the binary tree on the left (below) IS a binary search tree and the one on the right is NOT.





You will implement several methods in the `BinaryTree` and `BST` classes. The `BST` class must extend the `BinaryTree` class. Methods that you must complete (implement or override) are as follows:

BinaryTree: `contains(String)`, `isBST()`

BST: `contains(String)`, `add(String)`, `makeBalanced()`

In the BinaryTree class

```
public boolean contains(String s)
  // precondition: s is string
  // postcondition: returns true if s in the this tree, false otherwise
  // constraint:  this method MUST use recursion

public boolean isBST()
  // note: an empty tree is a valid BST
  // postcondition: returns true if this tree is a valid binary search tree, false otherwise
  // notes: A BST object might NOT satisfy the binary search tree property.
  //        This method should not use instanceof. It should determine if the
  //        values and structure of this tree satisfy the binary search tree property or not.
```

How can you check if a binary tree is a binary search tree? You can use the following fact to help with this: *A binary tree is a binary search tree if and only if an inorder traversal of the tree will display the values in sorted order.* You are free to implement this as you wish. A suggestion would be for you to modify the traversal to build a list (an ArrayList for example) and then check if the resulting list is in sorted order.

In the BST class

```
@Override
public boolean contains(String s)
  // precondition: s is string and this BST is a valid binary search tree
  // postcondition: returns true if s in the this tree, false otherwise
  // efficiency: (i) this method must be efficient
  //             (ii) do NOT use recursion for this method

@Override
public void add(String s)
  // precondition:  (i) s is a string
  //               (ii) this BST is a valid binary search tree
  // postcondition: (i) s is added to this tree, if it is not already
  //                    in the tree, and tree remains a valid binary search tree
  //               (ii) s is added as a leaf in the tree (if added)

public BST makeBalanced()  // optional
  // postcondition: returns a new binary search tree that
  //                (i) has the same data (strings) as this tree
  //                (ii) has minimal height
  //                (iii) is a valid binary search tree
```

The makeBalanced() method must create a new balanced binary search tree with minimal height. For a given tree there may be many different valid binary search trees with the same minimal height. It does not matter which optimal tree you return. This method is not required for the assignment.

Note: The height of tree is the longest distance from the root of the tree to any of its leaf nodes. A binary tree with a single node has height 0. A binary tree with two nodes has height 1. The BinaryTree class has a height method that computes the height of a binary tree.

There is a PrintBinaryTree class that is also provided. This will allow you to visualize your trees when developing, testing and debugging your code.

Submission Recap

A complete assignment will consist of a single file (`assignment4-P2.zip`) that has the following files included in it:

```
BinaryTree.java
BST.java
```

You classes should NOT be part of a package.

Do NOT use any other classes that involve binary trees or binary search trees.

You can use any other classes as you see fit.