Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	07-02
Receipt ID:	5a58f7a7a37251d6e48e21a5934276dd

Turnin Successful!

The following file(s) were received:

BinarySearchTree.java (5477 bytes)

```
//name:    date:
import java.util.*;
/*****************************************************************
   Make an ArrayList of input strings.  Build the Binary Search Trees
(using TreeNodes) from the letters in the string.   Display it as a sideways
tree (take the code from TreeLab).  Prompt the user for a target and
search the BST for it.  Display the tree's minimum and maximum values.
Print the letters in order from smallest to largest.
   ****************************************************************/
public class BinarySearchTree
{
   public static Scanner keyboard = new Scanner(System.in);
   public static void main(String[] args)
   {
      ArrayList<String> str = new ArrayList<String>();
      str.add("MAENIRAC");
      str.add("AMERICAN");
      str.add("AACEIMNR");

      for( String s : str )
      {
         System.out.println("String: "  + s);
         TreeNode root = null;
         root = buildTree( root, s );
         System.out.println( display(root, 0) );
         System.out.print("Input target: ");
         String target =  keyboard.next();    //"I"
         boolean itemFound = find(root, target);
         if(itemFound)
            System.out.println("found: " + target);
         else
            System.out.println(target +" not found.");

         System.out.println("Min = " + min(root));
         System.out.println("Max = " + max(root));

         System.out.print("In Order: ");
         System.out.println( smallToLarge(root) );
         System.out.println("\n---------------------");
      }
   }
   public static TreeNode buildTree(TreeNode t, String s)
   {
      for(char c: s.toCharArray())
         t = insert(t, ""+c);
      return t;
   }
      /*************************
      Recursive algorithm to build a BST:  if the node is null, insert the
      new node.  Else, if the item is less, set the left node and recur to
      the left.  Else, if the item is greater, set the right node and recur
      to the right.
      ******************************/
   private static TreeNode insert(TreeNode t, String s)
   {
      if(t == null)
         t = new TreeNode(s);
      else if(s.compareTo((String)t.getValue())<=0)
      {
         if(t.getLeft()==null)
            t.setLeft(new TreeNode(s));
         else
            insert(t.getLeft(),s);
      }
      else if(s.compareTo((String)t.getValue())>0)
      {
         if(t.getRight()==null)
            t.setRight(new TreeNode(s));
         else
            insert(t.getRight(),s);
      }
      return t;
   }

   /* copy the code that is in TreeLab  */
   public static String display(TreeNode t, int level)
   {
      String toRet = "";
      if(t == null)
         return "";
      toRet += display(t.getRight(), level + 1); //recurse right
      for(int k = 0; k < level; k++)
         toRet += "\t";
      toRet += t.getValue() + "\n";
      toRet += display(t.getLeft(), level + 1); //recurse left
      return toRet;

   }
      /********************
      Iterative algorithm:  create a temporary pointer p at the root.
      While p is not null, if the p's value equals the target, return true.
      If the target is less than the p's value, go left, otherwise go right.
      If the target is not found, return false.

      Find the target. Recursive algorithm:  If the tree is empty,
      return false.  If the target is less than the current node
      value, return the left subtree.  If the target is greater, return
      the right subtree.  Otherwise, return true.
      ******************************/
   public static boolean find(TreeNode t, Comparable x)
   {
      if(t==null)
         return false;
      else if(x.compareTo(t.getValue())<0)
         return find(t.getLeft(), x);
      else if(x.compareTo(t.getValue())>0)
         return find(t.getRight(), x);
      return true;
   }
      /***************************
      starting at the root, return the min value in the BST.
      Use iteration.   Hint:  look at several BSTs. Where are
      the min values always located?
      ******************************/
   public static Object min(TreeNode t)
   {
      while(t.getLeft()!=null)
         t = t.getLeft();
      return t.getValue();
   }
      /*****************
      starting at the root, return the max value in the BST.
      Use recursion!
      ********************/
   public static Object max(TreeNode t)
   {
      if(t.getRight()==null)
         return t.getValue();
      return max(t.getRight());
   }
   public static String smallToLarge(TreeNode t)
   {
      String toReturn = "";
      if(t == null)
         return "";
      toReturn += smallToLarge(t.getLeft());
      toReturn += t.getValue() + " ";
      toReturn += smallToLarge(t.getRight());
      return toReturn;

   }
}
   /***************************************
 String: MAENIRAC
      R
   N
 M
         I
      E
         C
   A
      A
 Input target: I
 found: I
 Min = A
 Max = R
 In Order: A A C E I M N R
 ---------------------
 String: AMERICAN
      R
         N
   M
         I
      E
         C
 A
   A
 Input target: I
 found: I
 Min = A
 Max = R
 In Order: A A C E I M N R
 ---------------------
 String: AACEIMNR
                  R
               N
            M
         I
      E
   C
 A
   A
 Input target: i
 i not found.
 Min = A
 Max = R
 In Order: A A C E I M N R
 ---------------------
 ************************************/
```