Programming Homework Assignment #4
Due Date: Fri., Feb. 18, 11:55 PM
Upload the source files (.java files) with output (copied and pasted to the end of the main file) ZIPPED into one .zip file (submit under Week 7)

Problem:
Write a Java program which changes and completes the Binary Tree and Binary Search Tree (BST) classes and practices using BST objects.

Complete the BinaryTree class member methods given in the HW4_JavaCode file (I'm calling "code file" below):

- "copy" constructor, which is just a constructor which has a BinaryTree parameter (assign the return value of calling deepCopyTree(passing parameter's root) to this' root)
- deepCopyTree (algorithm given in the code file as PROTECTED)
- protected inorder method (similar to preorder) with the BinaryNode parameter as well as Visitor parameter
- protected postorder (similar to preorder) with the BinaryNode parameter as well as Visitor parameter

Change part of the Binary Search Tree (BST) class as indicated in the HW4_JavaCodefile (note that the class heading is now: `public class BST<E extends DeepCloneable<E>> extends BinaryTree<E>`, but the class has a PRIVATE Comparator variable):

- ADD to the default constructor (and the constructor with an array parameter) a Comparator<E> parameter and assign the parameter to the Comparator<E> instance variable
- ADD a "COPY constructor" that calls the BinaryTree's deepCopyTree, then assigns the Comparator<E> of the parameter to the instance variable
- every method that compares data MUST use the compare method that has 2 E parameters and returns an int (examples will be given in a separate file)
- complete/fix these methods:
  - getEntry (algorithm given in the code file)
  - findNode (algorithm given in the code file)
  - getFirst() and getLast() (see details in the code file)

Use the separate class given in the HW4-JavaCodeFile to be used for the data in the list (I'm calling the class DeAnzaContact) which implements DeepCloneable<DeAnzaContact>.

Define 2 Comparator classes which implement Comparator<DeAnzaContact> (hints given in the Comparators file on Catalyst):

- called NameComparator that overrides one method: int compare (DeAnzaContact left, DeAnzaContact right), returns the result of compareToIgnoreCase for left's last name (pass right's last name) IF it's not 0, else return the result of compareToIgnoreCase for left's

first name (pass right's first name) IF it's not 0, otherwise return the result of compareToIgnoreCase for left's department (pass right's department)

- called DeptComparator that overrides one method: int compare (DeAnzaContact left, DeAnzaContact right), which returns result of compareToIgnoreCase for left's department (pass right's department IF it's not 0, else return the result of compareToIgnoreCase for left's last name (pass right's last name) IF it's not 0, else return the result of compareToIgnoreCase for left's first name (pass right's first name)

Define 2 Visitor classes which implement Visitor<DeAnzaContact>:

- called NameVisitor that overrides one method: void visit(DeAnzaContact card) that writes the toString of the parameter to System.out
- call DeptVisitor that overrides one method: void visit(DeAnzaContact card) that writes the members of the DeAnzaContact in the format: department, last name, first name, phone (use the accessors)

Write main so it has 2 BST<DeAnzaContact> variables. Assign to one of the variables a new BST with a new NameComparator passed as an argument. Assign to the 2nd variable a new BST with a new DeptComparator as an argument. Then do the following (for which most should be a function or points may be deducted if main is too long): (MUST BE IN THIS ORDER)

- read the file which has several sets of DeAnzaContacts, which has one line per string. In a loop (to the end of file), after reading one set of 4 strings (in the order last, first, phone, dept), create a DeAnzaContact object (passing the 4 strings) and insert the SAME DeAnzaContact object to each BST
- call each BST's **inorder** method (one at a time), passing a NameVisitor for the name-ordered BST and a DeptVisitor for the dept-ordered BST
- call the static **testBST** method (given in the code file, should be in the same class as main) for EACH BST (one at a time)
- call a <u>method that you write</u> that tests deleting from ONE BST<DeAnzaContact> (both are parameters) in ONE method so it does the following for the BST parameter:
  - get the first and last DeAnzaContact from the tree
  - try to delete the first, then the last, check if successful and display a message indicating if deleted (or if not deleted if unsuccessful)
- call the static **testClone** method given in the code file, should be in the same class as main)
- call **postorder** for ONE OF THE BST passing either a NameVisitor or DeptVisitor (depending on which tree you called this for)

See test runs on Catalyst. Use the input files given on Catalyst.

Extra Credit Problems (due the last day of the quarter!): Textbook (Data Abstraction & problem Solving by Carrano) pp. 490-491 #19 (for our BST), and TEST in a program.