Binary Search Tree

This assignment involves implementing a standard binary search tree as a Java generic. Because this assignment will be auto-graded using a test harness we will provide, your implementation must conform to the public interface below, and include at least all of the public and private members that are shown:

```
// The test harness will belong to the following package; the BST
// implementation will belong to it as well.  In addition, the BST
// implementation will specify package access for the inner node class
// and all data members in order that the test harness may have access
// to them.
//
package MinorP2.DS;

public class BST<T extends Comparable<? super T>> {

    class BinaryNode {
       // Initialize a childless binary node.
       public BinaryNode( T elem ) { . . . }
       // Initialize a binary node with children.
       public BinaryNode( T elem, BinaryNode lt, BinaryNode rt ) { . . . }
       // Clone node (see specification for details).
       public BinaryNode clone();

       T          element;  // The data in the node
       BinaryNode left;     // Pointer to the left child
       BinaryNode right;    // Pointer to the right child
    }

    BinaryNode root;        // pointer to root node, if present.

    // Initialize empty BST; an empty BST contains no nodes.
    public BST( ) { . . . }
    // Return true iff BST contains no nodes.
    public boolean isEmpty( ) { . . . }
    // Return pointer to matching data element, or null if no matching
    // element exists in the BST.  "Matching" should be tested using the
    // data object's compareTo() method.
    public T find( T x ) { . . . }
    // Insert element x into BST, unless it is already stored.  Return true
    // if insertion is performed and false otherwise.
    public boolean insert( T x ) { . . . }
    // Delete element matching x from the BST, if present.  Return true if
    // matching element is removed from the tree and false otherwise.
    public boolean remove( T x ) { . . . }
    // Return the tree to an empty state.
    public void clear( ) { . . . }
    // Return true iff the two objects involved are both BSTs, have the
    // same physical structure, and store equal values in corresponding
    // nodes.  “Equal” should be tested using the data object’s equals()
    // method.
    public boolean equals(Object other);
}
```

You may safely add features to the given interface, but if you omit or modify members of the given interface you will almost certainly face compilation errors when you submit your implementation for testing.

You must implement tree traversals recursively, not iteratively. You will certainly need to add a number of private recursive helper functions that are not shown above. Since those will never be called directly by the test code, the interfaces are up to you.

You must place the declaration of your BST generic in a package named `MinorP2.DS` and specify package access for members as indicated above, or compilation will fail.

Design and implementation requirements

There are some explicit requirements, in addition to those on the *Programming Standards* page of the course website:

- You must implement the BST to conform to the specification.
- The insertion logic must not allow duplicate records to be inserted.
- Deletion must be handled exactly as described later in this specification.
- The `equals()` method must conform to the "equals-contract" described in the course notes.
- Under no circumstances should any of the specified BST member functions write output.

Testing:

We will be testing your implementation with our own test driver. We may (or may not) release information about that driver before the assignment is due. In any case, it is your responsibility to design and carry out a sensible test of your implementation before submitting it. For that purpose, you may share test code (**but absolutely no tree code!!**) via the class Forum.

Be sure you test all of the interface elements thoroughly, both in isolation and in interleaved fashion.

Evaluation:

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation and design, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

Note that the evaluation of your project may depend substantially on the quality of your code and documentation.

What to turn in and how:

This assignment will be auto-graded using a test harness on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.6.21.

Submit a single `.java` file (not zipped) containing your BST generic to the Curator System. Submit nothing else. Your solution should not write anything to standard output.

Your submitted source file will be placed in the appropriate subdirectory with the packaged test code, and will then be compiled with the test harness using the following command, executed in the root of the source directory tree:

```
javac testDriver.java
```

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

`http://www.cs.vt.edu/curator/`.

You will be allowed to submit your solution multiple times; the highest score will be counted.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Programming Standards page in one of your submitted files.

Note on deletion:

The test harness will expect you to handle deletion in a precisely-specified manner. Deletion of a leaf simply requires setting the pointer to that node to `null`. Deletion of a node with one non-empty subtree simply requires setting the pointer to that node to point to the node's subtree.

Deletion of a node with two non-empty subtrees must be handled in the following manner. First, locate the node rMin that holds the minimum value in the right subtree of the node to be deleted. Then, detach rMin from the right subtree using the appropriate logic for a leaf or a one-subtree node. Next, replace the node to be deleted with rMin, by resetting pointers as necessary. Do not simply copy the data reference from rMin to the node containing the value to be deleted.

The test harness will expect deletion to be handled in precisely this manner.