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 Minor.P2.DS;

// BST<> provides a generic implementation of a binary search tree
//
// BST<> implementation constraints:
//   - The tree uses package access for root, and for the node type.
//   - The node type uses package access for its data members.
//   - The tree never stores two objects for which compareTo() returns 0.
//   - All tree traversals are performed recursively.
//   - Optionally, the BST<> employs a pool of deleted nodes.
//     If so, when an insertion is performed, a node from the pool is used
//     unless the pool is empty, and when a deletion is performed, the
//     (cleaned) deleted node is added to the pool, unless the pool is
//     full.  The maximum size of the pool is set via the constructor.
//
// User data type (T) constraints:
//   - T implements compareTo() and equals() appropriately
//   - compareTo() and equals() are consistent; that is, compareTo()
//     returns 0 in exactly the same situations equals() returns true
//
public class BST<T extends Comparable<? super T>> {

    class BinaryNode {
       // Initialize a childless binary node.
       // Pre:   elem is not null
       // Post:  (in the new node)
       //        element == elem
       //        left == right == null
       public BinaryNode( T elem ) { . . . }

       // Initialize a binary node with children.
       // Pre:   elem is not null
       // Post:  (in the new node)
       //        element == elem
       //        left == lt, right == rt
       public BinaryNode( T elem, BinaryNode lt, BinaryNode rt ) { . . . }

       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
    BinaryNode pool;        // pointer to first node in the pool
    int        pSize;       // size limit for node pool
```

```
    // Initialize empty BST with no node pool.
    // Pre:   none
    // Post:  (in the new tree)
    //        root == null, pool == null, pSize = 0
    public BST( ) { . . . }

    // Initialize empty BST with a node pool of up to pSize nodes.
    // Pre:   none
    // Post:  (in the new tree)
    //        root == null, pool = null, pSize == Sz
    public BST( int Sz ) { . . . }

    // Return true iff BST contains no nodes.
    // Pre:   none
    // Post:  the binary tree is unchanged
    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.
    // Pre:  x is null or points to a valid object of type T
    // Post: the binary tree is unchanged
    public T find( T x ) { . . . }

    // Insert element x into BST, unless it is already stored.  Return true
    // if insertion is performed and false otherwise.
    // Pre:   x is null or points to a valid object of type T
    // Post:  the binary tree contains x
    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.
    // Pre:   x is null or points to a valid object of type T
    // Post:  the binary tree does not contain x
    public boolean remove( T x ) { . . . }

    // Delete the subtree, if any, whose root contains an element matching x.
    // Pre:   x is null or points to a valid object of type T
    // Post:  if the tree contained x, then it the subtree rooted at that
    //           node has been removed
    public boolean prune( T, x) { . . . }

    // Return the tree to an empty state.
    // Pre:   none
    // Post:  the binary tree contains no elements
    public void clear( ) { . . . }

    // Return true iff other is a BST that has the same physical structure
    // and stores equal data values in corresponding nodes.  "Equal" should
    // be tested using the data object's equals() method.
    // Pre:   other is null or points to a valid BST<> object, instantiated
    //           on the same data type as the tree on which equals() is invoked
    // Post:  both binary trees are unchanged
    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 all 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 binary search tree generic in a package named `Minor.P2.DS` and specify package access for members as indicated above, or compilation will fail when you submit it.

Note on deletion

The test harness will expect you to handle deletion of a node 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, and will penalize you if it is not.

Note on the node pool

If the node pool is enabled (by calling the second constructor):

- The pool will hold up to `pSize` nodes, which will not contain data elements (i.e., `null` data reference).
- The nodes in the pool will be linked into a linear list via their `right` child pointers.
- When a deletion is performed, the node will be added to the pool unless the pool is full.
- When an insertion is performed, a node from the pool will be used unless the pool is empty.

Design and implementation requirements

There are some explicit requirements, stated in the header comment for `BST<>`, in addition to those on the *Programming Standards* page of the course website. Your implementation must conform to those requirements. In addition, none of the specified `BST<>` member functions should 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 or later.

Submit your `BST.java` file (not a zip or jar) 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 following pledge statement at the beginning of the file that contains `main()`:

```
//    On my honor:
//
//    - I have not discussed the Java language code in my program with
//      anyone other than my instructor or the teaching assistants
//      assigned to this course.
//
//    - I have not used Java language code obtained from another student,
//      or any other unauthorized source, either modified or unmodified.
//
//    - If any Java language code or documentation used in my program
//      was obtained from another source, such as a text book or course
//      notes, that has been clearly noted with a proper citation in
//      the comments of my program.
//
//    - I have not designed this program in such a way as to defeat or
//      interfere with the normal operation of the Curator System.
//
//    <Student's Name>
```

We reserve the option of assigning a score of zero to any submission that does not contain this statement.