Goals
- to use the binary tree node class - BTnode<E>
- to build a binary search tree class
- to gain more recursion experience

Overview
You MUST use binary trees as developed in class for this assignment. The approach differs from the one in the text, but refer to the text for examples and a related explanation. If there is a conflict, what we've done in class is right. As always, ask for help if you are confused.

For this assignment, you will write the insert and remove method for a program that builds a binary search tree of characters. It will read in two lines of characters. It will put the first line in a binary search tree and then print the tree's contents both sideways graphically and in inorder fashion.

It will then read in the second line of input removing the characters of the second line from the binary search tree (if they are there). Finally, it will print the modified tree's contents both sideways and in inorder fashion.

In both cases, there may be duplicate characters in the lines. This is okay. Store all copies of a character in separate nodes. If there are three copies of a character in the first line, and two are removed because they are in the second line, one copy remains in the tree.

Example Input (Bold) and Output
Enter a line of input to insert: marauder
Tree after input.
   u
   r   r
   m   e
   d   ~
   a   a
Inorder traversal: aademrru
Enter a line of input to remove: rummage
Tree after trimming.
   d
   a
Inorder traversal: adr

Specification of the Overall Program
You are writing the insert and remove methods plus two supporting methods for remove.

The program obtains two lines of input from the user. It uses the first line to create a sorted binary search tree of characters. When the characters are all in the tree, it prints the tree sideways so that we can see where the nodes are. It also print the characters in sorted order on one line by traversing the tree in inorder fashion. It then uses the second line of input by deleting the characters in the second line from the tree. When all the characters from the second line have been deleted from the tree, The remaining tree is printed. Note that spaces and tabs are character inputs that will be put into the tree although you don't see them printed. They are allowable input, but beware having them in your test cases because you'll have difficulty seeing them in the sideways tree.

You must use the BTnode<E> class we provide. You must implement the insert and remove methods recursively.

Steps to Start
To create a directory for your tree lab, copy the binarytree directory from the lab directory. That directory contains the btree package as well as a start to this program in BStree.java. The btree package contains BTnode.java and Btree.java which contain the BTreeNode class and binary tree methods. The TreeLines.java file contains the main method that reads the two lines, asks to insert and remove, and prints.

The author discusses the insert and remove methods for binary search trees starting on page 505. His discussion of insert doesn't use recursion although our lectures do. The textbook might be helpful, but it is easier to write these recursively. To get full credit on this assignment, you MUST write them recursively.

<table>
<thead>
<tr>
<th>Binary Search Tree Storage Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(p. 499 of text plus our header node constraint)</td>
</tr>
</tbody>
</table>

- In a binary search tree, the entries of the nodes can be compared with a total order semantics. These two rules are followed for every node n:
  1. Every element in n's left subtree is less than or equal to the element in node n.
  2. Every element in n's right subtree is greater than the element in node n.

The tree has a header node containing null as its element. By the above rules, the tree's contents will be stored in the right subtree of that header node.
Solving the Problem

- Read through all parts of this handout. There are implementation hints at the bottom the page. They'll help.
- Examine what you have and understand problem. Examine the simple version of the program. It compiles and executes cleanly but doesn't seem to do anything. Read the comments. Understand the data representation. Note that the tree has a BTnode<Character> header node containing the null character. Having this header node at the top allows us to add or remove the top node. However, we must skip it when starting any printing.
- Remember iterative enhancement. Enhance the program slowly. Test the newly added feature before moving on to the next part. We've used this approach all semester. You should keep doing it in your remaining courses and whenever you design software.
- Insert characters of first input line into tree. TreeLines calls the insert method to add a character to the tree. Insert calls insertWithNode that takes in a node and adds the character to the appropriate place in the binary search tree. Write the insertWithNode method. The node passed into this method cannot be null. Look ahead before falling off a subtree. Your implementation must be recursive.
- Test what you have. Run your program multiple times with different input. There are no JUnit tests.
- Remove characters of second input line from tree. TreeLines calls remove with the target character to remove. Remove calls removeFromNode to remove the target from the subtree with this node as its parent. Write the removeFromNode method. It needs the dataFromDeletedRightmost and replaceChild methods. Write those as well. As part of its work, removeFromNode calls the other two methods. See below.
- Check your work. Your final output should have two sideways trees and two inorder traversals as shown in the example. The example is a good test case.

Your code should be appropriately commented and indented. Remember that each method should have a good one-line comment. What we are giving you has comments. Be sure your name is at the top after mine. Compile again immediately before submitting your code to be sure that you didn't accidentally type something the compiler doesn't recognize. We will be accepting only the BSTree.java file.

Hints for the methods - their implementations must be recursive.

All connections are made from above. Draw pictures to help you understand.

- insertWithNode The method needs to find where the character fits into the tree and add a new node. Consider the cases of putting it in the left tree and putting it in the right tree. For each of those, insert recursively in an existing tree or add a new node if there is no child in that direction. Don't fall off to null because you won't be able to connect the new node into the tree.
- removeFromNode The method needs to remove one copy of the target from the tree if the target is in the tree. But you need to make sure that the tree stays connected and organized. Consider the four cases of null, looking in the left tree, looking in the right tree, or removing it from this node (and fixing the tree). In fixing the tree, if the left tree is null, just move up the right. That will involve replaceChild. But if the left isn't null, use dataFromDeletedRightmost to replace this node's data with the largest value in the left tree and remove the node where that largest value was found.
- dataFromDeletedRightmost The method finds the largest value in the tree, and both returns it and replaces its place in the tree with its left subtree which may be null. To be able to replace it, you need the parent and can call replaceChild. Where is the largest value in a binary search tree? Discussion around page 509 in the text should help. It's easier than it sounds. Be sure to pass the found value back through the recursive calls.
- replaceChild The method takes a parent, the old child being replaced, and the new child that replaces it. Check whether the old child is the left or right child of the parent and attach the new child appropriately. This method is not recursive. It's just an if-else based on whether two nodes are the same node (== is appropriate) and setting children.