Binary Search Tree – Build and Operate

This project has two parts. In the first we will read a transaction stream and build and operate a Binary Search Tree (BST) based on the stream. This will involve building and operating several trees in succession. In the second part we will balance the BST.

Part 1. Creating a Java program that builds and operates a Binary Search Tree (BST). To do that the program reads a stream of transactions that can perform several types of operations. The transactions include a transaction code and a numeric data node value to be stored in the BST. The transaction types and meanings:

Transaction Code	Transaction Argument	Definition/Action to be taken
I	Integer value	Insert a node into the BST (or establish the BST if it's empty)
F	Integer search value	Find a node with a given value in the tree, return some positive indicator, otherwise return "-1"
R	Integer value	Remove the value from the BST if found – return some positive indicator if found and removed, otherwise "-1" if not found
D	<none>	Display (print) the entire tree in a given format and the number of nodes and height at the time of display
E	<none>	Reset the tree to being Empty

In creating and coding the program:

1. The input is a single transaction text file, with the following description:
   - Contains a series of transactions, one per line
   - No limit on the number of transaction lines, but plan on no more than 1,000
   - Each transaction has the transaction code in the first position, one or more spaces, and an integer value between 0 and 999 (max 3 digits) – ignore all other input on the line
   - The transaction stream drives what the program does – insert, find, remove, etc., and an "E" means reset the BST to being empty and start over with new transactions (if any)
   - Continue processing the transaction stream until there are no more
   - Code reads the input file as an argument to the program.

2. Examples of a transaction stream (your program should ignore comments):

```
I 100      // establish a tree with root node value = 100
I 162      // insert node with value = 162
I 31       //insert 31
I 48       // insert 48
I 92       // insert 92
I 119      // insert 119
F 162      // Find node = 162, return positive indicator
F 131      // Find node = 119, return -1
R 132      // Remove node = 132, return -1
R 48       // Remove node = 48, return positive indicator
D          // Display/print the tree
E          // Reset the tree to being empty
```

(…more transactions could follow, possibly building, displaying, and resetting multiple BSTs.)

3. Use a BST implementation using a linked-type (not array) strategy.

4. Output should include an echo of the transaction ("insert node = 100" for the first line above, for example), the result of the transaction ("162 inserted"), trees displayed via the "D" transaction, and an "end of processing line" after the last transaction is read and processed. Before the end of line processing print/display the total number of trees built and total number displayed.
5. While it is always a good idea to validate the input, we may assume the provided transaction file will be error free (and hopefully in a code format readable by all).

Part 2 . Ensure that the first BST built from the provided input (there will be several in the input stream) is balanced using one of the BST balancing algorithms. - AVL Tree Algorithm.