/\*\*

\* This is the main class that will implement a GUI application

\* showing the results of sorting a BST in ascending and

\* descending order.

\*

\* Name: Nabeel Hussain

\* Class: CMSC 350

\* Professor: Didier Vergamini

\* Project 3

\* Date: 12/02/2016

\*

\* **@author** Nabeel Hussain

\*/

This program for project 3 involves implementing a Binary Search Tree using recursion, and then sorting the Integer or Fraction contents in it by ascending and descending order. The sorted result is shown using a GUI, and the program contains radio buttons for the user to select what type of sort order and numeric type the list should be displayed in.

My program has **five classes:**

The first class is my generic TreeNode<T> class, which is a generic class that defines the nodes making up the BST. It will be used to create my BST of either Integer or Fraction numeric type.

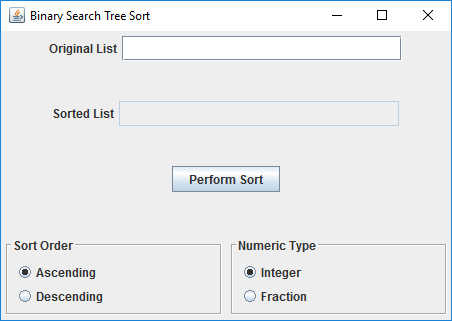
The second class is NumberFormatExpression.java, which extends the Exception class. This exception will be thrown if there are any improperly formatted fractions such as “3/4/8” in the original list that needs to be sorted.

The third class is Fraction.java, which defines the fraction numeric type, in order to hold fractions in the BST. It has a constructor that accepts a valid string representation of a fraction and a toString method. It also implements the Comparable interface, so that it contains a compareTo method. This method will convert the string representation of a fraction into a numeric value, and then compare it with another. Depending on whether the fraction being compared is greater than, equal to, or less than the other object, it will return the appropriate int value.

The fourth class is my generic BinarySearchTree<T> class, which builds the BST containing the user inputted numbers. It also holds the methods that will be used by my GUI class to perform the ascending and descending sort of the list. It also contains a method to insert values into the appropriate location of the BST using recursion, which returns a TreeNode containing the updated tree. The methods to sort the tree in ascending and descending order will take in a TreeNode holding the BST, and then use an in-order traversal to sort it in ascending order. To sort in descending order, it traverses the tree from the other side by accessing the right node first, then the parent node, and then the left node.

The fifth class is my main class called BinarySearchTreeSortGUI, which will be used to display the GUI for this assignment. It will contain the buttons and text fields to meet the functional requirements for this assignment. It will also properly extract the user inputs from a textfield into an array, which will be used by the BinarySearchTree class methods to perform the sorting. The main class will throw and catch any exceptions if there are any non-integer, non-numeric, or improperly formatted fractions, while extracting the inputs into the array.

My GUI:



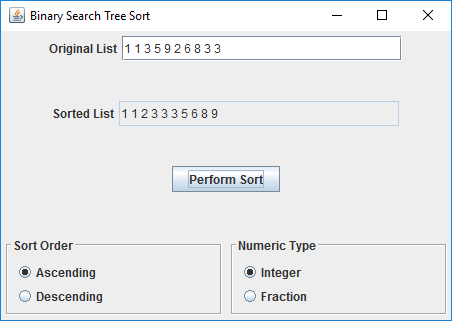
**Test Plan:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input** | **Expected Output**  Sorted List: | **Did Test Pass?** |
| 1 | **Original List = 1 1 3 5 9 2 6 8 3 3**  \*Integer numeric type” | Ascending: 1 1 2 3 3 3 5 6 8 9  Descending: 9 8 6 5 3 3 3 2 1 1 | Y |
| 2 | **Original List = 1 1 3 ab 9 2 6 8 3 3**  \*Integer numeric type”  \*non-numeric input in Integer list\* | JOption Pane Show Message Dialog:  "Non integer Input" | Y |
| 3 | **Original List = 1/2 1/4 3/2 5/3 9/2 1/2**  \*Fraction numeric type” | Ascending: 1/4 1/2 1/2 3/2 5/3 9/2  Descending: 9/2 5/3 3/2 1/2 1/2 1/4 | Y |
| 4 | **Original List = 1/2 3/4/8 3/2 5/3 9/2 1/2**  \*Fraction numeric type”  \* Improperly formatted fraction in Fraction list” | JOption Pane Show Message Dialog:  "Non numeric Input" | Y |
| 5 | **Original List = 1/2 1 3/2 5 9/2 1/2**  \*Fraction numeric type\*  \*Integer values in the Fraction list should be converted into fraction form, in order to be used in the compareTo method. Integers are fractions. | Ascending: 1/2 1/2 1/1 3/2 9/2 5/1  Descending: 5/1 9/2 3/2 1/1 1/2 1/2 | Y |
| 6 | **Original List = 1/2 1 3/2 5 6 2**  \*Integer numeric type”  \*Fraction values in an Integer list cannot convert the fraction back into an Integer” | JOption Pane Show Message Dialog:  "Non integer Input" | Y |
| 7 | **Original List = 1/2 -1/2 3/2 -3/2 9/2**  \*Fraction numeric type”  \*Contains negative fractions” | Ascending: -3/2 -1/2 1/2 3/2 9/2  Descending: 9/2 3/2 1/2 -1/2 -3/2 | Y |
| 8 | **Original List = 1/2 1/-2 3/2 -3/2 9/2**  \*Fraction numeric type”  \*Denominator must always be positive, so it should normalize the fraction by placing the negative sign into the numerator in the fractions when doing the comparison” | Ascending: -3/2 1/-2 1/2 3/2 9/2  Descending: 9/2 3/2 1/2 1/-2 -3/2 | Y |

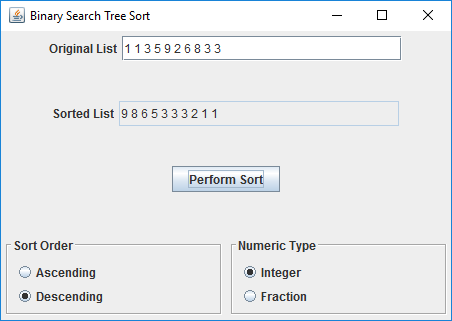
**Screen shots of successful compilation and running for all test cases below:**

**Test Case 1: List containing all valid integers to sort**

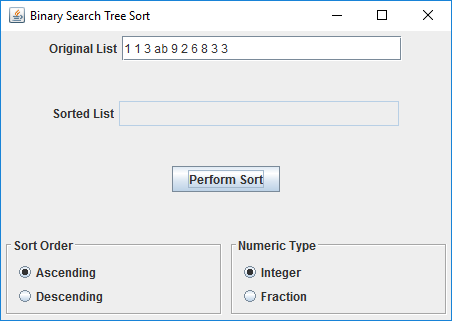
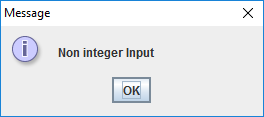
Ascending Order:



Descending Order:

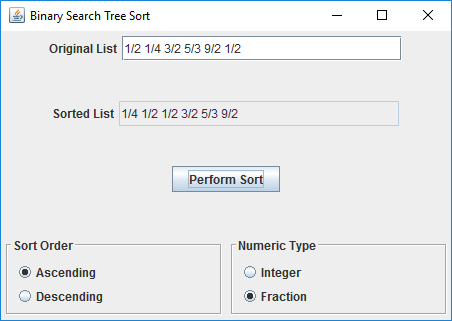


**Test Case 2: List containing “ab” as an integer will throw an exception.**

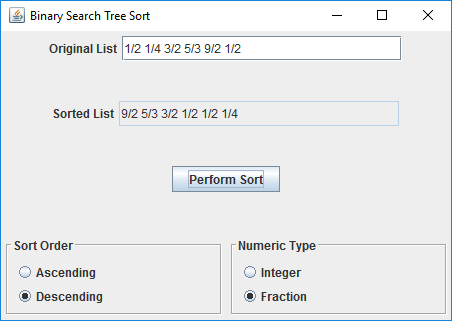


**Test Case 3: List containing all valid fractions to sort**

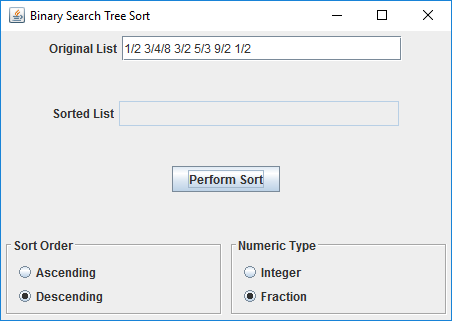
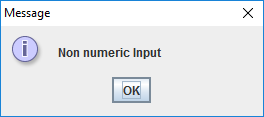
Ascending Order:



Descending Order:

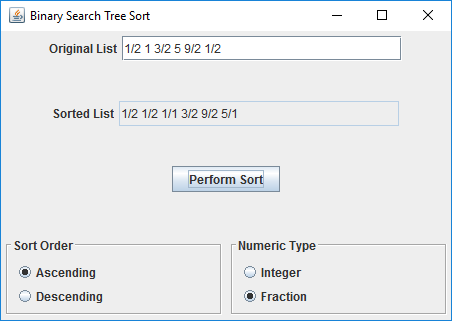


**Test Case 4: List containing improperly formatted fractions will throw an exception.**

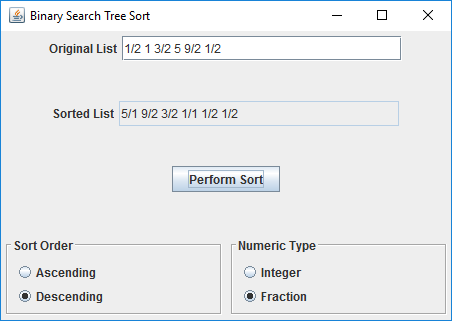


**Test Case 5: Lists being sorted in fraction numeric type should allow integer values, because integers are fractions.**

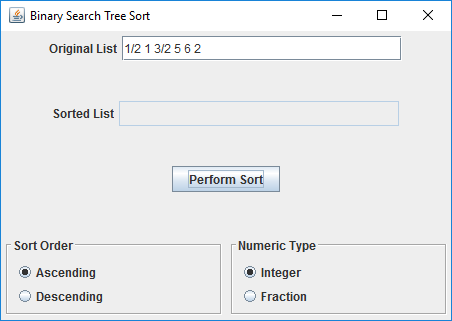
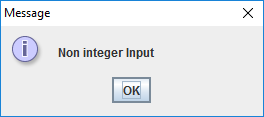
Ascending Order:



Descending Order:

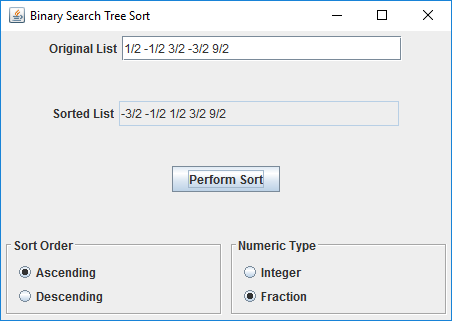


**Test Case 6: List containing fractions cannot be sorted as integer numeric type, and will throw an exception.**

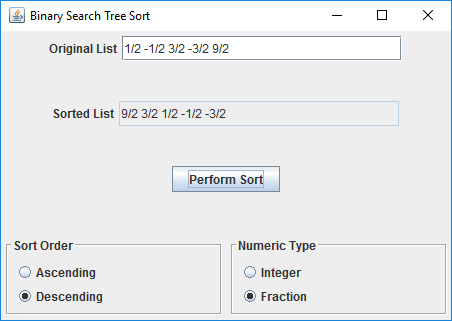


**Test Case 7: Contains negative fractions to sort**

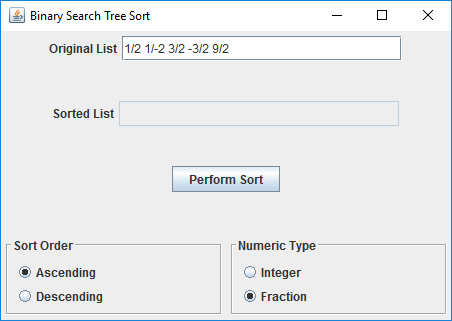
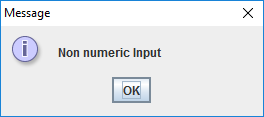
Ascending Order:



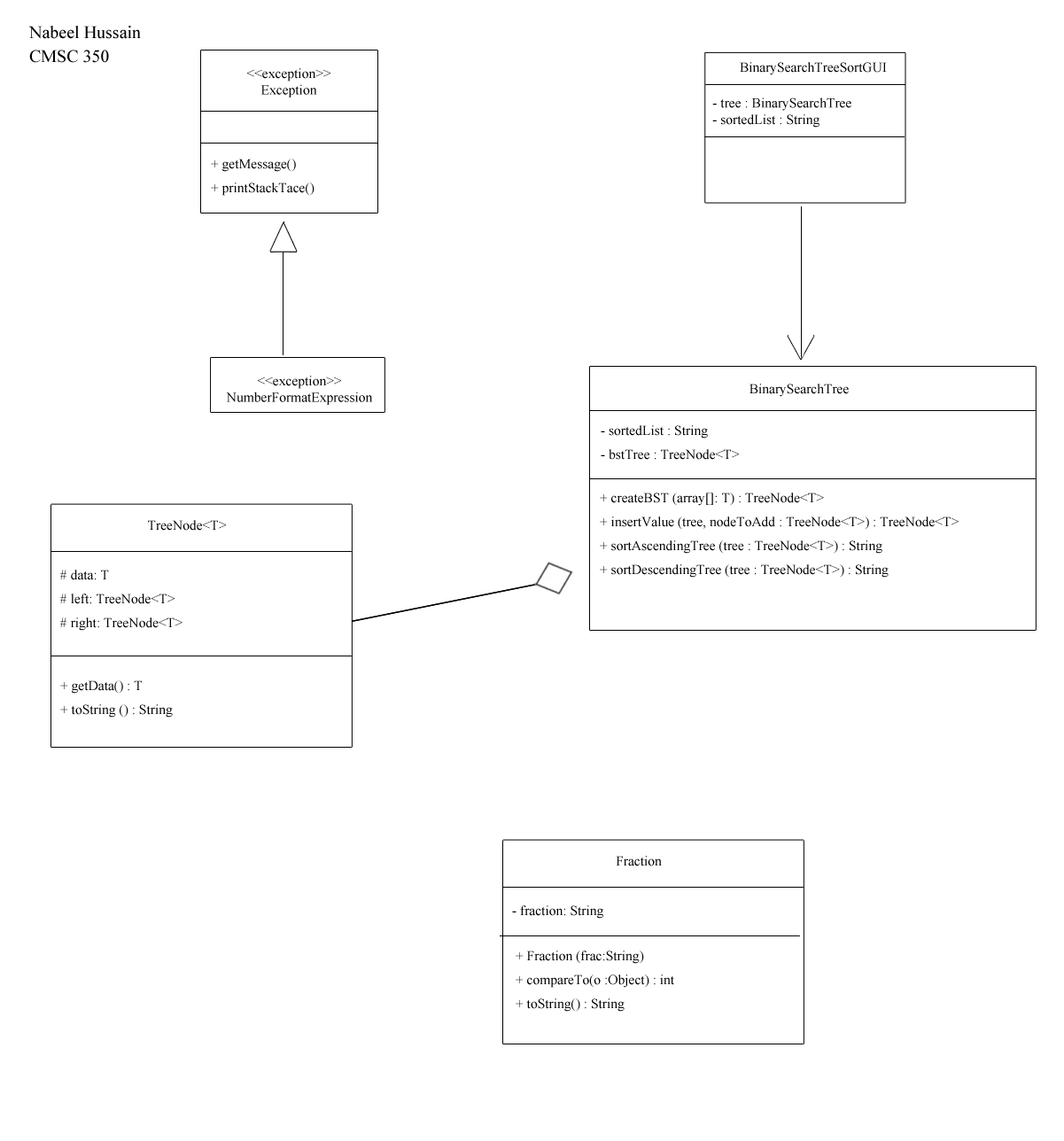
Descending Order:



**Test Case 8: Fractions containing a negative denominator will throw an exception. If there is a negative fraction, the ‘-’ sign must be in the numerator**



**UML Diagram:**



**Lessons Learned:**

While working on Project 3, there were many new concepts and techniques that I learned about building binary search tress, such as using recursion to go about adding new nodes to the tree and traversing them. I found that adding new values to the BST was by far the most challenging part of this assignment, especially when duplicate values needed to be stored in the tree as well. It took a long time for me to figure out how to tackle this part, and how to properly traverse through the BST recursively. However, once I was able to figure out a working solution for adding a node, then I added the stipulation that if a duplicate node is found, it would be placed as a right child in the tree.

Overall, this assignment was the most challenging and time consuming for me so far, but it definitely helped me understand and reinforce all the material I learned from this week and past week’s readings much better.