John Kucera

Prof. Ioan Salomie

CMSC 350 Data Structures and Analysis

20 February 2020

Week 6: Project 3 Solution Description

**Binary Search Tree Sort GUI**

1. **Assumptions, Main design decisions, and error handling**

Like my usual GUIs, I have assumed that anything can be entered. This includes extra spaces, characters, and any other possible tokens. I have also assumed that the only input to be accepted are integers (signed and unsigned) OR fractions of #/# format, decimals excluded, and a single space between each element. With that in mind, any additional white space in between integers will raise a Number Format Exception. For Fractions, a Malformed Fraction Exception is raised when there are additional white spaces between the input fractions. Another important assumption is that choosing one numeric type will make the other numeric type count as invalid input. For example, if type Integer is chosen, input fractions will cause a Number Format Exception to be raised, and if type Fraction is chosen, input integers will cause a Malformed Fraction Exception to be raised. Trailing white space IS allowed and will be bypassed.

Assuming anything can be entered, I have tried to catch each possible error that could be entered through the GUI. This includes:

1. NullPointerException – for cases where the input is empty
2. NumberFormatException – for cases where there is non-numeric input, when additional white space between integers, or a fraction is entered when type integer had been selected
3. MalformedFractionException – for cases where input includes improperly formatted fractions such as 3/4/6, when additional white space is between fractions, or an integer is entered when type fraction had been selected

As for the main design decisions, I decided to use the minimal implementation skeleton provided in Project 3 Indications for my BST generic class. In addition to the suggested components (BSTnode class + constructor, BSTnode root, BST default constructor, insertNode, insertNodeRecursive, inOrderTraversal), I created the methods getLeft() and getRight() to return the left and right nodes (respectively). I also created the methods getAscend() and getDescend() to return the ascending order or descending order of a given BST (respectively). As for my other classes, I created P3GUI to contain the GUI and BST creation, MalformedFractionException as a custom exception constructor class, and Fraction to hold fraction constructor.

The way the program flows is like this: the user selects their numeric type and sort order from the radio buttons, inputs their list of according numeric types, and presses the “Perform Sort” button. In P3GUI, an array of Strings is created which holds the original list. A BST is created using the generics BST class, of either type Integer or of type Fraction.

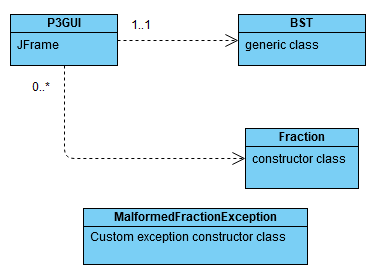
The elements in the original array are inserted into the BST using the insertNode method. The insertNode method uses the insertNodeRecursive method which determines the placement of each value on the BST by comparing them to nodes already on the BST. This is recursive, continuing until there are no values left to insert.

Next, P3GUI uses the getAscend() or getDescend() method depending on which sorting the user selected. If Ascending is selected, getAscend() will use the inOrderTraversal() method which creates a string from in-order traversal through the BST. The string is returned to the GUI through getAscend() and is displayed as the sorted list. On the other hand, if Descending is selected, getDescend() will use the getAscend() method for an ascending string of elements, then turn it into an array to reverse the order of elements. This results in a descending order of elements, which is then turned back into String and then returned to the GUI output.

If numeric type fraction is selected, the Fraction constructor in Fraction.java will be used to create the BST in the GUI. This provides assignments for the numerator, denominator, and value of the whole fraction. There is also a toString() method to return the String format of the fraction and compareTo() method for the sake of comparing the fraction value to other nodes on the BST.

Finally, the GUI checks for duplicate values in the array of elements after the sorting is successful. A JOptionPane is displayed after the check. If duplicate values are present, the JOptionPane will notify the user of this. Keep in mind that even if this is the case, duplicate values are STILL ALLOWED and will not cause an error. Otherwise, the JOptionPane will let the user know that there are no duplicate values present.

1. **UML Class diagram**

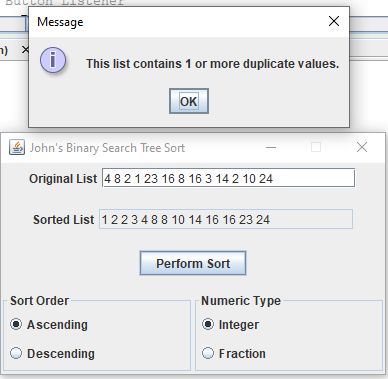


1. **Test Cases**

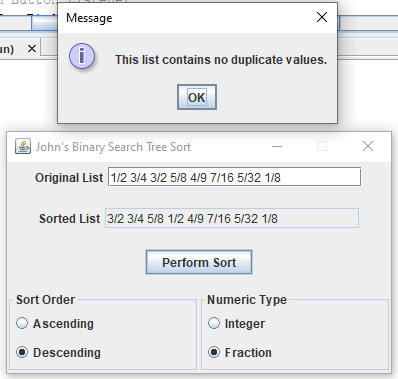
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What aspect is tested** | **Input** | **Expected Output** | **Actual Output** | **Pass/**  **Fail** |
| If the user can close the GUI | Click “X” button on frame | GUI is closed and exited | GUI is closed and exited | PASS |
| Integers, Ascending Order, duplicates included | 4 8 2 1 23 16 8 16 3 14 2 10 24  -Ascending  -Integer | Sorted List: 1 2 2 3 4 8 8 10 14 16 16 23 24  JOptionPane indicates "This list contains 1 or more duplicate values." | Sorted List: 1 2 2 3 4 8 8 10 14 16 16 23 24  JOptionPane indicates "This list contains 1 or more duplicate values." | PASS |
| Integers, Descending Order, duplicates included, trailing white space | 4 8 2 1 23 16 8 16 3 14 2 10 24  (5 spaces at the end)  -Descending  -Integer | Sorted List: 24 23 16 16 14 10 8 8 4 3 2 2 1  JOptionPane indicates "This list contains 1 or more duplicate values." | Sorted List: 24 23 16 16 14 10 8 8 4 3 2 2 1  JOptionPane indicates "This list contains 1 or more duplicate values." | PASS |
| Integers, Ascending Order, no duplicates, trailing white space | 45 -9 600 -23 988 1982 -340  (4 spaces at the end)  -Ascending  -Integer | Sorted List: -340 -23 -9 45 600 988 1982  JOptionPane indicates "This list contains no duplicate values." | Sorted List: -340 -23 -9 45 600 988 1982  JOptionPane indicates "This list contains no duplicate values." | PASS |
| Integers, Descending Order, no duplicates | 45 -9 600 -23 988 1982 -340  -Descending  -Integer | Sorted List: 1982 988 600 45 -9 -23 -340  JOptionPane indicates "This list contains no duplicate values." | Sorted List: 1982 988 600 45 -9 -23 -340  JOptionPane indicates "This list contains no duplicate values." | PASS |
| Fractions, Ascending Order, no duplicates | 1/2 3/4 3/2 5/8 4/9 7/16 5/32 1/8  -Ascending  -Fraction | Sorted List: 1/8 5/32 7/16 4/9 1/2 5/8 3/4 3/2  JOptionPane indicates "This list contains no duplicate values." | Sorted List: 1/8 5/32 7/16 4/9 1/2 5/8 3/4 3/2  JOptionPane indicates "This list contains no duplicate values." | PASS |
| Fractions, Descending Order, no duplicates, trailing white space | 1/2 3/4 3/2 5/8 4/9 7/16 5/32 1/8  (6 spaces at the end)  -Descending  -Fraction | Sorted List: 3/2 3/4 5/8 1/2 4/9 7/16 5/32 1/8  JOptionPane indicates "This list contains no duplicate values." | Sorted List: 3/2 3/4 5/8 1/2 4/9 7/16 5/32 1/8  JOptionPane indicates "This list contains no duplicate values." | PASS |
| Fractions, Ascending Order, duplicates included, trailing white space | 9/8 -3/4 10/900 -600/4 5/5 5/66 5/5  (7 spaces at the end)  -Ascending  -Fraction | Sorted List: -600/4  -3/4 10/900 5/66 5/5 5/5 9/8  JOptionPane indicates "This list contains 1 or more duplicate values." | Sorted List: -600/4  -3/4 10/900 5/66 5/5 5/5 9/8  JOptionPane indicates "This list contains 1 or more duplicate values." | PASS |
| Fractions, Descending Order, duplicates included | 9/8 -3/4 10/900 -600/4 5/5 5/66 5/5  -Descending  -Fraction | Sorted List: 9/8 5/5 5/5 5/66 10/900 -3/4  -600/4  JOptionPane indicates "This list contains 1 or more duplicate values." | Sorted List: 9/8 5/5 5/5 5/66 10/900 -3/4  -600/4  JOptionPane indicates "This list contains 1 or more duplicate values." | PASS |
| Integers, Non-numeric input (1) | 2 3 b 4 7  -Ascending  -Integer | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Integers, Non-numeric input (2) | 2 5 \* + 0 >  -Descending  -Integer | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Integers, input fraction | 29 4/5 67 3 5  -Ascending  -Integer | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Fractions, Non-numeric input (1) | 2/3 a/c 5/6 2/1  -Ascending  -Fraction | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Fractions, Non-numeric input (2) | 100/233 3/& 2/3 9/8  -Descending  -Fraction | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Fractions, input integer | 1/2 4/5 6 7/8  -Ascending  -Fraction | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | PASS |
| Integers, additional white space in between elements | 4 5 89 2 34  -Descending  -Integer | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | Message Dialog Box: "Your list is invalid due to non-numeric input. Please try again." | PASS |
| Fractions, additional white space in between elements | 4/5 6/7 3/4 9/8  -Ascending  -Fraction | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | PASS |
| Fractions, malformed fraction (1) | 5/6 3/4/5 3/2 6/7  -Ascending  -Fraction | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | PASS |
| Fractions,  malformed fraction (2) | 800/9 2/3 8/ 3/4  -Descending  -Fraction | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | Message Dialog Box: "Your list is invalid due to a malformed fraction. Please try again." | PASS |
| No input | No input  press Perform Sort | Message Dialog Box: "Please enter a list of integers OR a list of fractions." | Message Dialog Box: "Please enter a list of integers OR a list of fractions." | PASS |

1. **Screenshots of program execution**

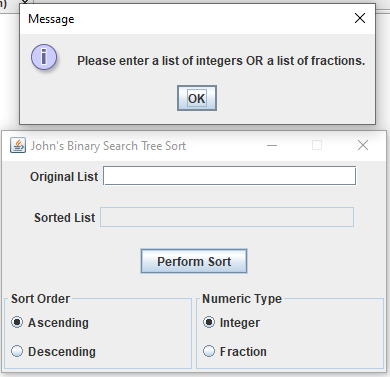
GUI, sorting a list of integers in ascending order, with message indicating duplicates:



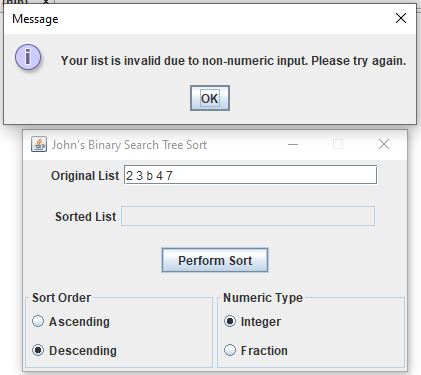
GUI, sorting a list of fractions in descending order, with message indicating no duplicates:



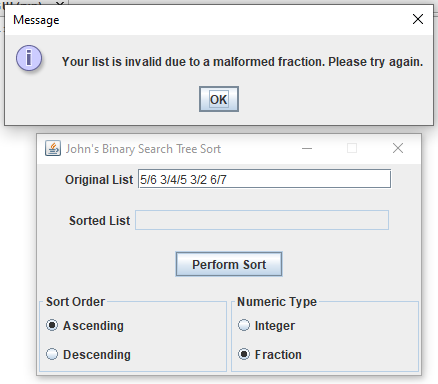
Empty input error:



Non-numeric input error:



Malformed fraction error:



1. **Lessons learned**

This was my second experience with trees, and it was much better than the first considering that I fully understand the concepts of in-order, pre-order, and post-order traversal. Understanding when you should use which direction of traversal is vastly important to using trees. I was able to manage this BST with no problem.

However, this was my first time using Generics (I’m sure I had not learned about them in any previous classes – or maybe I just forgot). While it took an effort to grasp at first, it became quite easy to understand the concept of generics after going through many helpful webpages and some textbook resources that have things to say on the subject. It took me phrasing like this to understand it: “You can have a class that constructs a BST using only Fraction objects. Or, you can have a GENERIC class that constructs a BST using EITHER Fraction or Integer objects.” With my growing understanding of the concept of “implements” and “extends”, generics became simple to grasp pretty quickly.

On a note that is not related to BSTs, I became very aware of the specificity of Exception catching when writing this program. It was interesting to see which exceptions would be caught first in cases where two of them apply to one error. For example, if I input a single letter “a” when choosing Fraction type, I will get a Malformed Fraction Exception. While it is a malformed fraction, you’d think most users would prefer for it to be a Number Format Exception. This time around, I did not bother sorting that out because an error is raised either way and both are applicable. Still, in programs where it matters more, I need to be ready to prepare more specific exceptions to be caught. This is also why it’s important to show the user WHERE they made the error in the message dialog box.