/\*\*

\*

\* File: CMSC 350 project 3

\* Author: Bedemariam Degef

\* Date: November 30, 2019

\* Purpose: This program uses a binary search tree \* in order to create a sorted list

\*

\*/

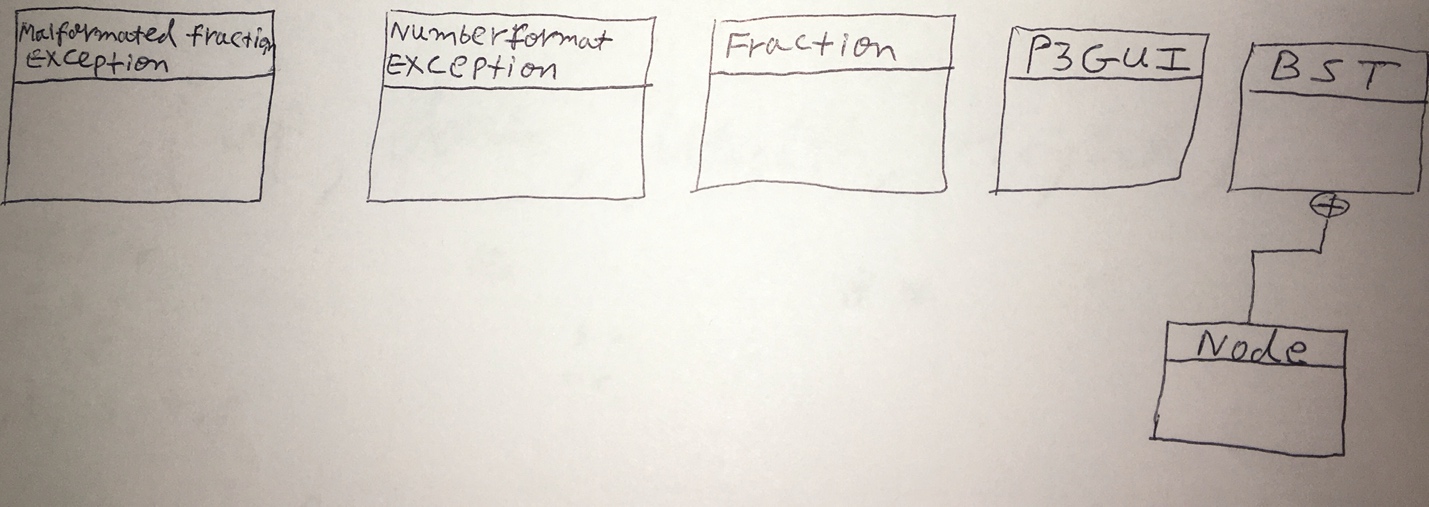
1. **Assumptions, main design decisions, error handling;**

My assumption after reading the instruction was that I will be creating a binary search tree from the input and sort it in ascending or descending order. The user generated exceptions should be implemented similar to the previous project. Reading the assigned topics and other online resources have helped me understand what I needed to do.

There are going to be five classes. The first class is BST which is a generic class. It will contain the node subclass class and is used to build the binary search tree. It will perform an in-order tree traversal that generates and returns a string that contains the tree elements in sorted order. It will allow for duplicate entries and will be written using recursion. The second class is Fraction which defines fractions. It will have a constructor that accepts a string representation of a fraction and a toString method. Since it will also implement the comparable interface, it will contain the compareTo method. The other two classes will be MalformedFractionException which handles exceptions for improper fractions and NumberFormatException for invalid user inputs. The last class is P3GUI which will define and implement the GUI.

There are going to be two exceptions used in this project. InvalidInputException is thrown and handled for any illegal inputs. NullPointerException is handled for empty input values. In addition, MalformedFractionException is caught and handled for improper fraction inputs.

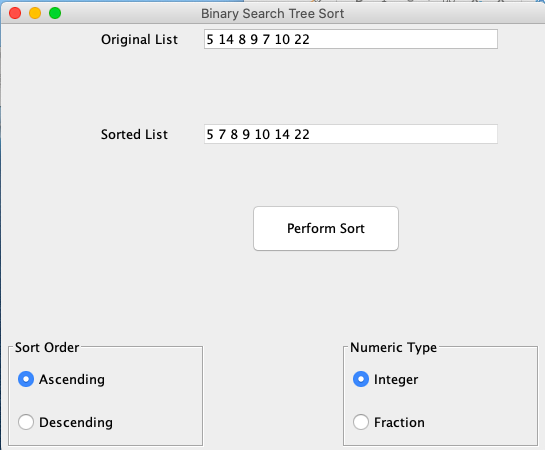
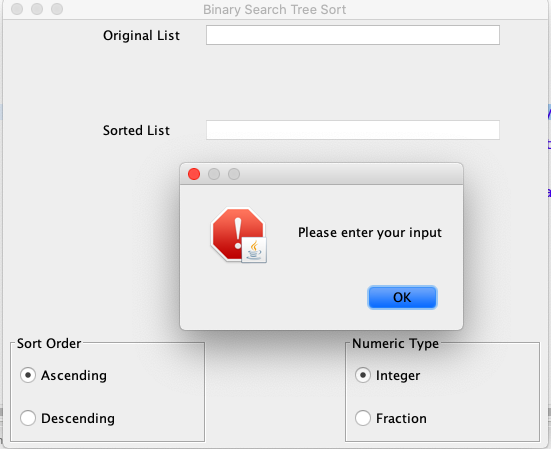
1. **A UML class diagram that includes all classes you wrote. Do not include predefined classes. You need only include the class name for each individual class, not the variables or methods;**

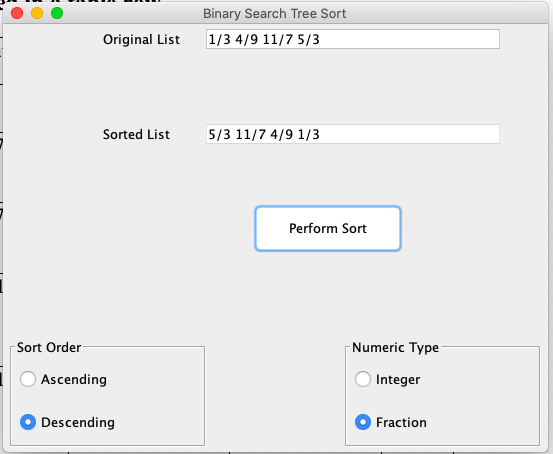
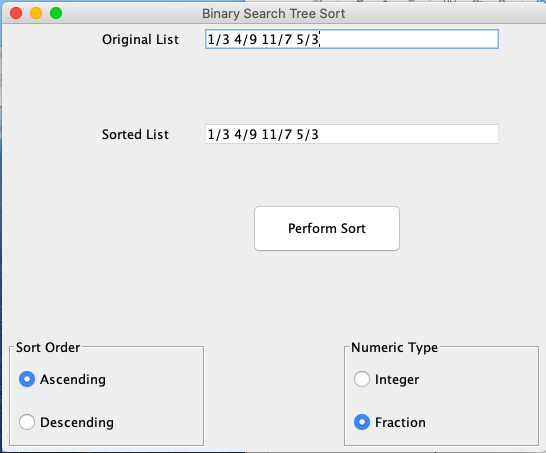
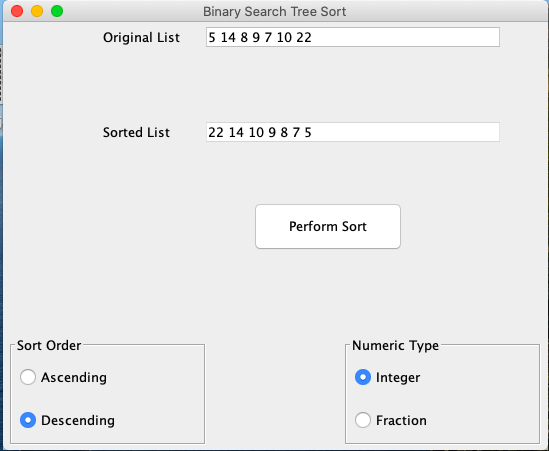
****

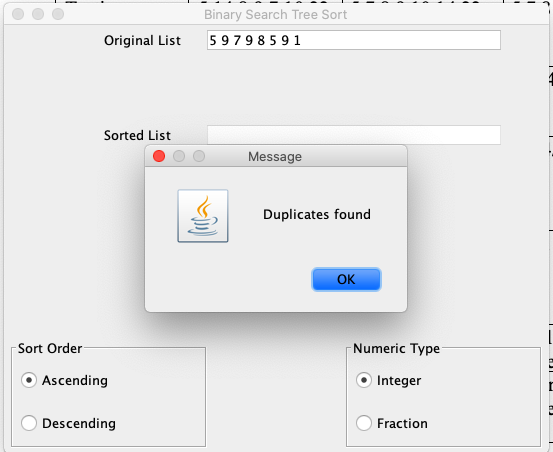
1. **A table of test cases including the test cases that you have created to test the program. The table should have 5 columns indicating (i) what aspect is tested, (ii) the input values, (iii) the expected output, (iv) the actual output and (v) if the test case passed or failed. Each test case will be defined in a table row.**

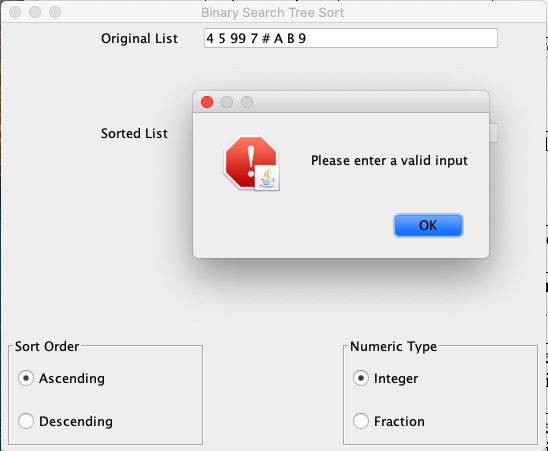
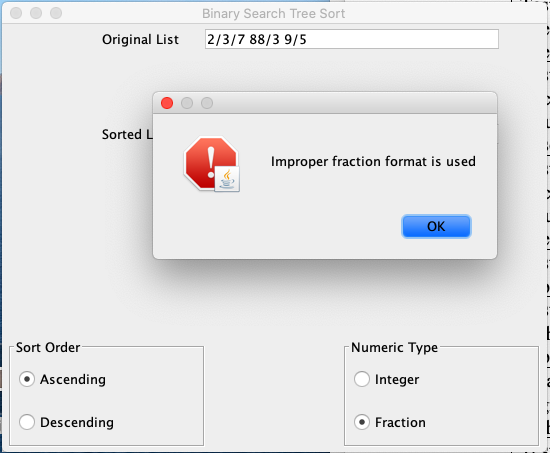
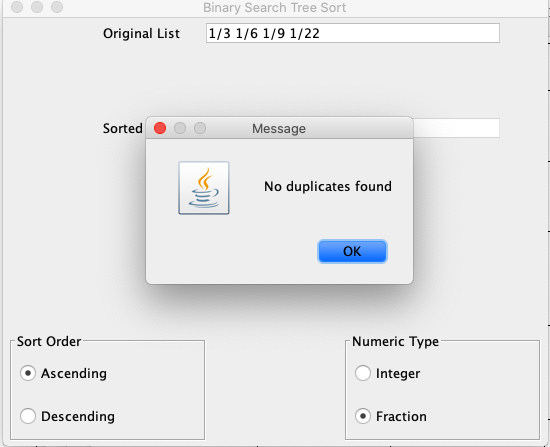
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **What aspect is tested** | **The input values** | **The expected output** | **The actual output** | **Pass or fail** |
| Testing with no inputs |  | Please enter your input | Please enter your input | Passed |
| Testing integer values (Ascending) | 5 14 8 9 7 10 22 | 5 7 8 9 10 14 22 | 5 7 8 9 10 14 22 | Passed |
| Testing integer values (Descending) | 5 14 8 9 7 10 22 | 22 14 10 9 8 7 5 | 22 14 10 9 8 7 5 | Passed |
| Testing fraction values (Ascending) | 1/3 4/9 11/7 5/3 | 1/3 4/9 11/7 5/3 | 1/3 4/9 11/7 5/3 | Passed |
| Testing fraction values (Descending) | 1/3 4/9 11/7 5/3 | 5/3 11/7 4/9 1/3 | 5/3 11/7 4/9 1/3 | Passed |
| Testing with duplicates | 5 9 7 9 8 5 9 1 | Duplicates found | Duplicates found | Passed |
| Testing without duplicates | 1/3 1/6 1/9 1/22 | No duplicates found | No duplicates found | Passed |
| Invalid tokens beginning with a digit | 4 5 99 7 # A B 9 | Please enter a valid input | Please enter a valid input | Passed |
| Testing non-numeric values | 4 @ 99 7 100 1 | Please enter a valid input | Please enter a valid input | Passed |
| Testing malformed fraction inputs | 2/3/7 88/3 9/5 | Improper fraction format is used | Improper fraction format is used | Passed |

1. **Relevant screenshots of program execution;**

****

****

****

****

**(5) Lessons learned from the project;**

I have gained more experience working with binary search trees. It was also a learning experience implementing the comparable interface and working on a generic class. All in all, it was a good learning experience, and I look forward to the next project.