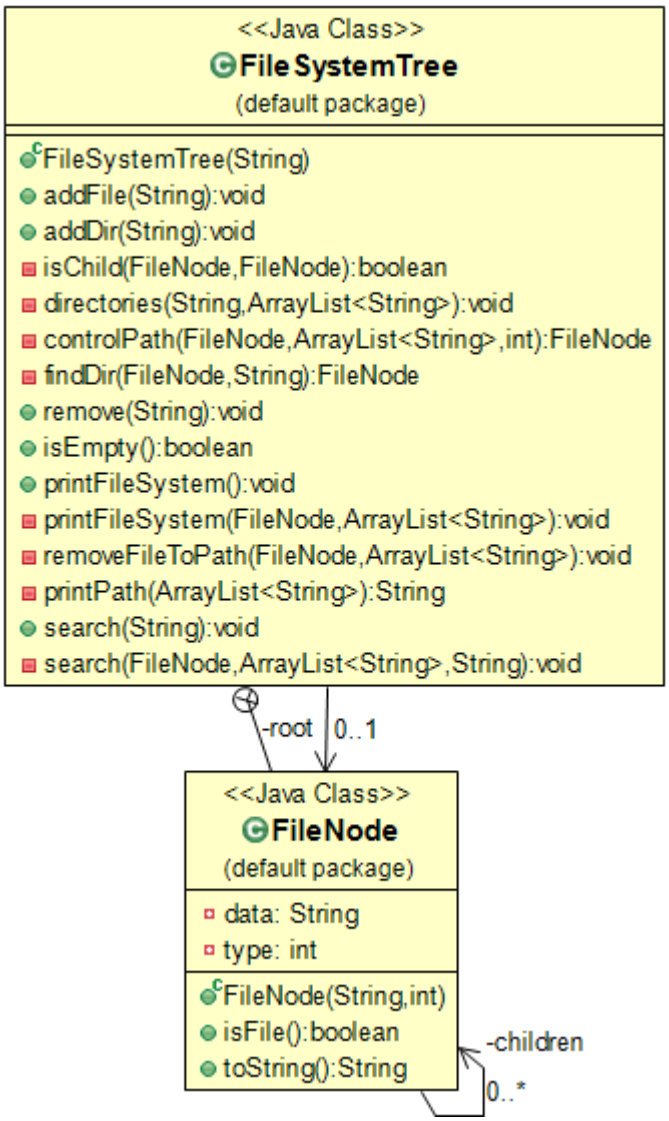
GIT Department of Computer Engineering
CSE 222/505- Spring 2020
Homework 5 Report

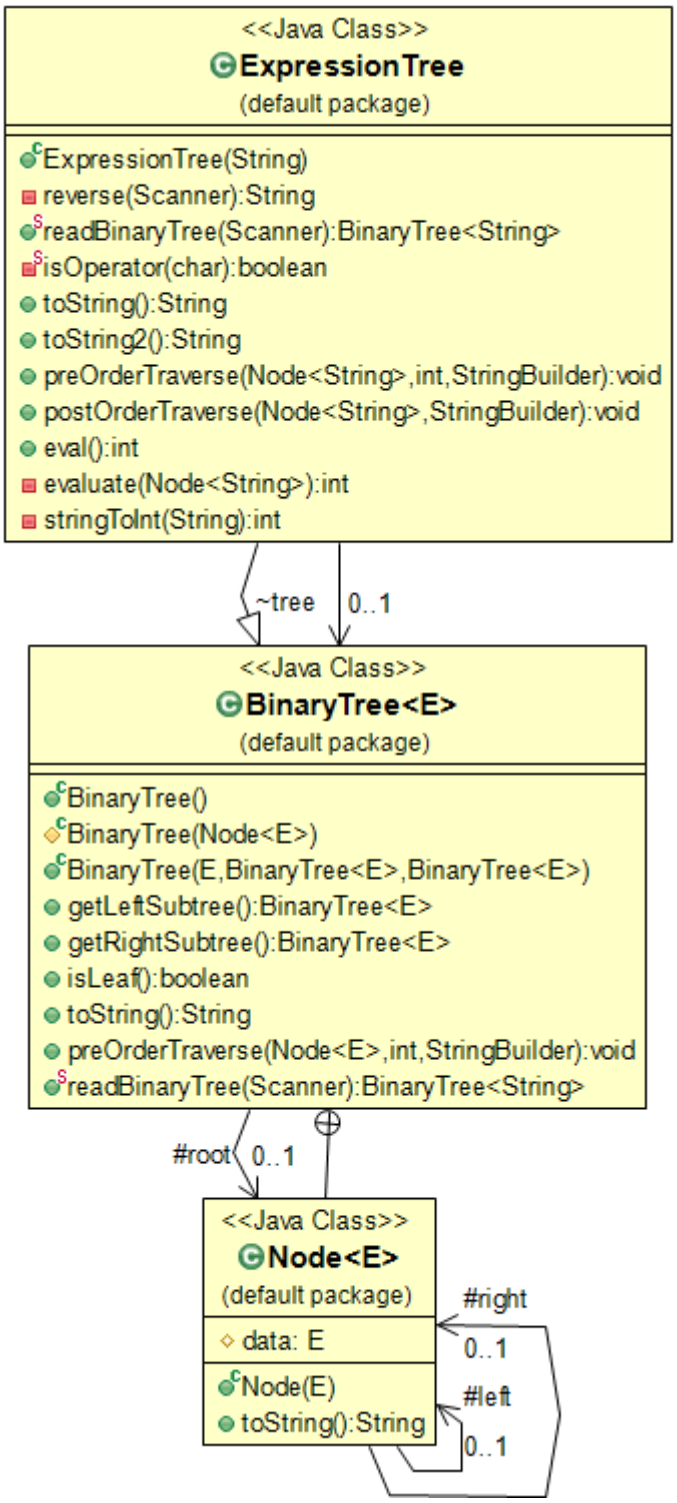
Şeyda ÖZER
171044023

CLASS DIAGRAMS

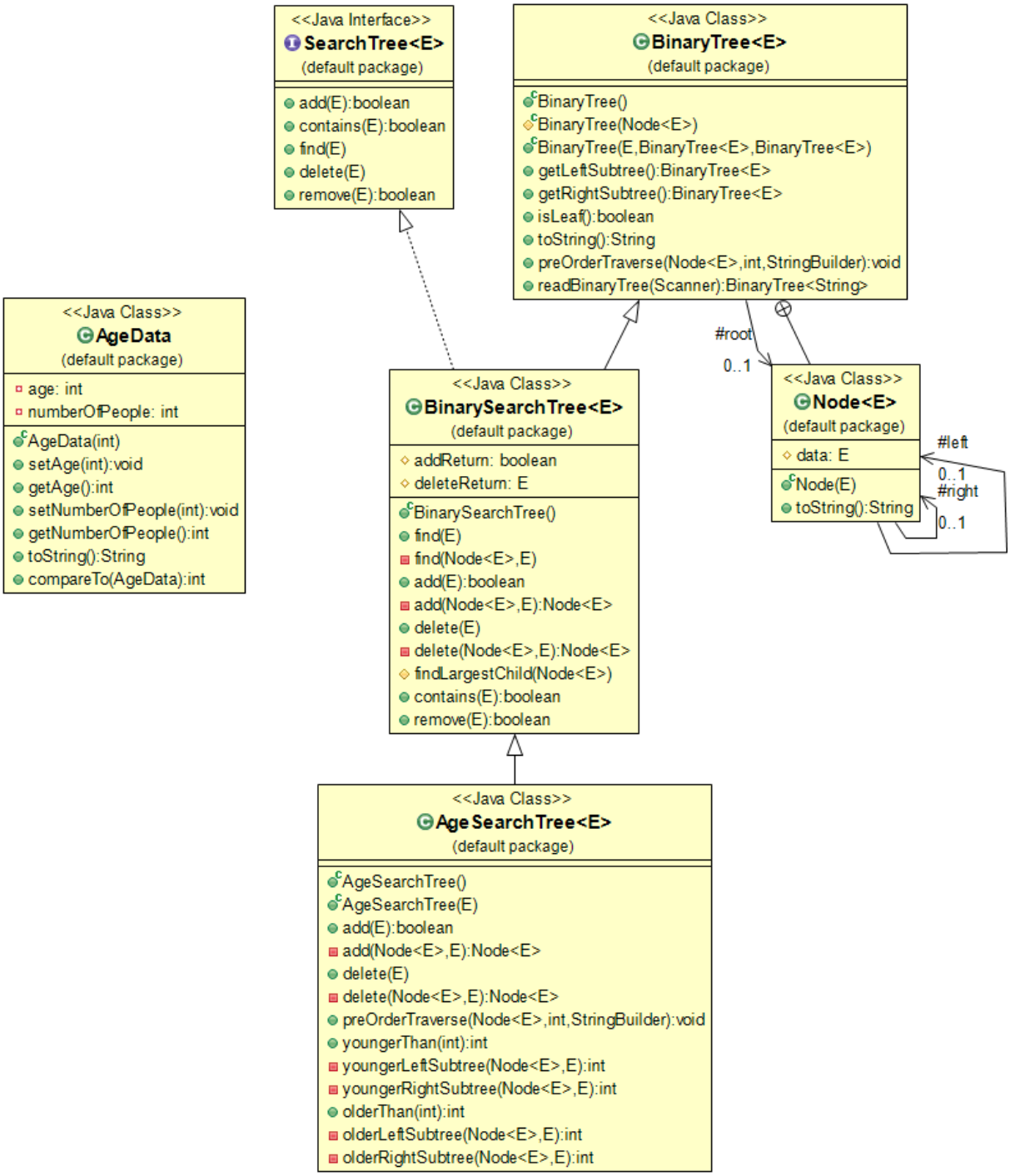
Q1:



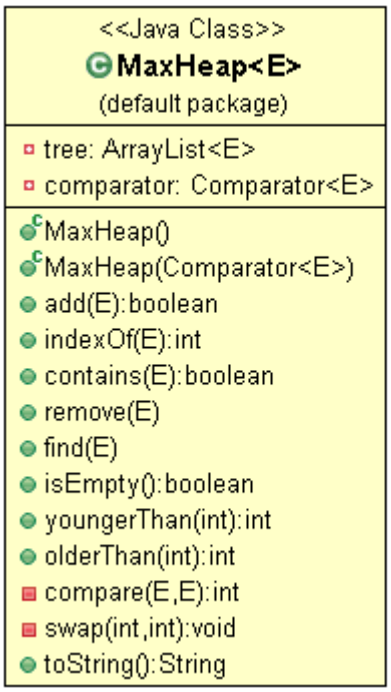
Q2:

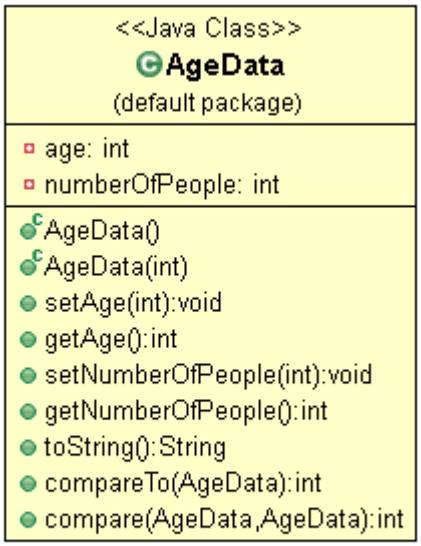

<<Java Class>>
ExpressionTree
(default package)
ExpressionTree(String)
reverse(Scanner):String
readBinaryTree(Scanner):BinaryTree<String>
isOperator(char):boolean
toString():String
toString2():String
preOrderTraverse(Node<String>,int,StringBuilder):void
postOrderTraverse(Node<String>,StringBuilder):void
eval():int
evaluate(Node<String>):int
stringToInt(String):int
~tree
0..1
<<Java Class>>
BinaryTree<E>
(default package)
BinaryTree()
BinaryTree(Node<E>)
BinaryTree(E,BinaryTree<E>,BinaryTree<E>)
getLeftSubtree():BinaryTree<E>
getRightSubtree():BinaryTree<E>
isLeaf():boolean
toString():String
preOrderTraverse(Node<E>,int,StringBuilder):void
readBinaryTree(Scanner):BinaryTree<String>
#root
0..1
<<Java Class>>
Node<E>
(default package)
data: E
Node(E)
toString():String
#right
0..1
#left
0..1

Q3:


<<Java Interface>>
SearchTree<E>
(default package)
add(E):boolean
contains(E):boolean
find(E)
delete(E)
remove(E):boolean
<<Java Class>>
BinaryTree<E>
(default package)
BinaryTree()
BinaryTree(Node<E>)
BinaryTree(E,BinaryTree<E>,BinaryTree<E>)
getLeftSubtree():BinaryTree<E>
getRightSubtree():BinaryTree<E>
isLeaf():boolean
toString():String
preOrderTraverse(Node<E>,int,StringBuilder):void
readBinaryTree(Scanner):BinaryTree<String>
<<Java Class>>
AgeData
(default package)
age: int
numberOfPeople: int
AgeData(int)
setAge(int):void
getAge():int
setNumberOfPeople(int):void
getNumberOfPeople():int
toString():String
compareTo(AgeData):int
<<Java Class>>
BinarySearchTree<E>
(default package)
addReturn: boolean
deleteReturn: E
BinarySearchTree()
find(E)
find(Node<E>,E)
add(E):boolean
add(Node<E>,E):Node<E>
delete(E)
delete(Node<E>,E):Node<E>
findLargestChild(Node<E>)
contains(E):boolean
remove(E):boolean
#root
0..1
<<Java Class>>
Node<E>
(default package)
data: E
Node(E)
toString():String
#left
0..1
#right
0..1
<<Java Class>>
AgeSearchTree<E>
(default package)
AgeSearchTree()
AgeSearchTree(E)
add(E):boolean
add(Node<E>,E):Node<E>
delete(E)
delete(Node<E>,E):Node<E>
preOrderTraverse(Node<E>,int,StringBuilder):void
youngerThan(int):int
youngerLeftSubtree(Node<E>,E):int
youngerRightSubtree(Node<E>,E):int
olderThan(int):int
olderLeftSubtree(Node<E>,E):int
olderRightSubtree(Node<E>,E):int

Q4:


<<Java Class>>
MaxHeap<E>
(default package)
tree: ArrayList<E>
comparator: Comparator<E>
MaxHeap()
MaxHeap(Comparator<E>)
add(E):boolean
indexOf(E):int
contains(E):boolean
remove(E)
find(E)
isEmpty():boolean
youngerThan(int):int
olderThan(int):int
compare(E,E):int
swap(int,int):void
toString():String


<<Java Class>>
AgeData
(default package)
age: int
numberOfPeople: int
AgeData()
AgeData(int)
setAge(int):void
getAge():int
setNumberOfPeople(int):void
getNumberOfPeople():int
toString():String
compareTo(AgeData):int
compare(AgeData,AgeData):int

PROBLEM SOLUTION APPROACH

Q1 – FileSystemTree

Firstly, I need to implement FileNode class to handle the nodes of the tree. This node can be created either for a file or a directory. I solved this problem by adding a private int variable to the class. The value of this variable is 1 for the directory node and 2 for the file node. In this way, I have determined my node type. Thus, I was able to check the current file type while implementing the add and remove methods. I used LinkedList for the children of the node in the FileNode Class. While I was traversing the tree, I went to the leaf node for every child of the root. In this tree, the leaf nodes are the file node and directory nodes that has no child. In the search and print methods, for each leaf node the whole branch needs to be printed. I used the helper functions in these methods. I use an ArrayList in these methods. I add the name of the node to the list and then remove the name of the printed node. Thus, each node is printed once.

Q2 – ExpressionTree

Firstly, I used BinaryTree object for data field because I have to use readBinaryTree method in the ExpressionTree constructor and the method's return type is BinaryTree. I override readBinaryTree method according to the prefix expression. For this reason, if the taken String is prefix expression, I reversed the expression to postfix expression, in constructor. Later I created methods that are necessary for this assignment.

Q3 – AgeSearchTree

Firstly, I implemented the BinaryTree class, BinarySearchTree class and SearchTree interface. Later I created the AgeData class. This class implements Comparable interface bacause it must be comparable. Objects of AgeData class are compared according to the age data, so compareTo method compares ages of two object. The data type of AgeSearchTree class must be Comparable because it is extends BinarySearchTree class. We used the AgeSearchTree class with AgeData class. For this reason, I when I implemented necessary methods for this assignment, I did the operations according to the AgeData objects.

Q4 – MaxHeap

This time, AgeData class also implements Comparator interface because while we creating a heap, we will compare the number of people have the same age. Namely, we are not comparing the elements according to the their natural ordering. Therefore I override the compare method of the Comparator class. Later I created a compare method for a MaxHeap class , so this class can make the necessary comparisons for heap structure.

TEST CASES

Q1 – FileSystemTree

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass / Fail
1	Removing a directory or file, when the file system has no directory or file except for root directory	remove("root/directory")	Throws an exception. The file system has no file or directory except for root drectory.	As Expected	Pass
2	Removing root directory	remove("root")	Throws an exception. The root directory cannot removed.	As Expected	Pass
3	Adding a directory to the directory that is not in the system	add("root/third_directory/directory")	Throws an exception. The path cannot be found.	As Expected	Pass
4	Adding a file to the directory that is not in the system	add("root/third_directory/file.txt")	Throws an exception. The path cannot be found.	As Expected	Pass
5	Adding a directory to the directory but there is a directory with the same name in the directory	add("root/first_directory")	Throws an exception. The directory is aldready added.	As Expected	Pass
6	Adding a file to the directory but there is a file with the same name in the directory	add("root/first_file.txt")	Throws an exception. The file is aldready added.	As Expected	Pass
7	Searching with string that is not in any directory / file on the file system.	search("new")	Nothing	As Expected	Pass
8	Searching with the valid string	search("f")	Prints all directories that contains "f"	As Expected	Pass
9	Removing the directory/file that is not in the system	remove("root/third_directory")	Throws an exception. The path cannot be found.	As Expected	Pass
10	Removing a empty directory	remove("root/first_directory/new_directory/files")	The files directory is removed.	As Expected	Pass

11	Removing a directory that is not empty	remove("root/members")	Ask the user whether to remove or not. If the answer is yes, the directory will be removed.	As Expected	Pass
12	Removing a file	remove("root/first_file.txt")	The firs_file.txt file is removed.	As Expected	Pass

Q2 – ExpressionTree

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass / Fail
1	Creating an object with preorder expression	new ExpressionTree("+ + 10 * 5 15 20 ")	The object is created without any problems.	As Expected	Pass
2	Creating an object with postorder expression	new ExpressionTree("+ "10 5 15 * + 20 +")	The object is created without any problems.	As Expected	Pass
3	Printing the tree as a preorder traversal	toString()	Printing the tree as a preorder traversal	As Expected	Pass
4	Printing the tree as a postorder traversal	toString2()	Printing the tree as a postorder traversal	As Expected	Pass
5	Evaluating a tree with preorder expression	eval()	The result of the method is true	As Expected	Pass
6	Evaluating a tree with postorder expression	eval()	The result of the method is true	As Expected	Pass

Q3 – AgeSearchTree

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass / Fail
1	Adding a element to tree.	add(new AgeData(10))	The element is added.	As Expected	Pass
2	Adding the same element to tree again.	add(new AgeData(10))	The number of people field of the element increase 1.	As Expected	Pass
3	Using youngerThan method	youngerThan(15)	This method returns the number of people younger than 15	As Expected	Pass
4	Using olderThan method.	olderThan(10)	This method returns the number of people older than 10	As Expected	Pass

5	Removing an object that has number of people data greater than 1.	remove(new AgeData(5))	The number of people data of the object decrease 1.	As Expected	Pass
6	Removing an element from the tree.	remove(new AgeData(15))	The element is removed.	As Expected	Pass
7	Removing an element that is not in the tree.	remove(new AgeData(80))	There is no change in the tree.	As Expected	Pass
8	Finding an element.	find(new AgeData(20))	The element is found.	As Expected	Pass
9	Finding an element that is not in the tree.	find(new AgeData(80))	The element is not found.	As Expected	Pass
10	Control of the order of the elements	toString()	Printing the tree as a pretorder traversal	As Expected	Pass

Q4 – MaxHeap

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass / Fail
1	Adding a element to tree.	add(new AgeData(10))	The element is added.	As Expected	Pass
2	Adding the same element to tree again.	add(new AgeData(10))	The number of people field of the element increase 1.	As Expected	Pass
3	Using youngerThan method	youngerThan(15)	This method returns the number of people younger than 15	As Expected	Pass
4	Using olderThan method.	olderThan(10)	This method returns the number of people older than 10	As Expected	Pass
5	Removing an object that has number of people data greater than 1.	remove(new AgeData(10))	The number of people data of the object decrease 1.	As Expected	Pass
6	Removing an element from the tree.	remove(new AgeData(15))	The element is removed.	As Expected	Pass
7	Removing an element that is not in the tree.	remove(new AgeData(80))	There is no change in the tree.	As Expected	Pass
8	Finding an element.	find(new AgeData(50))	The element is found.	As Expected	Pass
9	Finding an element that is not in the tree.	find(new AgeData(80))	The element is not found.	As Expected	Pass
10	Control of the order of the elements.	toString()	Printing the tree according to the heap structure	As Expected	Pass

RUNNING AND RESULTS

Q1 – FileSystemTree

Test Case 1:

Test Data:

```
try{
    myFileSystem.remove("root/first_directory");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before removing :

```
root
```

Result:

```
java.lang.UnsupportedOperationException: The file system has no file or directory except for root directory.
```

Test Case 2:

Test Data:

```
try{
    myFileSystem.remove("root");
} catch (Exception e) {
    System.out.println(e);
}
```

Result:

```
java.lang.UnsupportedOperationException: The root cannot removed!
```

Test Case 3:

Test Data:

```
try{
    myFileSystem.addDir("root/third_directory/directory");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before adding:

```
root / first_directory
root / first_file.txt
```

Result:

```
java.lang.NullPointerException: The path cannot be found!
```

Test Case 4:

Test Data:

```
try{
    myFileSystem.addFile("root/third_directory/file.txt");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before adding:

```
root / first_directory
root / first_file.txt
```

Result:

```
java.lang.NullPointerException: The path cannot be found!
```

Test Case 5:

Test Data:

```
myFileSystem.addDir("root/first_directory");
try{
    myFileSystem.addDir("root/first_directory");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before adding:

```
root / first_directory
root / first_file.txt
```

Result:

```
java.lang.UnsupportedOperationException: This directory is already added.
```

Test Case 6:

Test Data:

```
myFileSystem.addFile("root/first_file.txt");
try{
    myFileSystem.addFile("root/first_file.txt");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before adding:

```
root / first_directory
root / first_file.txt
```

Result:

```
java.lang.UnsupportedOperationException: This file is already added.
```

Test Case 7:

Test Data:

```
System.out.println("Search file or directory names including “new”");
myFileSystem.search("new");
```

The tree before searching:

```
root / first_directory
root / first_file.txt
```

Result:

```
Search file or directory names including “new”
There are no results.
```

Test Case 8:

Test Data:

```
System.out.println("Search file or directory names including “f”");
myFileSystem.search("f");
```

The tree before searching:

```
root / first_directory / new_directory / files
root / first_file.txt
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Result:

```
Search file or directory names including “f”
dir - root / first_directory
dir - root / first_directory / new_directory / files
file - root / first_file.txt
dir - root / photos / family
file - root / members / file1.txt
file - root / members / file2.txt
```

Test Case 9:

Test Data:

```
try{
    myFileSystem.remove("root/third_directory");
} catch (Exception e) {
    System.out.println(e);
}
```

The tree before removing:

```
root / first_directory / new_directory / files
root / first_file.txt
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Result:

```
java.lang.NullPointerException: The path cannot be found!!
```

Test Case 10:

Test Data:

```
myFileSystem.remove("root/first_directory/new_directory/files");
```

The tree before removing:

```
root / first_directory / new_directory / files
root / first_file.txt
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Result:

```
root / first_directory / new_directory
root / first_file.txt
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Test Case 11:

Test Data:

```
myFileSystem.remove("root/members");
```

The tree before removing:

```
root / first_directory / new_directory
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Result:

The answer is yes:

```
y
root / first_directory / new_directory
root / first_file.txt
root / photos / family
```

The answers is no:

```
n
root / first_directory / new_directory
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Test Case 12:

Test Data:

```
myFileSystem.remove("root/first_file.txt");
```

The tree before removing:

```
root / first_directory / new_directory / files
root / first_file.txt
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Result:

```
root / first_directory / new_directory
root / photos / family
root / members / file1.txt
root / members / file2.txt
```

Q2 – ExpressionTree

Test Case 1:

Test Data:

```
ExpressionTree expTree1 = new ExpressionTree("+ + 10 * 5 15 20");
```

Result: The tree is created.

Test Case 2:

Test Data:

```
ExpressionTree expTree2 = new ExpressionTree("10 5 15 * + 20 +");
```

Result: The tree is created.

Test Case 3:

Test Data:

```
System.out.println(expTree1.toString());

System.out.println(expTree2.toString());
```

Result:

```
+ + 10 * 5 15 20

+ 20 + * 15 5 10
```

Test Case 4:

Test Data:

```
System.out.println(expTree1.toString2());

System.out.println(expTree2.toString2());
```

Result:

```
10 5 15 * + 20 +

20 15 5 * 10 + +
```

Test Case 5:

Test Data:

```
System.out.println("The result of the evaluating of the tree1");
System.out.println(expTree1.eval());
```

Result:

```
The result of the evaluating of the tree1
105
```

Test Case 6:

Test Data:

```
System.out.println("The result of the evaluating of the tree2");
System.out.println(expTree2.eval());
```

Result:

```
The result of the evaluating of the tree2
105
```

Q3 – AgeSearchTree

Test Case 1:

Test Data:

```
ageTree.add(new AgeData(10));
```

Result:

The element is added.

```
10 - 1
```

Test Case 2:

Test Data:

```
ageTree.add(new AgeData(10));
ageTree.add(new AgeData(10));
```

Result:

```
10 - 2
```

Test Case 3:

Test Data:

```
System.out.print("The number of people younger than 15: ");
System.out.println(ageTree.youngerThan(15));
```

The tree (before this operation):

```
10 - 2
5 - 2
null
null
20 - 1
15 - 1
null
null
null
```

Result:

```
The number of people younger than 15: 4
```

Test Case 4:

Test Data:

```
System.out.print("The number of people older than 10: ");
System.out.println(ageTree.olderThan(10));
```

The tree (before this operation):

```
10 - 2
5 - 2
null
null
20 - 1
15 - 1
null
null
null
```

Result:

```
The number of people older than 10: 2
```

Test Case 5:

Test Data:

```
System.out.println("The element 5 is removing");
ageTree.remove(new AgeData(5));
```

The tree (before this operation):

```
10 - 2
5 - 2
null
null
20 - 1
15 - 1
null
null
null
```

Result:

```
The element 5 is removing
10 - 2
5 - 1
null
null
20 - 1
15 - 1
null
null
null
```

Test Case 6:

Test Data:

```
System.out.println("The element 15 is removing");
ageTree.remove(new AgeData(15));
```

The tree (before this operation):

```
10 - 2
5 - 1
null
null
20 - 1
15 - 1
null
null
null
```

Result:

```
The element 15 is removing
10 - 2
5 - 1
null
null
20 - 1
null
null
```

Test Case 7:

Test Data:

```
System.out.println("The element 80 is removing");
ageTree.remove(new AgeData(80));
```

The tree (before this operation):

```
10 - 2
5 - 1
null
null
20 - 1
null
null
```

Result:

```
The element 80 is removing
10 - 2
5 - 1
null
null
20 - 1
null
null
There is no change in the tree.
```

Test Case 8:

Test Data:

```
System.out.println(ageTree.find(new AgeData(20)));
System.out.println();
```

The tree (before this operation):

```
10 - 2
5 - 1
null
null
20 - 1
null
null
```

Result:

```
20 - 1
```

Test Case 9:

Test Data:

```
if(ageTree.find(new AgeData(80)) == null)
    System.out.println("The element(80) is not found.");
```

The tree (before this operation):

```
10 - 2
5 - 1
null
null
20 - 1
null
null
```

Result:

```
The element(80) is not found.
```

Test Case 10:

Test Data:

```
//Control of the order of the elements
System.out.println(ageTree);
```

Result:

```
10 - 2
5 - 2
null
null
20 - 1
15 - 1
null
null
null
```

Q4 – MaxHeap

Test Case 1:

Test Data:

```
heap.add(new AgeData(10));
```

Result:

The element is added.

```
10 - 1
```

Test Case 2:

Test Data:

```
heap.add(new AgeData(10));
heap.add(new AgeData(10));
```

Result:

```
10 - 2
```

Test Case 3:

Test Data:

```
System.out.print("The number of people younger than 15: ");
System.out.println(heap.youngerThan(15));
```

The tree (before this operation):

```
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```

Result:

```
The number of people younger than 15: 4
```

Test Case 4:

Test Data:

```
System.out.print("The number of people older than 10: ");
System.out.println(heap.olderThan(10));
```

The tree (before this operation):

```
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```

Result:

```
The number of people older than 10: 3
```

Test Case 5:

Test Data:

```
System.out.println("The element 10 is removing");
heap.remove(new AgeData(10));
```

The tree (before this operation):

```
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```

Result:

```
The element 10 is removing
5 - 2
10 - 1
70 - 1
50 - 1
15 - 1
```

Test Case 6:

Test Data:

```
System.out.println("The element 15 is removing");
heap.remove(new AgeData(15));
```

The tree (before this operation):

```
5 - 2
10 - 1
70 - 1
50 - 1
15 - 1
```

Result:

```
The element 15 is removing
5 - 2
10 - 1
70 - 1
50 - 1
```

Test Case 7:

Test Data:

```
System.out.println("The element(80) is removing");
heap.remove(new AgeData(80));
System.out.print(heap);
System.out.println("There is no change in the tree.\n");
```

The tree (before this operation):

```
5 - 2
10 - 1
70 - 1
50 - 1
```

Result:

```
The element(80) is removing
5 - 2
10 - 1
70 - 1
50 - 1
There is no change in the tree.
```

Test Case 8:

Test Data:

```
System.out.println(heap.find(new AgeData(50)));
System.out.println();
```

The tree (before this operation):

```
5 - 2
10 - 1
70 - 1
50 - 1
```

Result:

```
50 - 1
```

Test Case 9:

Test Data:

```
if(heap.find(new AgeData(80)) == null)
    System.out.println("The element(80) is not found.");
```

The tree (before this operation):

```
5 - 2
10 - 1
70 - 1
50 - 1
```

Result:

```
The element(80) is not found.
```

Test Case 10:

Test Data:

```
System.out.println(heap);
```

Result:

```
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```