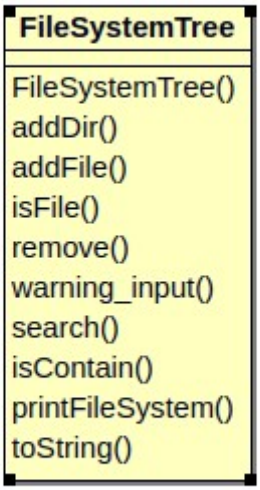
GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework 4 Report

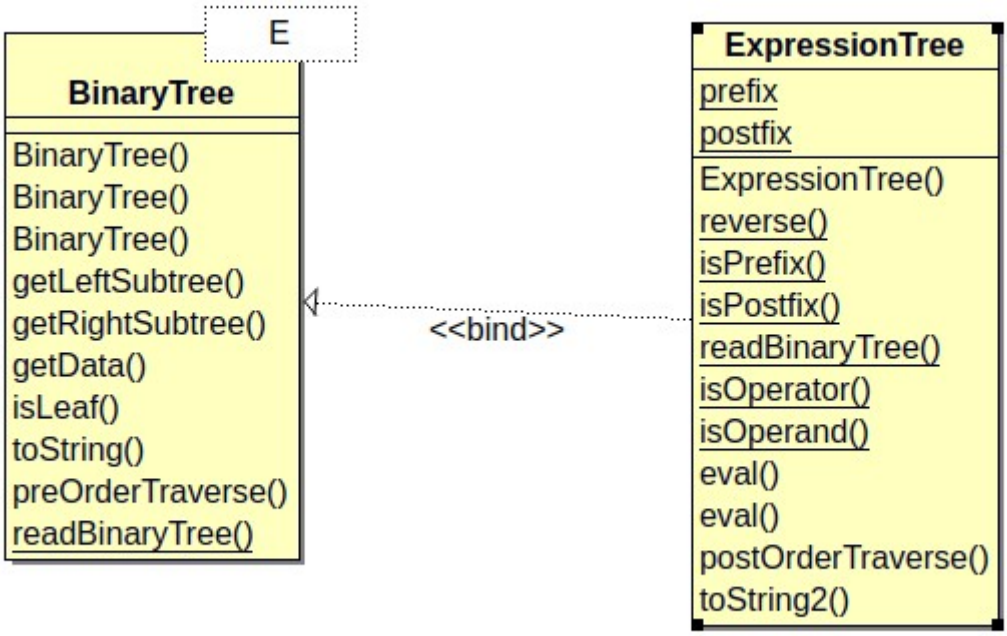
Muharrem Ozan Yeşiller
171044033

1) Class Diagrams

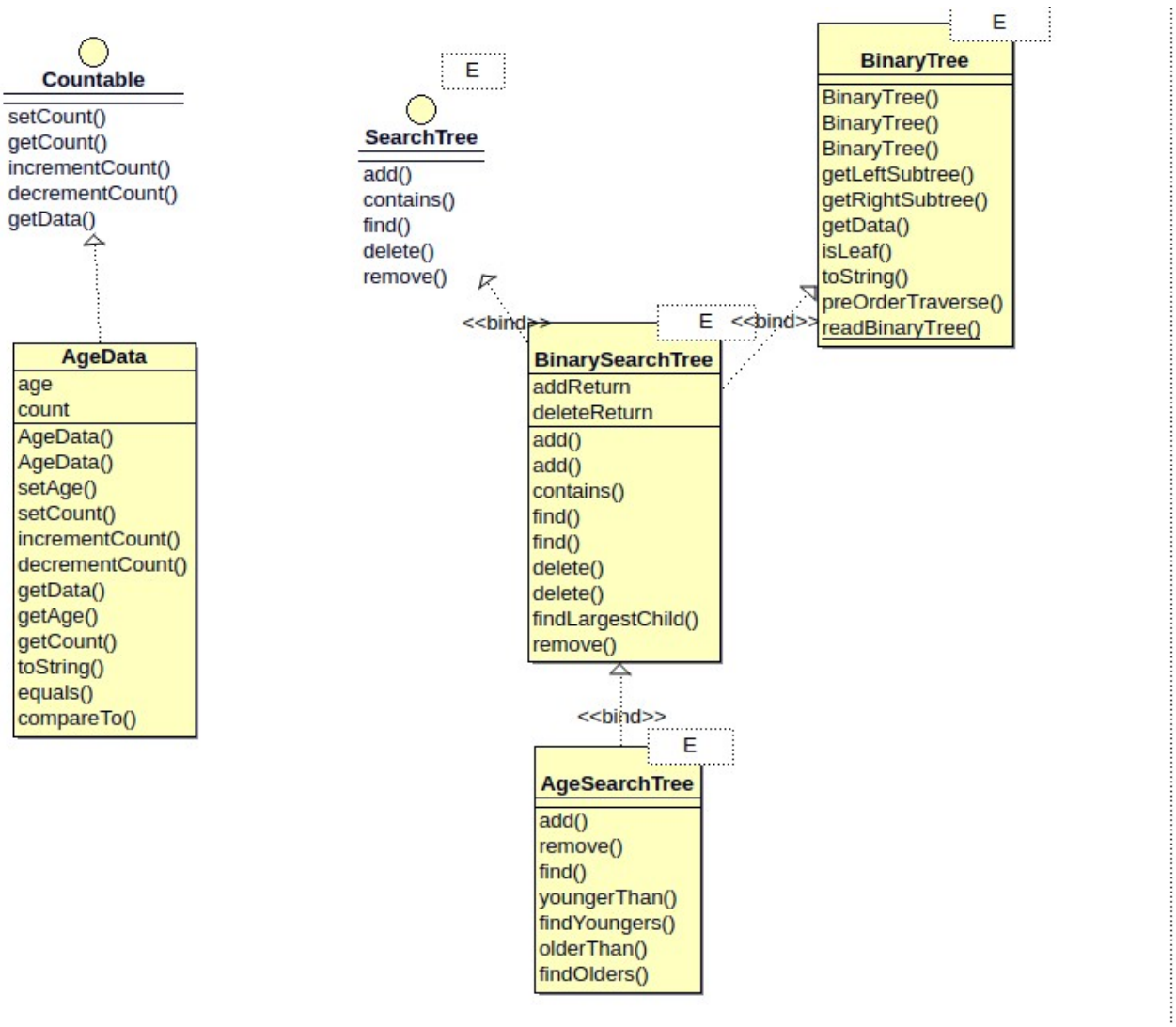
Question1:



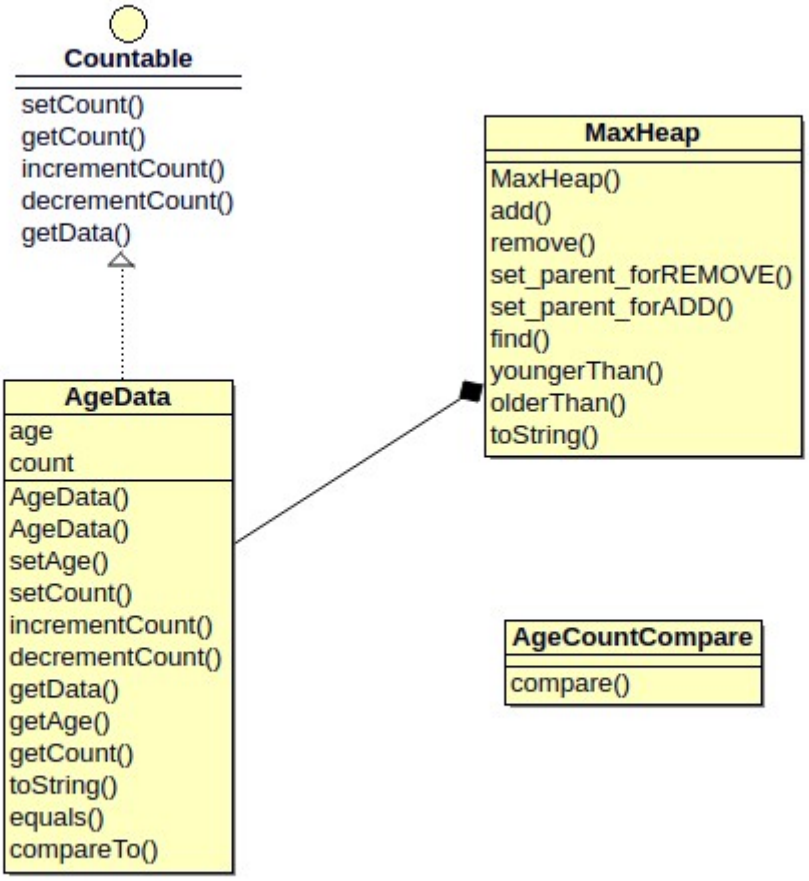
Question2:



Question3:


Countable
setCount()
getCount()
incrementCount()
decrementCount()
getData()
AgeData
age
count
AgeData()
AgeData()
setAge()
setCount()
incrementCount()
decrementCount()
getData()
getAge()
getCount()
toString()
equals()
compareTo()
E
SearchTree
add()
contains()
find()
delete()
remove()
<<bind>>
E
BinaryTree
BinaryTree()
BinaryTree()
BinaryTree()
getLeftSubtree()
getRightSubtree()
getData()
isLeaf()
toString()
preOrderTraverse()
readBinaryTree()
E
<<bind>>
BinarySearchTree
addReturn
deleteReturn
add()
add()
contains()
find()
find()
delete()
delete()
findLargestChild()
remove()
<<bind>>
E
AgeSearchTree
add()
remove()
find()
youngerThan()
findYoungers()
olderThan()
findOlders()

Question4:


Countable
setCount()
getCount()
incrementCount()
decrementCount()
getData()
MaxHeap
MaxHeap()
add()
remove()
set_parent_forREMOVE()
set_parent_forADD()
find()
youngerThan()
olderThan()
toString()
AgeData
age
count
AgeData()
AgeData()
setAge()
setCount()
incrementCount()
decrementCount()
getData()
getAge()
getCount()
toString()
equals()
compareTo()
AgeCountCompare
compare()

2) Problem Solutions Approach

Question1:

The designed class handles the file and directory system. Firstly, when creating the object, the root folder is created. Files or folders added later are added according to the path given in this root folder. The correctness of the path is checked in both additions here because it is not possible to create a folder in a file. If there is an error, the exception is thrown. When adding files or folders, folders on the road that are not on the system are added to the system. User approval must also be obtained for file or folder deletion. Because there may be other files in a folder, and all of them will be deleted.In this system, searching is also possible. All file or folder paths containing the searched string are displayed.

Question2:

The prefix and postfix expression given in this section is converted into a binary tree structure. An estimate of the expression is made during creation prefix or postfix? If the expression is probably a prefix expression, the read Binary method is called. If this prediction is wrong, the program will already throw an exception in this method. If the expression is postfix, the opposite of the expression is sent to the read Binary method. Likewise, if this possible postfix expression is not exactly a postfix expression, the exception is thrown and handle. If this statement is in the reading stage, if the statement is postfix, the reverse of the statement is created with the priority being the right tree. The expression prefix is created as a first left tree. There is also a method of calculating this expression. The root is accepted as the process and the left of the root is right and the right is calculated as the right value recusively.

Question3:

In this section, there is a data structure that sheds people into binary search tree by age. If an object added to the class exists in the binary search tree, the object's counter is increased. A counter is also reduced in deletion.
The use of counters here is a sign of countability. Therefore, we can instantiate this class with every class we can write by implementing a countable and comparable interface. Thus, it can implement this class in a generic structure, without being distracted by the object oriented programming principle. I have instantiated and tested this class with the AgeData class. We talked about adding and removing.

Our finding method takes an object and returns it if it finds it.

The youngerThan () method takes an age parameter and returns the sum of the counters of individuals with an age smaller than this tree structure.

The olderThan () method takes an age parameter and returns the sum of the counters of individuals with an age bigger than this tree structure.

Question4:

Here we dealt with the age binary search tree with the heap data structure. This section is written not working generic but work with AgeData.
In this part, the root of our heap will always be the object with the largest number of people relative to the sub-trees. If the element to be added is included in the data structure, the counter is increased and an update is made from the root of the tree to the top root. If there is no object, the classic ending is done. The deletion algorithm works similarly, if there is more than 1 object, the counter is reduced and the data structure is updated.

Our finding method takes an object and returns it if it finds it.

The youngerThan () method takes an age parameter and returns the sum of the counters of individuals with an age smaller than this tree structure.

The olderThan () method takes an age parameter and returns the sum of the counters of individuals with an age bigger than this tree structure.

3) Test Cases

Question1:

Test Scenario	Test Steps	Pass/Fail

Add new directory	addDir("root/first_directory");
	
	addDir("root/second_directory");
	
	addDir("root/second_directory/new_directory");
		PASS

Add new file	addFile("root/first_directory/new_file.txt");
	
	addFile("root/second_directory/new_directory/new_file.doc");
		PASS

Search file or directory	search("new");	**PASS**
Print file system	printFileSystem();	**PASS**
remove	remove("root/first_directory");	**PASS**

Question2:

Test Scenario	**Test Steps**	**Pass/ Fail**(Prefix)	**Pass/ Fail**(Postfix)
Read expression	new ExpressionTree ("+ + 10 * 5 15 20"); new ExpressionTree ("10 5 15 * + 20 +");	**PASS**	**PASS**
Postorder traverse and toString2() method	Tree1.toString2 () Tree2.toString2 ()	**PASS**	**PASS**
evaluate	Tree1.eval() Tree2.eval()	**PASS**	**PASS**

Question3:

Test Scenario	Test Steps	Pass/Fail

Add object that it don’t has or already has	add(new AgeData(10));
	
	add(new AgeData(20));
	
	add(new AgeData(5));
	
	add(new AgeData(15));
	
	add(new AgeData(10));
		PASS

Find number of younder than	youngerThan(15)	**PASS**
Find number of older than	olderThan(7)	**PASS**
find	find(new AgeData(10))	**PASS**

Remove object, data structure has one or more piece this object	remove(new AgeData(10));
	
	remove(new AgeData(20));
		PASS

Question4:

Test Scenario	Test Steps	Pass/Fail

Add object that it don’t has or already has	add(new AgeData(10));
	
	add(new AgeData(5));
	
	add(new AgeData(70));
	
	add(new AgeData(10));
	
	add(new AgeData(50));
	
	add(new AgeData(5));
	
	add(new AgeData(15));
		PASS

Find number of younder than	youngerThan(15);	**PASS**
Find number of older than	olderThan(7);	**PASS**
find	find(new AgeData(10));	**PASS**

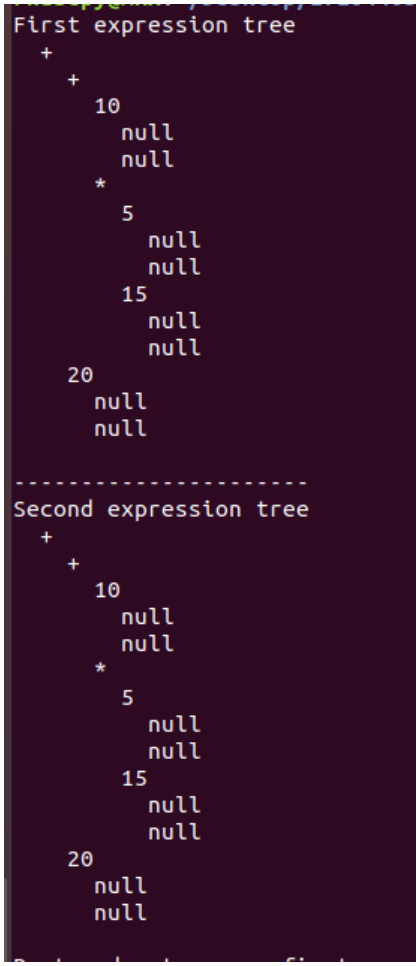
Remove object, data structure has one or more piece this object	remove(new AgeData(10));
	
	remove(new AgeData(15));
		PASS

4) Running command and results

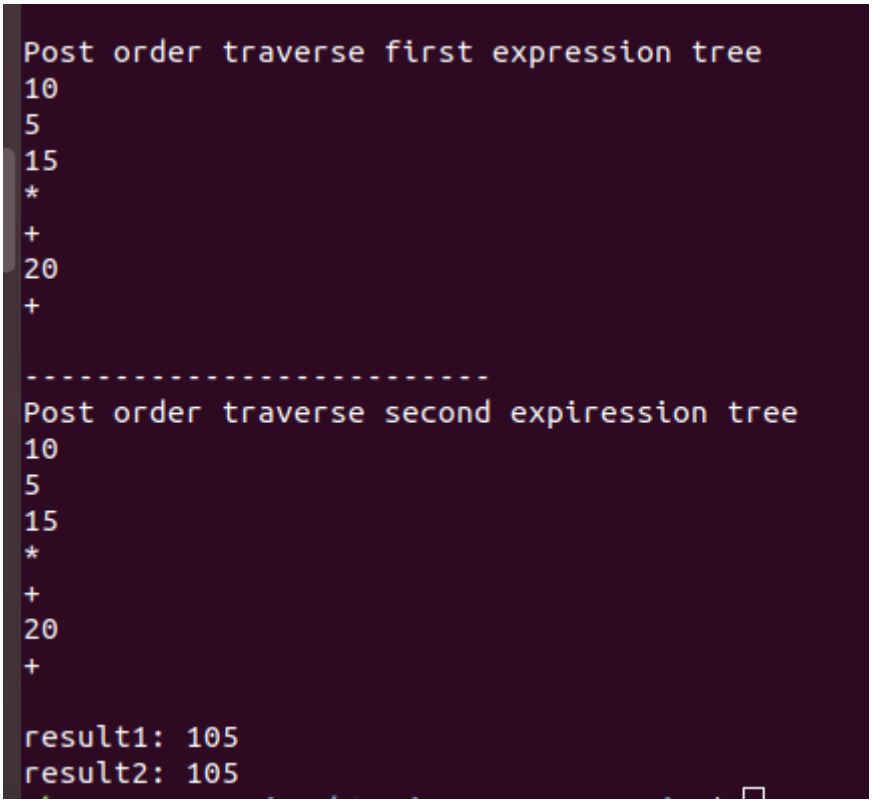
Question1:

```
Content of root:
dir - root/first_directory/
dir - root/second_directory/

Content of first_directory:
file - root/first_directory/new_file.txt/

Content of second_directory:
dir - root/second_directory/new_directory/

Content of new_directory:
file - root/second_directory/new_directory/new_file.doc/


file - root/first_directory/new_file.txt/
dir - root/second_directory/new_directory/
file - root/second_directory/new_directory/new_file.doc/

Are you sure you want to delete the root/first_directory? [y/n]
y
Content of root:
dir - root/second_directory/

Content of second_directory:
dir - root/second_directory/new_directory/

Content of new_directory:
file - root/second_directory/new_directory/new_file.doc/
```

Question2:


First expression tree
+
+
10
null
null
*
5
null
null
15
null
null
20
null
null

Second expression tree
+
+
10
null
null
*
5
null
null
15
null
null
20
null
null


Post order traverse first expression tree
10
5
15
*
+
20
+

Post order traverse second expiression tree
10
5
15
*
+
20
+
result1: 105
result2: 105

Question3:

```
ageTree.toString() method call:
10 - 2
5 - 1
null
null
20 - 1
15 - 1
null
null
null

ageTree.youngerThan(15) call:
3
ageTree.olderThan(7) call:
4
ageTree.find(new AgeData(10) call:
10 - 2
Removing 10 and 20...
10 - 1
5 - 1
null
null
15 - 1
null
null
```

Question4:

```
heap.toString() method call:
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1

heap.youngerThan(15) call:
4
heap.olderThan(7) call:
5
heap.find(new AgeData(10) call:
10 - 2
10 and 15 removing...
5 - 2
10 - 1
70 - 1
50 - 1
```