GIT Department of Computer Engineering

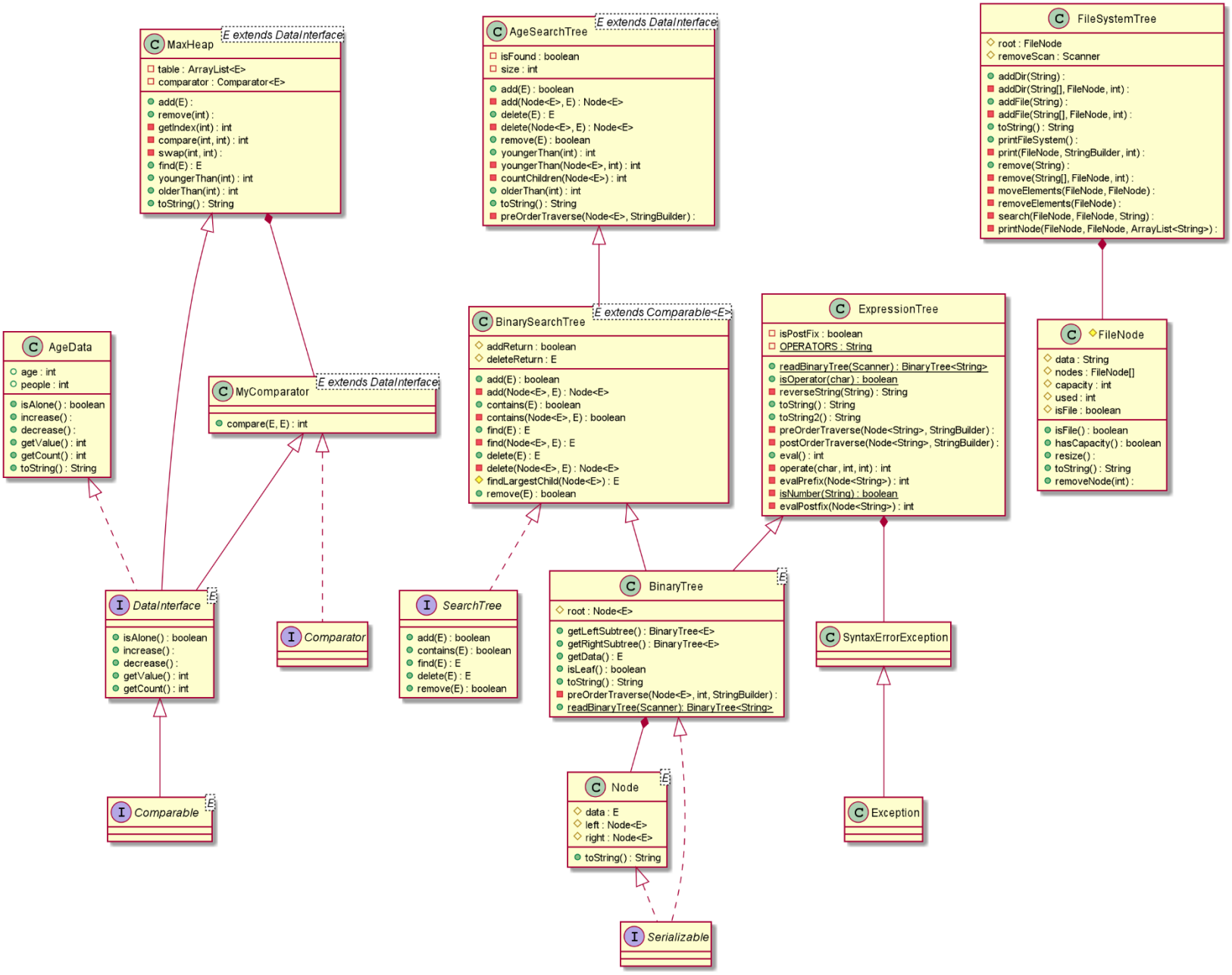
CSE 222/505 – Spring 2020

Homework #5 Report

Türker Tercan

171044032

Class Diagrams



Problem Solution Approach

Q1:

- To implement FileSystemTree class using general tree structure. I need to implement FileNode class to handle the nodes of the tree. A node can be created for a file or a directory.
- Inner static class FileNode, I kept a Boolean to store that whether the object is directory or file. If the node is file there can't be any child.
- In FileNode, children of a node can be any size. In order to that I implemented my own dynamic array. It stores children of that FileNode
- addDir method takes a path like "root/first_directory". First create a string array by dividing path to '/' characters. So we have directory names. Find the FindNode that named in string array if it exists go that node, if it is not creates that directory.
- addFile method works like addDir method. Just last node is always be a file.
- In these both methods, if a file occurs in traversing between nodes, it prints an error and returns.
- Remove method, it removes a directory or a file from the file system. A path is given like addFile and addDir method. The method will warn the user if the path cannot be found. When the node is found, asks the users whether to remove or not. Firstly, asks the user to remove all directory, if the user does not want to remove the whole directory, asks whether directory or files to remove and end of it, moves all directories and files in that directory to upper directory.
- Search method searches the entire system for a directory or a file including the given search characters in its name. The name of the file or a directory will be given to the method.
- printFileSystem will print the whole tree. It uses helper recursive method to traverse nodes and appends node's name to StringBuilder

Q2:

- ExpressionTree class of arithmetic operations which extends the BinaryTree class implementation given in my Data Structures book.
- A constructor to initialize the tree structure with the given expression string. The expression string will be given as a parameter to the constructor. This expression will include integer operands and arithmetic operators. Operands and operators will be separated by spaces. The constructor will use the overridden readBinaryTree method. It creates an expression tree by reading both prefix and postfix expressions.
- I implemented readBinaryTree to constructor tree from prefix expression. So when an object constructed, I checked it is prefix or postfix. If it is postfix expression, I reversed that given expression to use it like prefix. readBinaryTree method works recursively. Operands and operators divided by whitespaces. To ignore whitespaces, A scanner is

given to readBinaryTree. Scanner reads a string and checks it is operator or operand and takes action according to it.

- I implemented a postOrderTraverse method to raverse the tree post order and changed preOrderTraverse to proper append to StringBuilder
- toString2 method which will create a string of the tree structure un post order.
- Eval method which evaluates the expression and returns the result as an integer. It works recursively. If the object constructed by postfix expression, When eval method calculating transactions, operand left and operand right must be reversed to proper calculations.

Q3:

- We need to record the number of people in each age for a population. Extends the BinarySearchTree class of my book as AgeSearchTree class. I implemented AgeData class to handle both age and number of people at that age values. I should keep instances of AgeData in your tree. AgeData should be Comparable. CompareTo method of AgeData class will be used and the comparison will be done considering the age only.
- I implemented an interface to generic way to solve this problem. DataInterface extends Comparable interface and has methods to get age, people, increase and decrease methods
- AgeTree extends DataInterface<AgeData> so AgeSearchTree<AgeData> can construct.
- Add method firstly, check if a node with that age exists. If it exists, the number of people field of the AgeData object in that node will be increased 1.
- While removing a node, the remove function will first check if a node with that age exists. If it exists and the number of people field of this node's AgeData object is greater than 1, it will decrease the number of people field 1. If the number of people field is 1, it will remove the node.
- The find method will get an AgeData object of any age and find the AgeData object with the same age and return it.
- Add a youngerThan method which returns the number of people younger than an age.
- Add an olderThan method which returns the number of people older than an age. Be careful! If your youngerthan and olderThan methods always traverse all nodes that you cannot get whole credit. You should traverse only the nodes needs to be traversed.
- So youngerThan method traverses only necessary nodes. And older than method use youngerThan method to calculate.

Q4:

- Solve the problem in Q3 by using MaxHeap (where the maximum element is in the root node) this time. I implemented my heap by using ArrayList as described in my book. Implement MaxHeap class to handle the ArrayList heap operations. The key of heap will be "number of people" this time. So, the age which the highest number of people is at,

will be at the root. I wrote a Comparator to compara two objects of class AgeData. I already implement an interface to get, increase and decrease AgeData.

- add function to add a new record. It will first check if an AgeData object with that age exists in any index of the ArrayList. If it exists, the number of people field of the AgeData object in that node will be increased 1. (Check if any changes in heap is needed since the key is “number of people”.) Otherwise a new heap record with the AgeData object will be inserted.
- While removing a node, the remove function will first check if a node with that age exists. If it exists and the number of people field of this node’s AgeData object is greater than 1, it will decrease the number of people field 1. (Check if any changes needed since the key is “number of people”.) If the number of people field is 1, it will remove the node.
- The find method will get an AgeData object of any age and find the AgeData object with the same age and return it.
- Add a youngerThan method which returns the number of people younger than an age.
- Add an olderThan method which returns the number of people older than an age.

Test Cases:

Q1:

Test Scenario: addDir works

```
Test Data: myFileSystem.addDir("root/second_directory");
```

Expected:

root
    first_directory
    second_directory

Pass/Fail: Passed

Test Scenario: addDir(“”) prints an error

Test Data: myFileSystem.addFile("");

Expected: Empty path

Pass/Fail: Passed

Test Scenario: addFile() works properly

```
Test Data: myFileSystem.addFile("root/first_directory/dir1/file0");
```

Expected:

root
	first_directory
		dir1
			file0
	second_directory

Pass/Fail: Passed

Test Scenario: addFile() under a file

```
Test Data: myFileSystem.addFile("root/first_directory/dir1/file0/test.txt");
```

Expected:

You can not add a file under a file

File: file0

Pass/Fail: Passed

Test Scenario: remove() method

```
Test Data: myFileSystem.remove("root/first_directory");
```

Expected:

first_directory
	dir1
		file0

Do you want to remove directory with all elements in it(Y/N)
Y
root
	second_directory

Pass/Fail: Passed

Q2: Test Scenario: Constructs a tree from prefix and prints tree preOrderTraverse

```
Test Data:
ExpressionTree expTree = new ExpressionTree("+ + 10 * 5 15 20");
System.out.println(expTree.toString());
```

Excepted:
+ + 10 * 5 15 20
Pass/Fail: Passed

Test Scenario: Prints the constructed tree postOrderTraverse
Test Data:

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

Excepted:
+ 10 * 5 15 20 +
Pass/Fail: Passed

Test Scenario: Constructs a tree from postfix and prints tree preOrderTraverse and postOrderTraverse

```
Test Data:
ExpressionTree expTree2 = new ExpressionTree("10 5 15 * + 20 +");
System.out.println(expTree2.toString());
System.out.println(expTree2.toString2());
```

Excepted:
+ 20 + * 15 5 10
20 + * 15 5 10 +
Pass/Fail: Passed

Test Scenario: Evaluates the expressions both constructed trees
Test Data:

```
System.out.println(expTree.eval());
System.out.println(expTree2.eval());
```

Excepted:
105
105
Pass/Fail: Passed

Test Scenario: Empty expression
Test Data:

```
ExpressionTree test = new ExpressionTree("");
```

Excepted:
ExpressionTree$SyntaxErrorException: Empty expression
at ExpressionTree.<init>(ExpressionTree.java:37)
at testExpressionTree.main(testExpressionTree.java:15)
Pass/Fail: Passed

Test Scenario: Wrong Syntax
Test Data:

```
ExpressionTree test2 = new ExpressionTree("+ = 233 , .");
```

Excepted:
Invalid Syntax: =
Pass/Fail: Passed

Q3:

Test Scenario: Constructs objects and add elements and prints it

Test Data:

```
AgeSearchTree<AgeData> ageTree = new AgeSearchTree<>();
ageTree.add(new AgeData(10));
ageTree.add(new AgeData(20));
ageTree.add(new AgeData(5));
ageTree.add(new AgeData(15));
ageTree.add(new AgeData(10));
ageTree.add(new AgeData(8));
String treeStr = ageTree.toString();
System.out.print(treeStr);
```

Excepted:

10 - 2
5 - 1
null
8 - 1
null
null
20 - 1
15 - 1
null
null
null

Pass/Fail: Passed

Test Scenario: youngerThan() method works

Test Data:

```
System.out.println(ageTree.youngerThan(15));
```

Excepted:

4

Pass/Fail: 4

Test Scenario: olderThan() method works

Test Data:

```
System.out.println(ageTree.olderThan(15));
```

Excepted:

1

Pass/Fail: Passed

Test Scenario: find() method works

Test Data:

```
System.out.println(ageTree.find(new AgeData(10)).toString());
```

Excepted:

10 – 2

Pass/Fail: Passed

Test Scenario: remove method works
Test Data:

```
System.out.println(ageTree.remove(new AgeData(5)));
System.out.println(ageTree.toString());
```

Excepted:
true
10 - 2
8 - 1
null
null
20 - 1
15 - 1
null
null
null
Pass/Fail: Passed

Test Scenario: remove but object is not in the tree
Test Data:

```
System.out.println(ageTree.remove(new AgeData(3)));
```

Excepted:
false
Pass/Fail: Passed

Q4:
Test Scenario: Create a heap and add AgeData
Test Data:

```
//Create an empty heap
MaxHeap<AgeData> heap = new MaxHeap<>();
//Add nodes
heap.add(new AgeData(10));
heap.add(new AgeData(5));
heap.add(new AgeData(70));
heap.add(new AgeData(10));
heap.add(new AgeData(50));
heap.add(new AgeData(5));
heap.add(new AgeData(15));
String heapStr= heap.toString();
System.out.println(heapStr);
```

Excepted:
10 - 2
5 - 2
70 - 1
50 - 1
15 – 1
Pass/Fail: Passed

Test Scenario: youngerThan method works
Test Data:

```
System.out.println(heap.youngerThan(10));
```

Excepted:
2
Pass/Fail: Passed

Test Scenario: find method works
Test Data:

```
System.out.println(heap.find(new AgeData(10)).toString());
```

Excepted:
10 – 2
Pass/Fail: Passed

Test Scenario: remove method works and after that reorder preorder
Test Data:

```
heap.remove(10);
System.out.println(heap.toString());
```

Excepted:
5 - 2
10 - 1
70 - 1
50 - 1
15 – 1
Pass/Fail: Passed

Test Scenario: remove method not but element is not in the heap
Test Data:

```
heap.remove(2);
```

Excepted:
Exception in thread "main" java.util.NoSuchElementException
Pass/Fail: Passed

Running command and result

All methods and classes works properly.
All testes were successful.