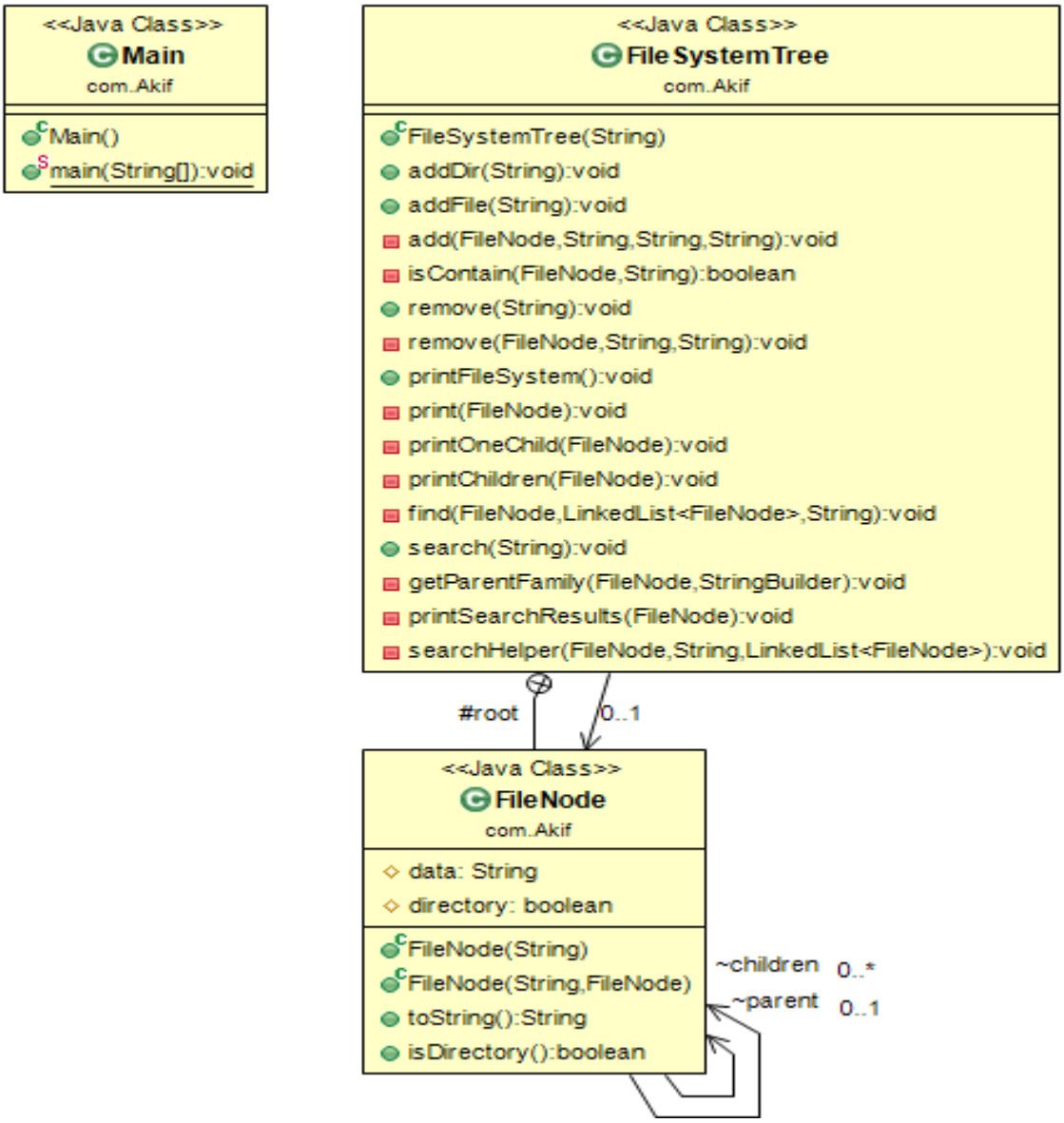
# GIT Department of Computer Engineering CSE 222/505 - Spring 2020 Homework 5 Report

**Akif KARTAL**
**171044098**

# Q1 REPORT

## 1. CLASS DIAGRAMS




## 2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.
2) **Analyze the problem :** I identify;
   – Input data
   – Output data
   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

   **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

   Firstly, I wrote the **FileNode** inner class. In this class I kept data of node, list of children and parent of node, a Boolean isDirectory field and 2 constructor and some methods. After that I wrote **FileSystemTree** class to handle a file system hierarchy in a general tree structure. In this class I kept root of tree and by using this root I did operation of this class.

   **To add a new directory or file** first I find parent of new node then by using this parent node if all directory path is true I add a new directory or file to the tree. While doing this to find parent node I used a **recursive traverse method** such that it traverses each node by traversing each child of that node.

   In other word, it starts from leftmost child node it traverses all children of this node and children of the that child after there is no children in left side such as if it hits the null on left side, it continues with other children. After all elements is done it stops automatically without a base case since there is no child anymore.

   **To delete a directory** I did same process in adding part. First I find node which will be deleted then by using its parent node if all directory path is true I delete that directory or file from the tree. While doing this to find this node I used a recursive traverse method that is mentioned above.

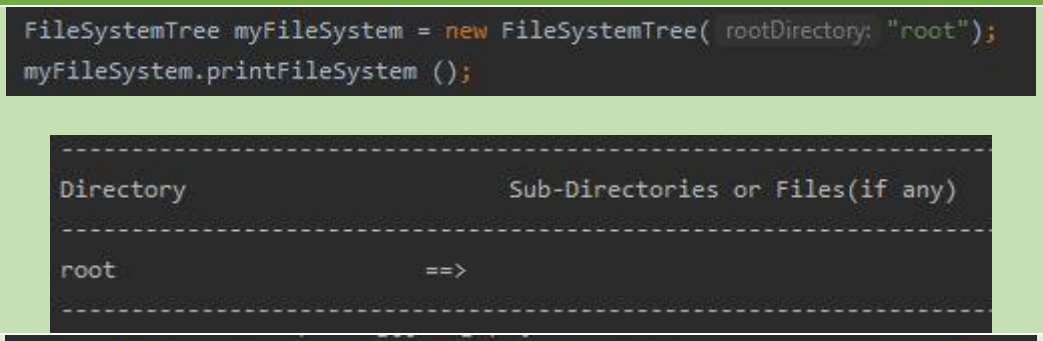
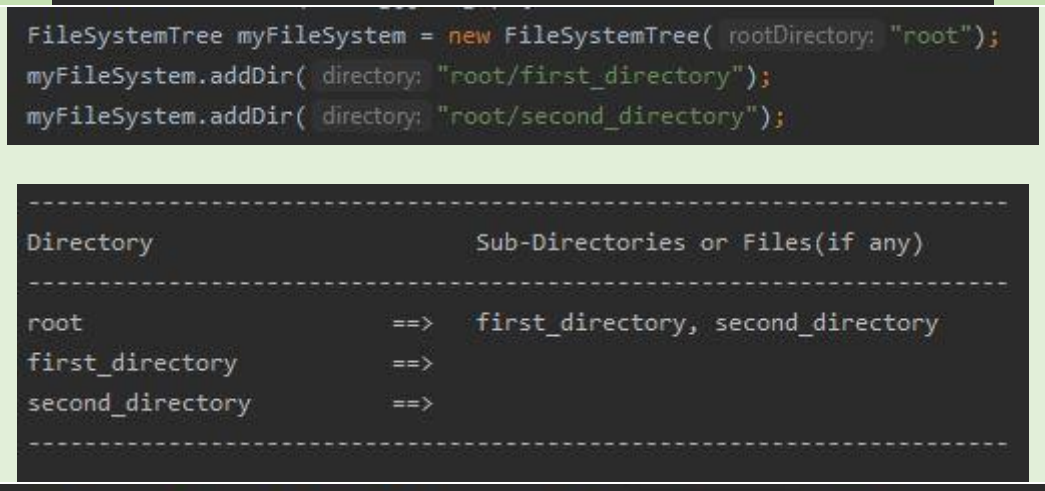
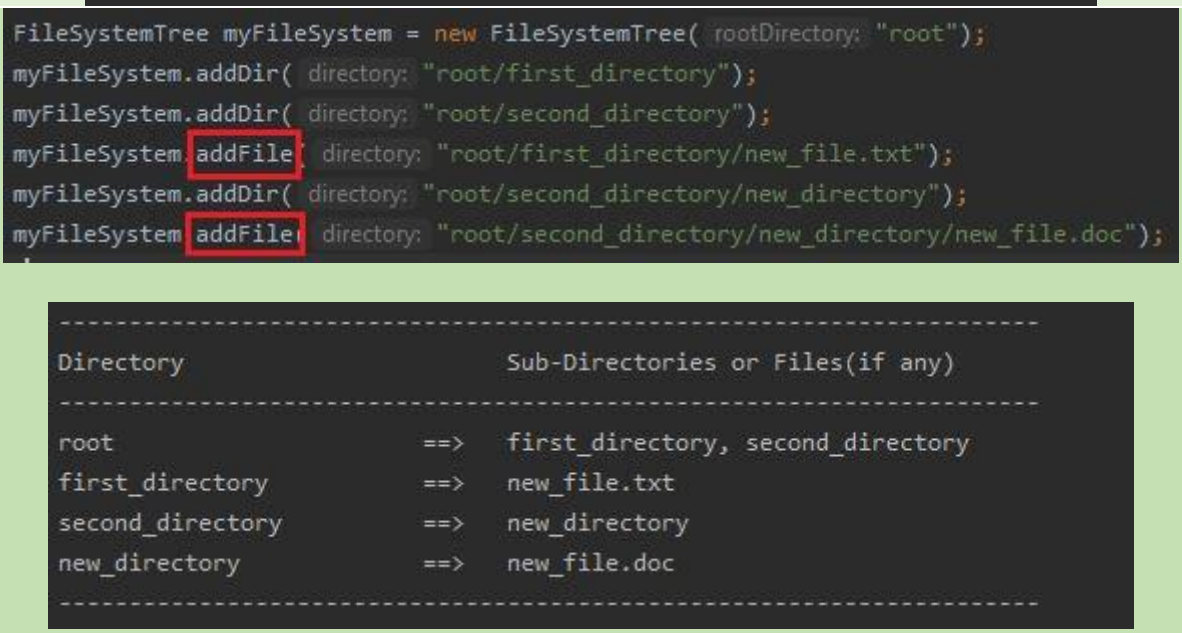
   **To search a directory** I traverse all tree by using by using my traverse process and if any directory or file contains given characters I add them a node list. After traversing is done I checked the list and I printed each element of this list by checking whether it is a directory or file.

   **To print all tree** again I traverse all tree and I printed each element of tree with its children in a good shape.
4) **Test and verify the program:** To test program I wrote the Main class in this class in main method I test each method of **FileSystemTree** class by calling the methods and I printed the test results.
5) **Maintain and update the program :** I keep the program up-to-date

### 3. TEST CASES

| Test ID | Test Case | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|---|---|
| **T1** | Construct the tree with root directory | Create object of class with proper root directory | "root" | The tree has been constructed correct way | As Expected | **Pass** |
| **T2** | Test “**addDir**” method | Call the method with proper parameter | "root/first_directory" | The directory has been added correct way | As Expected | **Pass** |
| **T3** | Test “**addFile**” method | Call the method with proper parameter | "root/first_directory/new_file.txt" | The file has been added correct way | As Expected | **Pass** |
| **T4** | Test “**remove**” method | Call the method with proper parameter | "root/second_directory/new_directory" | The directory has been removed correct way | As Expected | **Pass** |
| **T5** | Test “**search**” method | Call the method with proper parameter | "directory" | The results have been printed correct way | As Expected | **Pass** |
| **T6** | Test “**printFileSystem**” method | Call the method | A proper file sytem tree | The file system have been printed correct way | As Expected | **Pass** |

## 4. RUNNING AND RESULTS

| Test ID | Test Result |
|---|---|
| T1 |  |
| T2 |  |
| T3 |  |

```
------------------------------------------------------------------------
Directory                       Sub-Directories or Files(if any)
------------------------------------------------------------------------
root                      ==>   first_directory, second_directory
first_directory           ==>   new_file.txt
second_directory          ==>   new_directory
new_directory             ==>   new_file.doc
------------------------------------------------------------------------
```

**T4**

```
myFileSystem.remove( directory: "root/first_directory/new_file.txt");
myFileSystem.remove( directory: "root/second_directory/new_directory");
```

```
Your directory was removed!
Your Directory contains following Directories or files
new_file.doc
Do you want to remove(yes/no): yes
Your directory was removed!
------------------------------------------------------------------------
Directory                       Sub-Directories or Files(if any)
------------------------------------------------------------------------
root                      ==>   first_directory, second_directory
first_directory           ==>
second_directory          ==>
------------------------------------------------------------------------
```

**T5**

```
------------------------------------------------------------------------
Directory                       Sub-Directories or Files(if any)
------------------------------------------------------------------------
root                      ==>   first_directory, second_directory
first_directory           ==>   new_file.txt
second_directory          ==>   new_directory
new_directory             ==>   new_file.doc
------------------------------------------------------------------------
```

```
myFileSystem.search( word: "directory");
```

```
Your Search Results:
dir - root/first_directory
dir - root/second_directory
dir - root/second_directory/new_directory
```

```
myFileSystem.search( word: "new");
```

```
Your Search Results:
file - root/first_directory/new_file.txt
dir - root/second_directory/new_directory
file - root/second_directory/new_directory/new_file.doc
```

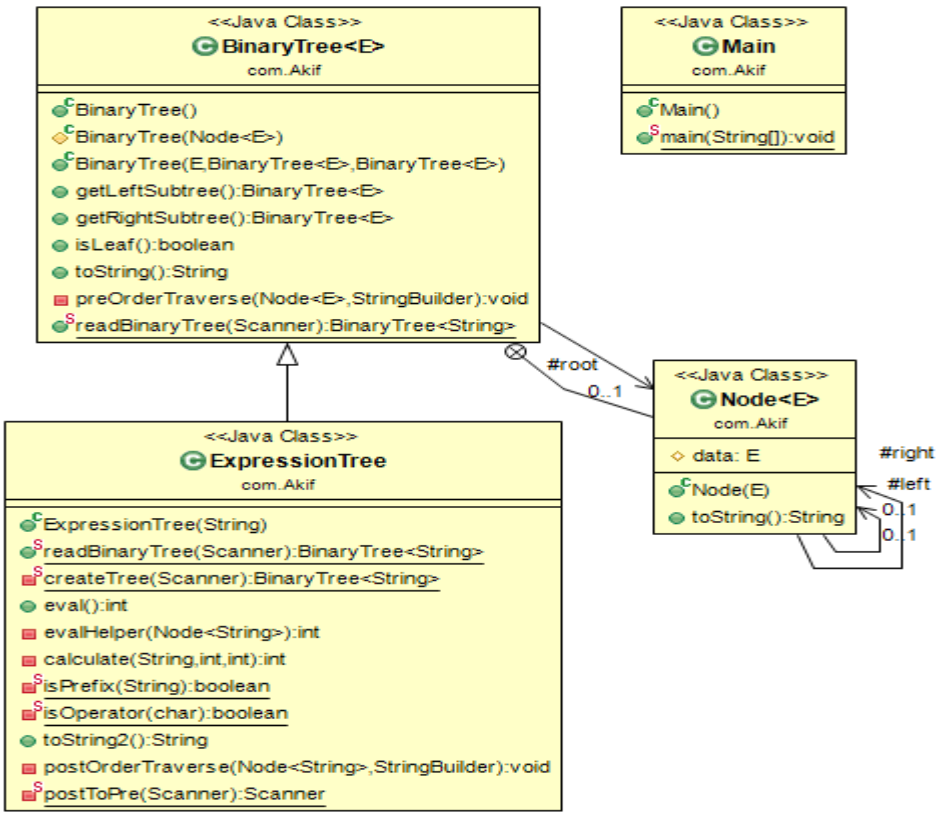
**T6**

```
myFileSystem.printFileSystem ();
```

```
-----------------------------------------------------------------------
Directory                       Sub-Directories or Files(if any)
-----------------------------------------------------------------------
root                      ==>   first_directory, second_directory
first_directory           ==>   new_file.txt
second_directory          ==>   new_directory
new_directory             ==>   new_file.doc
-----------------------------------------------------------------------
```

# Q2 REPORT

## 1. CLASS DIAGRAMS



## 2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.
2) **Analyze the problem :** I identify;
   – Input data
   – Output data
   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

   **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

   Firstly, I wrote the **BinaryTree** class from the book. In this class I only changed preOrderTraverse method to print expression as in example.

   Then, I wrote the **ExpressionTree** class which extends from BinaryTree class. In this class I override **readBinary** method such that it can create an expression tree by reading both prefix and postfix expressions. Still readBinary class works for prefix expressions but in this method if the expression is postfix I converted it prefix and I use the createTree which is nearly same as readBinary method and by doing this I created the tree.

   But also I changed this method since our expression does not contain “null” statements.

   In this class I also write **postOrderTraverse** and **toString2** method to create a string of the tree structure in post order.

   Lastly, I wrote **recursive eval** method which evaluates the expression without using a stack.

   **Note that** while creating tree or evaluating expression I didn’t use stack or list.
4) **Test and verify the program:** To test program I wrote the **Main** class in this class in main method I test each method of **ExpressionTree** class by calling the methods and I printed the test results.
5) **Maintain and update the program :** I keep the program up-to-date.

### 3. TEST CASES

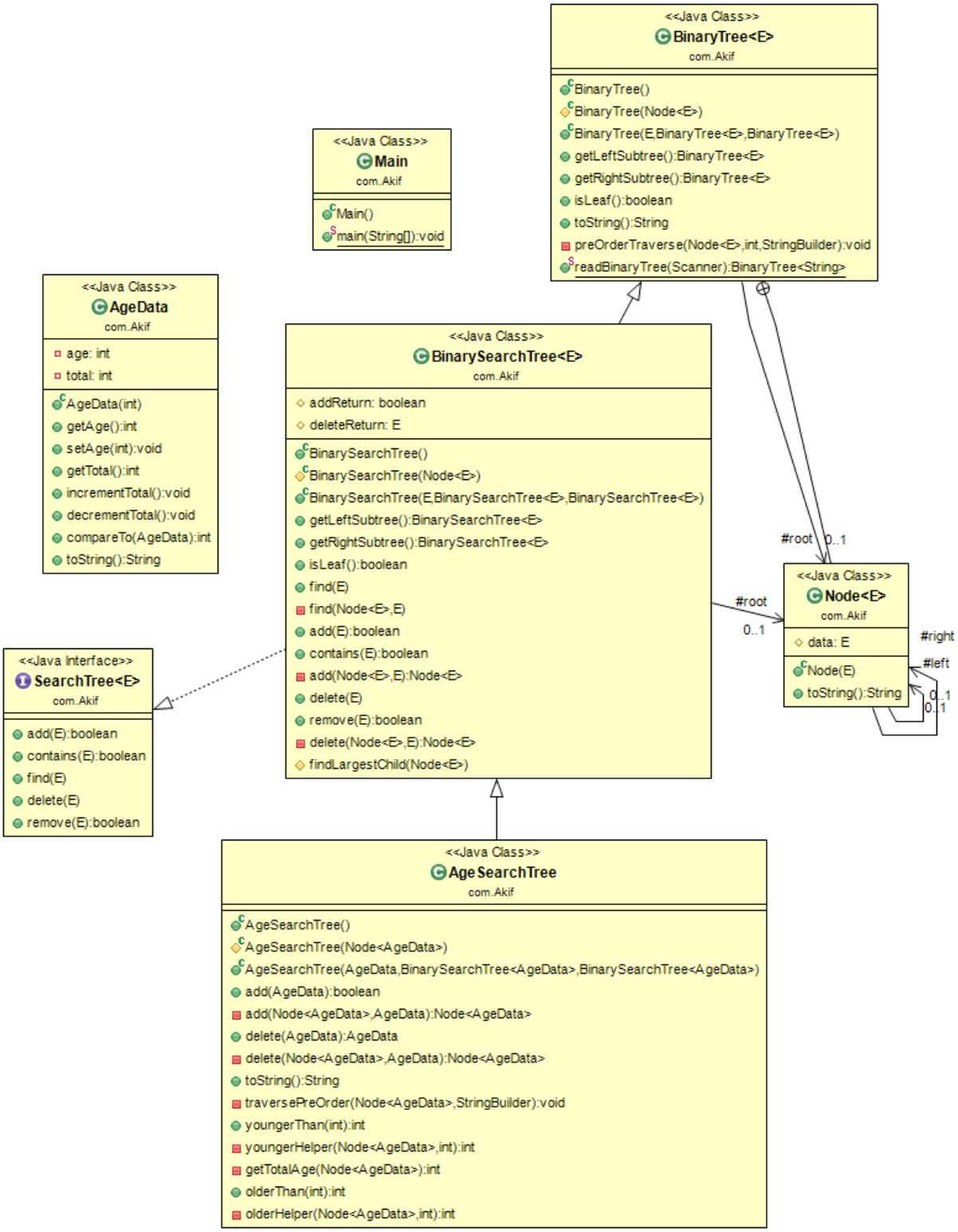
| Test ID | Test Case | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|---|---|
| T1 | Construct the tree with prefix expression | Create object of class with proper expression | "+ 7 + / - 10 * 5 3 10 - 2 / 4 8" | The tree has been constructed correct way | As Expected | Pass |
| T2 | Construct the tree with postfix expression | Create object of class with proper expression | "10 5 15 * + 20 +" | The tree has been constructed correct way | As Expected | Pass |
| T3 | Test the read binary tree in constructor | Create object of class with proper expression | "+ 7 + / - 10 * 5 3 10 - 2 / 4 8" | The tree has been constructed correct way | As Expected | Pass |
| T4 | Test toString2 method | Call method | A proper expression tree | The tree has been printed in postOrder | As Expected | Pass |
| T5 | Test toString method | Call method | A proper expression tree | The tree has been printed in preOrder | As Expected | Pass |
| T6 | Test eval method | Call method | A proper expression tree | Correct Result of expression | As Expected | Pass |

## 4. RUNNING AND RESULTS

| Test ID | Test Result |
|---|---|
| T1 | `ExpressionTree expTree = new ExpressionTree("+ 7 + / - 10 * 5 3 10 - 2 / 4 8");`<br>`System.out.println ("preOrder Traverse: "+expTree.toString ());`<br>`System.out.println ("postOrder Traverse: "+expTree.toString2 ());`<br><br>`preOrder Traverse: + 7 + / - 10 * 5 3 10 - 2 / 4 8`<br>`postOrder Traverse: 7 10 5 3 * - 10 / 2 4 8 / - + +` |
| T2 | `ExpressionTree expTree2 = new ExpressionTree("10 5 15 * + 20 +");`<br>`System.out.println ("preOrder Traverse: "+expTree2.toString ());`<br>`System.out.println ("postOrder Traverse: "+expTree2.toString2 ());`<br><br>`preOrder Traverse: + + 10 * 5 15 20`<br>`postOrder Traverse: 10 5 15 * + 20 +` |
| T3 | `ExpressionTree expTree = new ExpressionTree("+ 7 + / - 10 * 5 3 10 - 2 / 4 8");`<br>`System.out.println ("preOrder Traverse: "+expTree.toString ());`<br>`System.out.println ("postOrder Traverse: "+expTree.toString2 ());`<br><br>`preOrder Traverse: + 7 + / - 10 * 5 3 10 - 2 / 4 8`<br>`postOrder Traverse: 7 10 5 3 * - 10 / 2 4 8 / - + +` |
| T4 | `ExpressionTree expTree = new ExpressionTree("+ 7 + / - 10 * 5 3 10 - 2 / 4 8");`<br>`System.out.println ("preOrder Traverse: "+expTree.toString ());`<br>`System.out.println ("postOrder Traverse: "+expTree.toString2 ());`<br><br>`preOrder Traverse: + 7 + / - 10 * 5 3 10 - 2 / 4 8`<br>`postOrder Traverse: 7 10 5 3 * - 10 / 2 4 8 / - + +` |
| T5 | `ExpressionTree expTree = new ExpressionTree("+ 7 + / - 10 * 5 3 10 - 2 / 4 8");`<br>`System.out.println ("preOrder Traverse: "+expTree.toString ());`<br>`System.out.println ("postOrder Traverse: "+expTree.toString2 ());`<br><br>`preOrder Traverse: + 7 + / - 10 * 5 3 10 - 2 / 4 8`<br>`postOrder Traverse: 7 10 5 3 * - 10 / 2 4 8 / - + +` |
| T6 | `ExpressionTree expTree = new ExpressionTree("+ 7 + / - 10 * 5 3 10 - 2 / 4 8");`<br>`System.out.println ("Evaluation result 1: "+expTree.eval ());`<br>`ExpressionTree expTree2 = new ExpressionTree("10 5 15 * + 20 +");`<br>`System.out.println ("Evaluation result 2: "+expTree2.eval ());`<br><br>`Evaluation result 1: 9`<br>`Evaluation result 2: 105` |

# Q3 REPORT

## 1. CLASS DIAGRAMS



## 2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.
2) **Analyze the problem :** I identify;
   – Input data
   – Output data
   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

   **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

   Firstly, I wrote the **BinarySearchTree** class from the book which extends from BinaryTree class and implements SearchTree class. After that I wrote **AgeData** class to handle both age and number of people at that age values.

   This class implements Comparable interface.

   Lastly I wrote **AgeSearchTree** class which extends from BinarySearchTree class.

   Then I override only add and remove method from binary search tree class and I wrote other methods. Note that AgeSearchTree is not a generic class since it works only with AgeData objects.

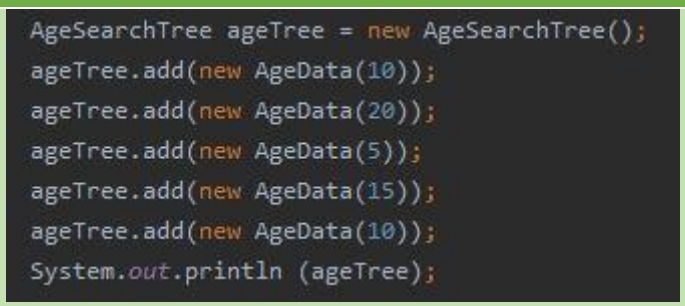
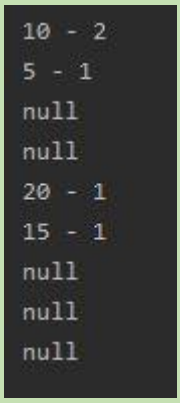
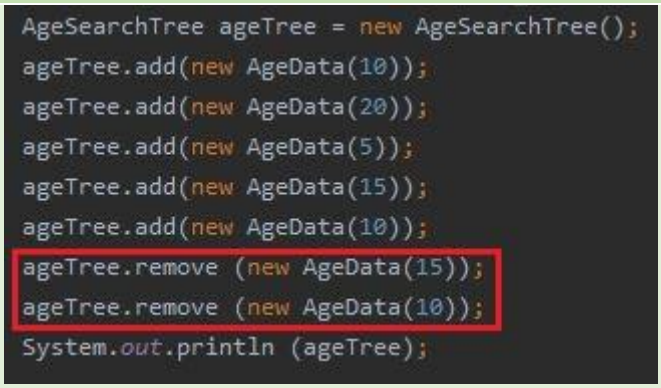
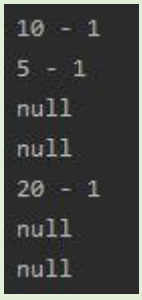
   While **printing tree**, I used preOrder traverse algorithm.

   While finding **younger and older ages** I only traverse the nodes needs to be traversed recursively.
4) **Test and verify the program:** To test program I wrote the Main class in this class in main method I test each method of **AgeSearchTree** class by calling the methods and I printed the test results.
5) **Maintain and update the program :** I keep the program up-to-date.

### 3. TEST CASES

| Test ID | Test Case | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|---|---|
| **T1** | Test "**add**" method | Call the method with proper parameter | new AgeData(10) | The AgeData object has been added to tree correct way | As Expected | **Pass** |
| **T2** | Test "**remove**" method | Call the method with proper parameter | new AgeData(15) | The AgeData object has been removed from tree correct way | As Expected | **Pass** |
| **T3** | Test "**find**" method | Call the method with proper parameter | new AgeData(10) | The correct result printed to the screen | As Expected | **Pass** |
| **T4** | Test "**youngerThan**" method | Call the method with proper parameter | 15 | The result has been calculated correctly | As Expected | **Pass** |
| **T5** | Test "**olderThan**" method | Call the method with proper parameter | 15 | The result has been calculated correctly | As Expected | **Pass** |
| **T6** | Test "**toString**" method | Call the method with proper parameter | A proper AgeSearchTree tree | Correct representation of tree | As Expected | **Pass** |

## 4. RUNNING AND RESULTS

| Test ID | Test Result |
|---|---|
| T1 |  <br>  |
| T2 |  <br>  |

**T3**

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

```
10 - 2
```

**T4**

```
AgeSearchTree 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));
System.out.print ("Younger than age 15: ");
System.out.println(ageTree.youngerThan( age: 15));
```

```
Younger than age 15: 3
```

**T5**

```
AgeSearchTree 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));
System.out.print ("Older than age 15: ");
System.out.println(ageTree.olderThan( age: 15));
```

```
Older than age 15: 1
```

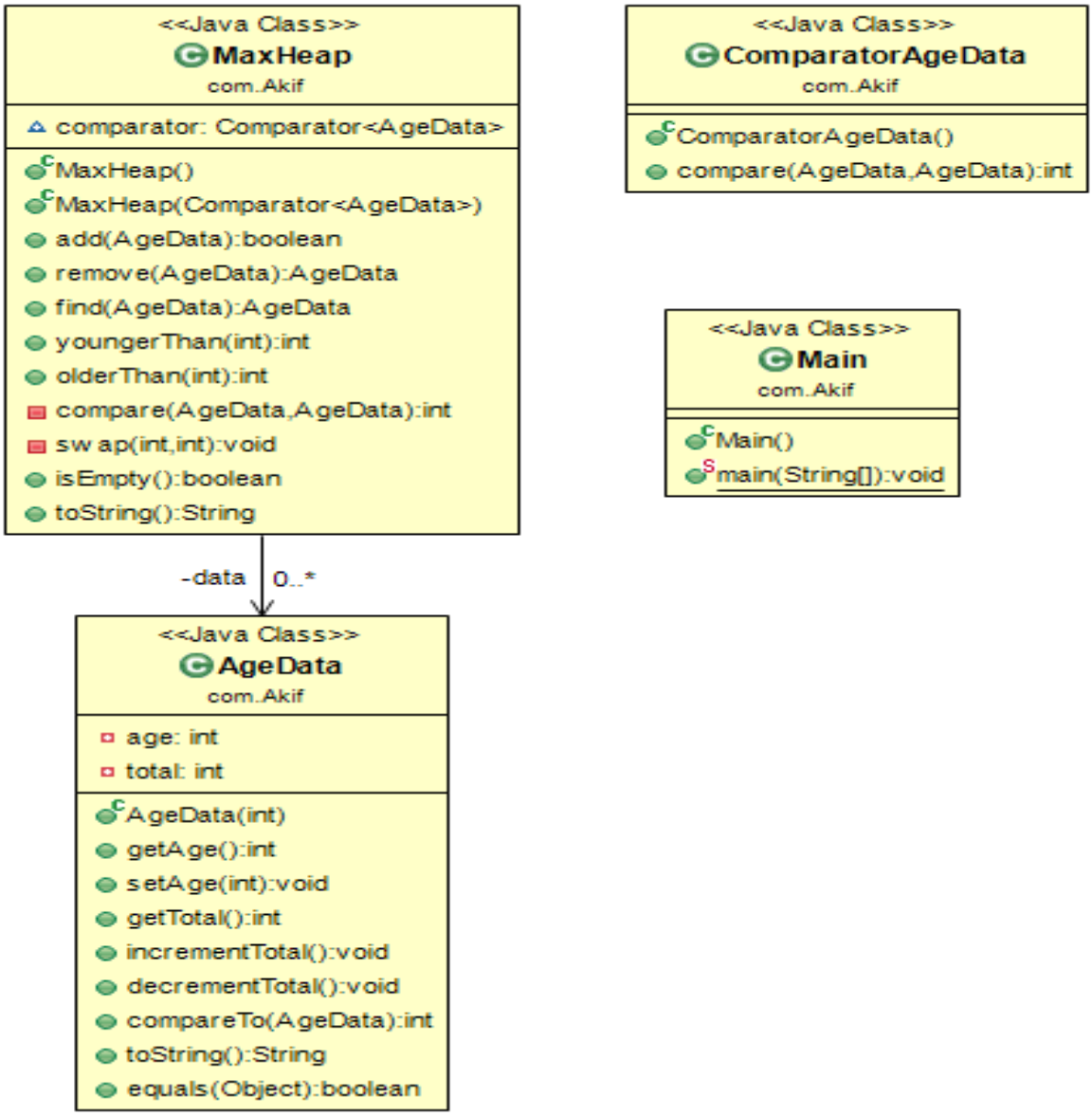
**T6**

```
AgeSearchTree 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));
System.out.println (ageTree);
```

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

# Q4 REPORT

## 1. CLASS DIAGRAMS




## 2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.
2) **Analyze the problem :** I identify;
   – Input data
   – Output data
   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

   **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

   Firstly, I wrote the **AgeData** class to handle both age and number of people at that age values. This class implements Comparable interface. This class compares two AgeData objects according to the number of people at that age.

   After That I wrote **ComparatorAgeData** class which implements Comparator<AgeData> interface. This class compares two AgeData objects according to the number of people at that age.

   Lastly, I wrote **MaxHeap** class to handle heap operations by using AgeData ArrayList.

   In this class I wrote only needed methods **add, remove, find** etc.

   While **writing this methods** to keep the ArrayList is in heap order I took some code pieces from book and I changed some part of them.

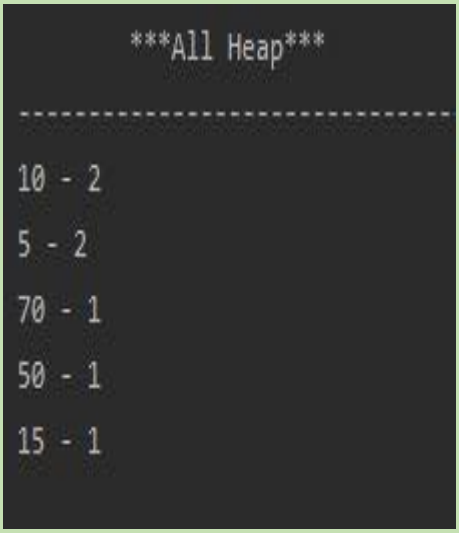
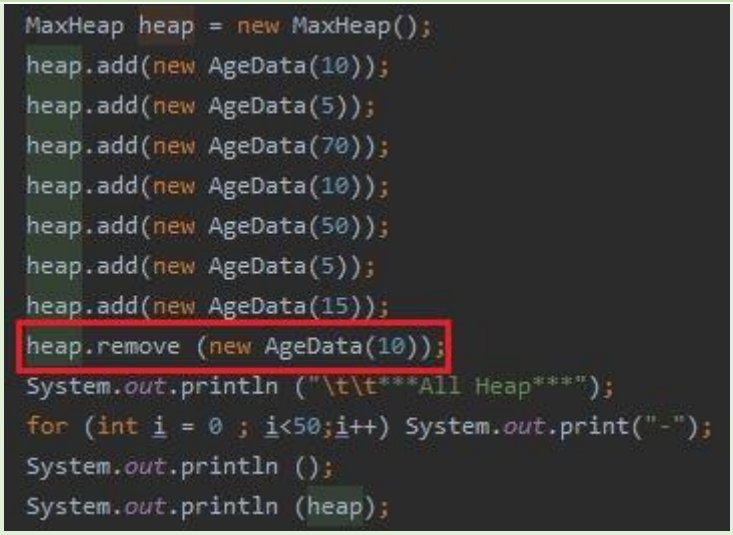
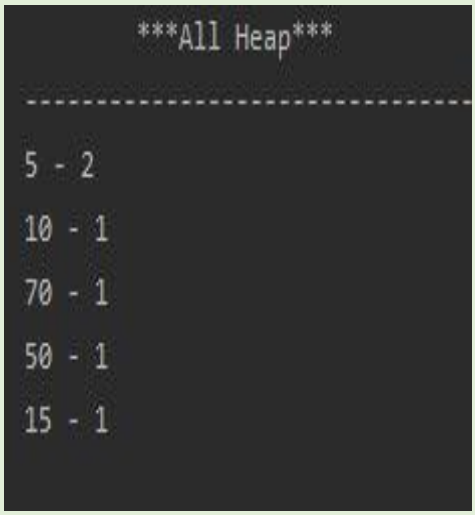
   While finding **younger and older ages** I traverse all arraylist and I calculate the result.

   While **printing tree**, I traverse all arraylist from the first element and I printed the each element. Note that the first element is key of heap according to “number of people” at that age and so on.
4) **Test and verify the program:** To test program I wrote the Main class in this class in main method I test each method of **MaxHeap** class by calling the methods and I printed the test results.
5) **Maintain and update the program :** I keep the program up-to-date.

### 3. TEST CASES

| Test ID | Test Case | Test Steps | Test Data | Expected Results | Actual Results | Pass/Fail |
|---|---|---|---|---|---|---|
| **T1** | Test "**add**" method | Call the method with proper parameter | new AgeData(10) | The AgeData object has been added to heap correct way | As Expected | **Pass** |
| **T2** | Test "**remove**" method | Call the method with proper parameter | new AgeData(10) | The AgeData object has been removed from heap correct way | As Expected | **Pass** |
| **T3** | Test "**find**" method | Call the method with proper parameter | new AgeData(10) | The correct result printed to the screen | As Expected | **Pass** |
| **T4** | Test "**youngerThan**" method | Call the method with proper parameter | 10 | The result has been calculated correctly | As Expected | **Pass** |
| **T5** | Test "**olderThan**" method | Call the method with proper parameter | 5 | The result has been calculated correctly | As Expected | **Pass** |
| **T6** | Test "**toString**" method | Call the method with proper parameter | A proper heap | Correct representation of heap | As Expected | **Pass** |

## 4. RUNNING AND RESULTS

| Test ID | Test Result |
|---|---|
| T1 |   |
| T2 |   |

```
//Create an empty heap
MaxHeap heap = new MaxHeap();
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));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
```

```
***All Heap***
--------------------------------------------------
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```

```
MaxHeap heap = new MaxHeap();
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));
heap.remove (new AgeData(10));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
```

```
***All Heap***
--------------------------------------------------
5 - 2
10 - 1
70 - 1
50 - 1
15 - 1
```

**T3**

```
//Create an empty heap
MaxHeap heap = new MaxHeap();
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));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println ("Find age 10 : "+ heap.find (new AgeData (10)));
System.out.print ("Younger than age 10: ");
System.out.println(heap.youngerThan( age: 10));
System.out.print ("Older than age 5: ");
System.out.println(heap.olderThan( age: 5));
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
```

```
        ***All Heap***
--------------------------------

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


--------------------------------
Find age 10 : 10 - 2
Younger than age 10: 2
Older than age 5: 5
--------------------------------
```

**T4**

```
//Create an empty heap
MaxHeap heap = new MaxHeap();
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));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println ("Find age 10 : "+ heap.find (new AgeData (10)));
System.out.print ("Younger than age 10: ");
System.out.println(heap.youngerThan( age: 10));
System.out.print ("Older than age 5: ");
System.out.println(heap.olderThan( age: 5));
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
```

```
        ***All Heap***
--------------------------------

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


--------------------------------
Find age 10 : 10 - 2
Younger than age 10: 2
Older than age 5: 5
--------------------------------
```

**T5**

```
//Create an empty heap
MaxHeap heap = new MaxHeap();
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));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println ("Find age 10 : "+ heap.find (new AgeData (10)));
System.out.print ("Younger than age 10: ");
System.out.println(heap.youngerThan( age: 10));
System.out.print ("Older than age 5: ");
System.out.println(heap.olderThan( age: 5));
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
```

```
***All Heap***
--------------------------------------------------
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1

--------------------------------------------------
Find age 10 : 10 - 2
Younger than age 10: 2
Older than age 5: 5
--------------------------------------------------
```

**T6**

```
//Create an empty heap
MaxHeap heap = new MaxHeap();
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));
System.out.println ("\t\t***All Heap***");
for (int i = 0 ; i<50;i++) System.out.print("-");
System.out.println ();
System.out.println (heap);
```

```
***All Heap***
--------------------------------------------------
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```