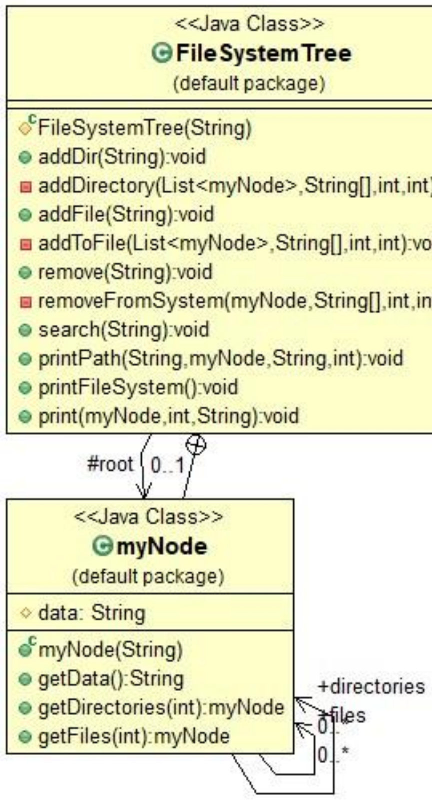
GIT Department of Computer Engineering
CSE 222/505 - Spring 2020
Homework #5 Report

EMİRHAN UZUN

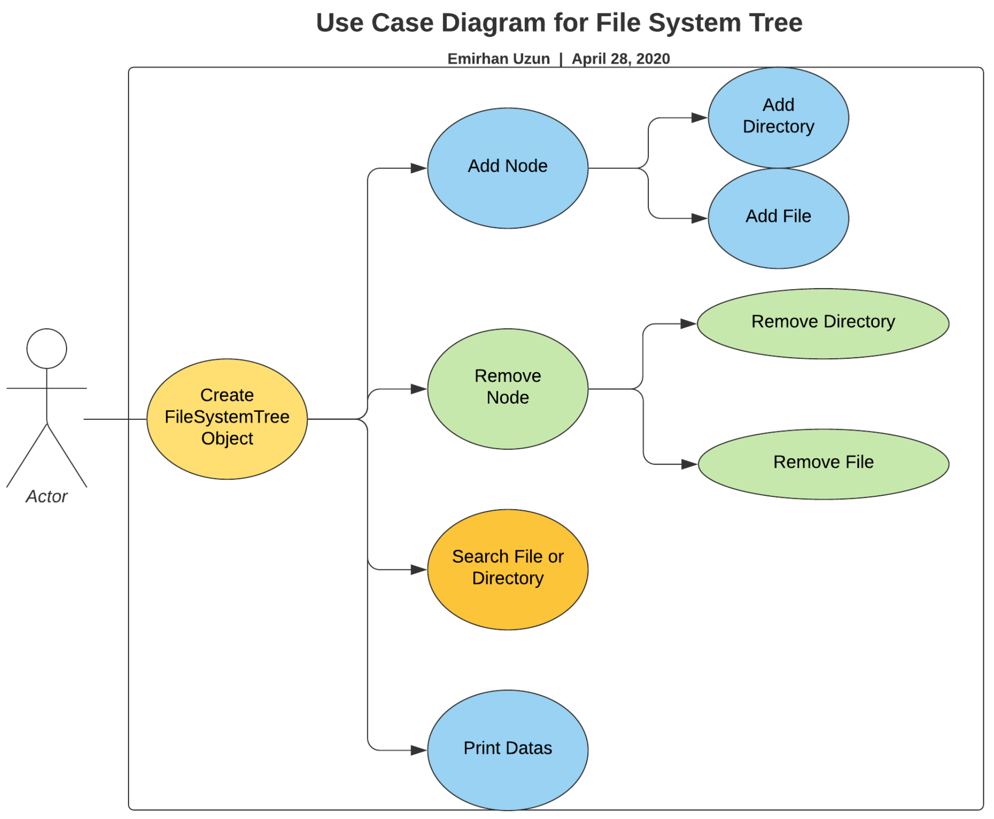
171044019

QUESTION 1

1) Class Diagram



2) Use Case Diagram



3) Problem Solution Approach

Firstly, I created my inner node class. In that class, I created a string field to the name of object and 2 list for directories and files. Then I have root (node type) data field.

In my add methods, I split the given path from “/” characters. And then I checked if that paths are correct or not. If they are true and not exist, then I added to list which is belong to (directory or file list).

In my remove methods, again I split the given path from “/” characters. I checked the directories and files in order. If the path is correct than I remove the path’s last object. Otherwise I throw an exception.

In search method, I recursively check directories and files in order. If these list nodes contains the given string, then I print the path of that node.

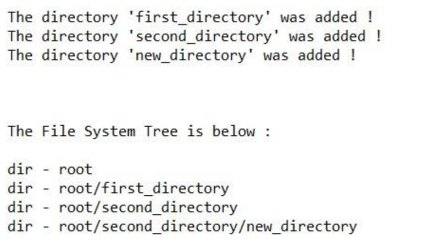
In print method, I firstly print the directories of current node roots, and then print the files of current node root. It seems like a pre order traversal method. If the directory list is my left child and file list is my right child, then we say the print methods works like pre order traversal method.

In summary, I check the lists before doing an operation, and then according to situation, I perform the operation.

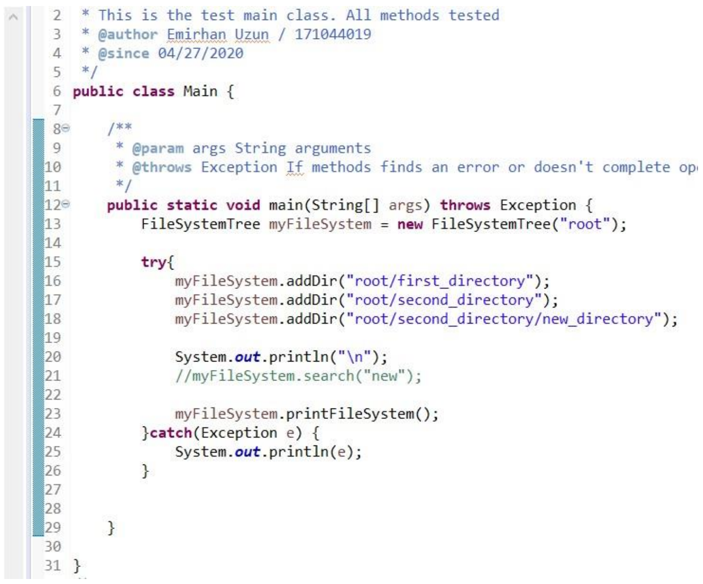
4) Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Check the constructor	1.Write the root name 2.Create the object	Root	Object should be created and root name must be initialized	As Expected	Pass
T02	Check addDir method	1.Write the paths truely	-Root -Root Path	The node should be created and added to root	As Expected	Pass
T03	Check addDir method with wrong path	1.Write the path wrongly	-Root -Root Path	The directory node shouldn’t be created and the program must give an error or exception	As Expected	Pass
T04	Check addFile method	1.Write the paths truely	-Root -Root Path	The node should be created and added to root	As Expected	Pass
T05	Check addFile method with wrong path	1.Write the path wrongly	-Root -Root Path	The file node shouldn’t be created and the program must give an error or exception	As Expected	Pass
T06	Check remove method	1.Write to path	-Root -Removing file or directory path string	If the files or directories contain the string, it removes the path of that node	As Expected	Pass
T07	Check search method	1.Write to path	-Root -Searching string	If the files or directories contain the string, it prints the path of that node	As Expected	Pass
T08	Check Print method		-Root	Prints the directories and files paths in order	As Expected	Pass

5) Running Command and Results

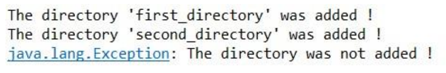


```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
The directory 'new_directory' was added !



The File System Tree is below :

dir - root
dir - root/first_directory
dir - root/second_directory
dir - root/second_directory/new_directory
```

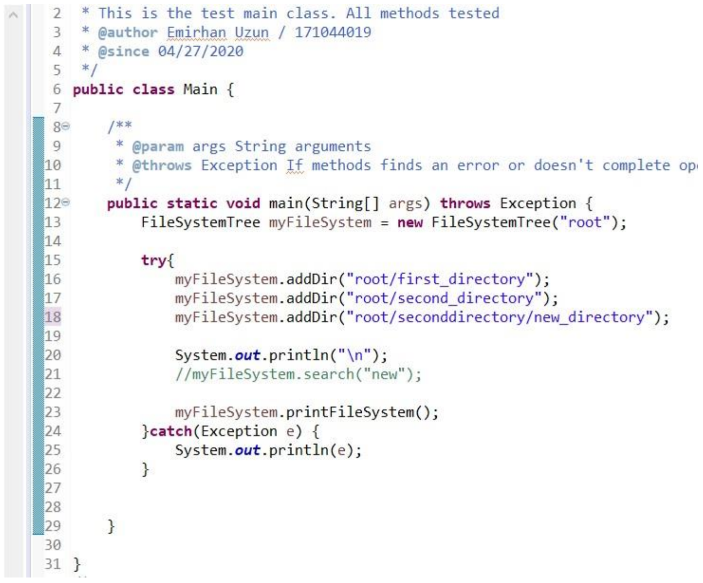


```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete op
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addDir("root/second_directory/new_directory");

            System.out.println("\n");
            //myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }

}
```

This is the addDir method with true paths

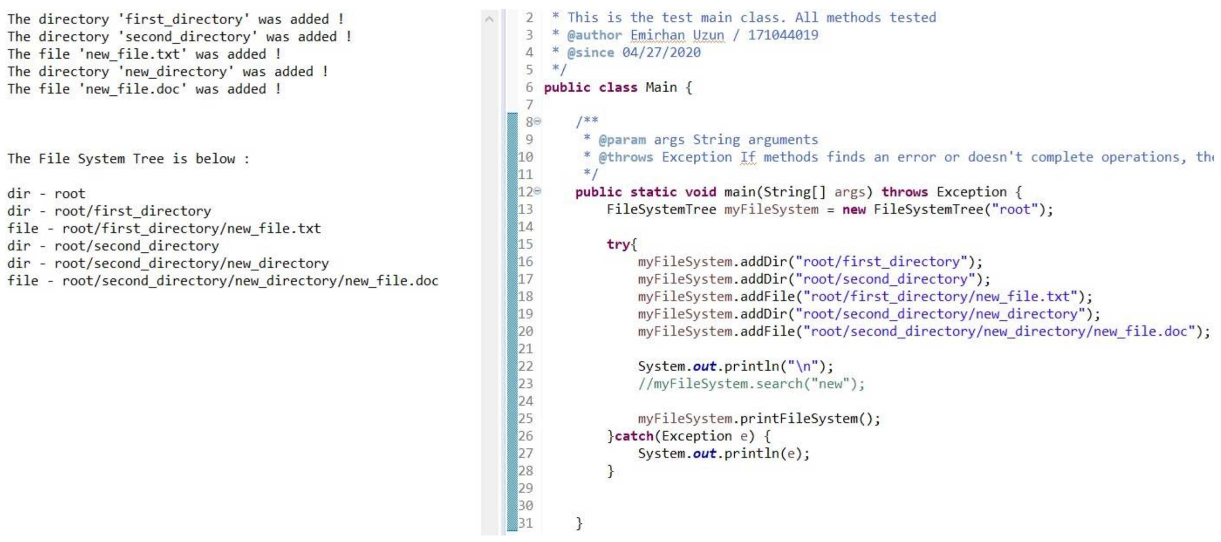


```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
java.lang.Exception: The directory was not added !
```



```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete op
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addDir("root/seconddirectory/new_directory");

            System.out.println("\n");
            //myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }

}
```

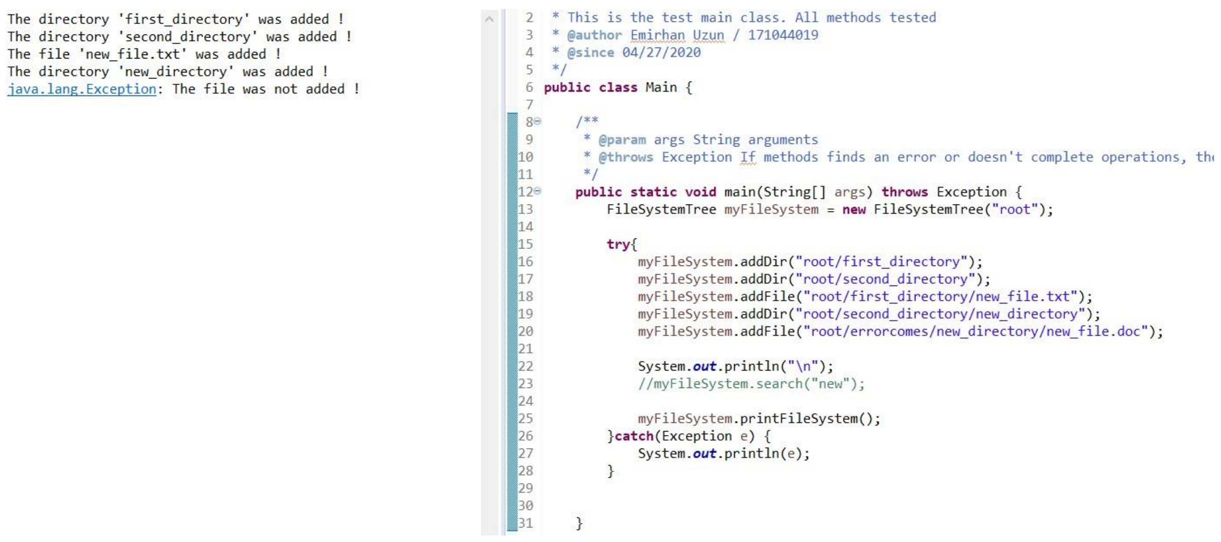
This is the addDir method with wrong path and it throws an exception



```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
The file 'new_file.txt' was added !
The directory 'new_directory' was added !
The file 'new_file.doc' was added !



The File System Tree is below :

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

```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete operations, th
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addFile("root/first_directory/new_file.txt");
            myFileSystem.addDir("root/second_directory/new_directory");
            myFileSystem.addFile("root/second_directory/new_directory/new_file.doc");

            System.out.println("\n");
            //myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }
```

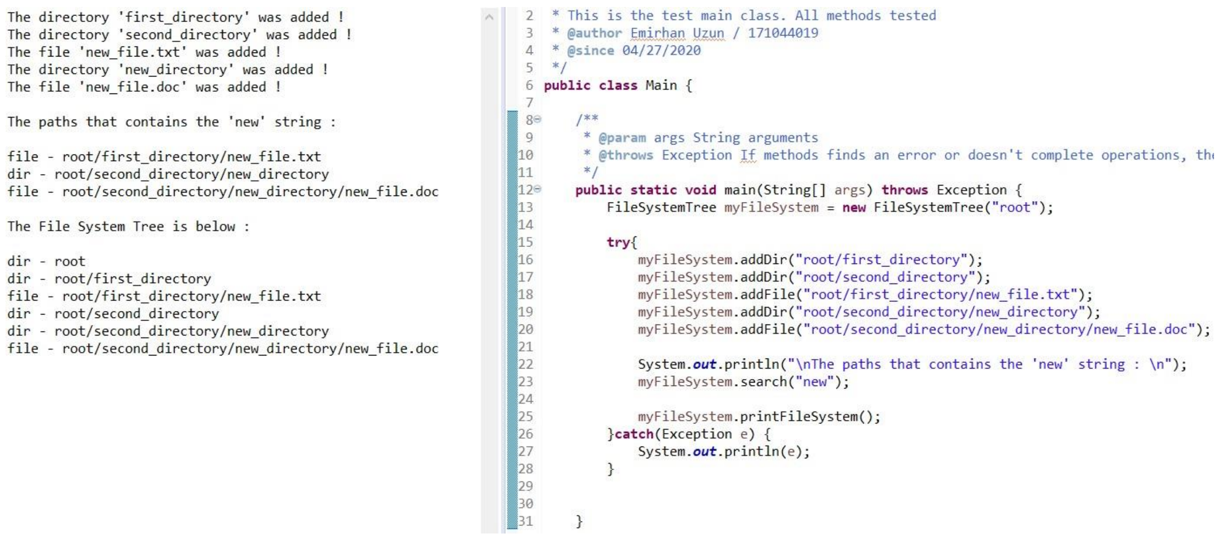
This is the addFile method with true path



```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
The file 'new_file.txt' was added !
The directory 'new_directory' was added !
java.lang.Exception: The file was not added !
```

```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete operations, th
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addFile("root/first_directory/new_file.txt");
            myFileSystem.addDir("root/second_directory/new_directory");
            myFileSystem.addFile("root/errorcomes/new_directory/new_file.doc");

            System.out.println("\n");
            //myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }
```

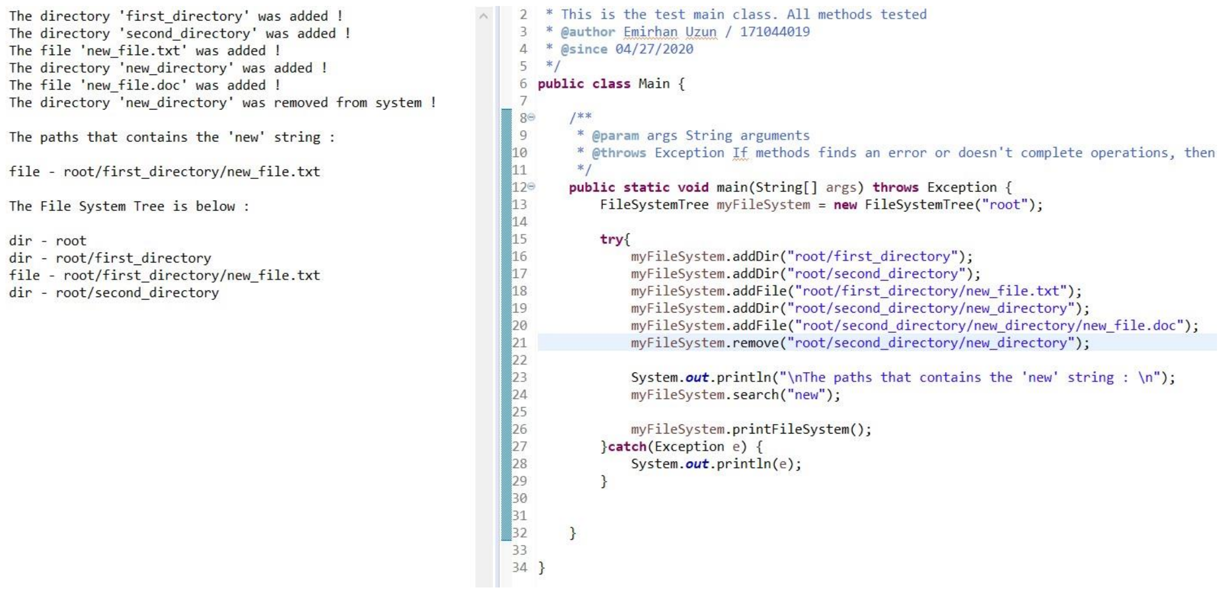
This is the addFile method with wrong path. It throws an exception



```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
The file 'new_file.txt' was added !
The directory 'new_directory' was added !
The file 'new_file.doc' was added !

The paths that contains the 'new' string :

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

The File System Tree is below :

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

```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete operations, th
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addFile("root/first_directory/new_file.txt");
            myFileSystem.addDir("root/second_directory/new_directory");
            myFileSystem.addFile("root/second_directory/new_directory/new_file.doc");

            System.out.println("\nThe paths that contains the 'new' string : \n");
            myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }
```

This is search method. It finds the contain given string and prints all finding object paths. The print method is in every method.



```
The directory 'first_directory' was added !
The directory 'second_directory' was added !
The file 'new_file.txt' was added !
The directory 'new_directory' was added !
The file 'new_file.doc' was added !
The directory 'new_directory' was removed from system !

The paths that contains the 'new' string :

file - root/first_directory/new_file.txt

The File System Tree is below :

dir - root
dir - root/first_directory
file - root/first_directory/new_file.txt
dir - root/second_directory
```

```java
 * This is the test main class. All methods tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or doesn't complete operations, then
     */
    public static void main(String[] args) throws Exception {
        FileSystemTree myFileSystem = new FileSystemTree("root");

        try{
            myFileSystem.addDir("root/first_directory");
            myFileSystem.addDir("root/second_directory");
            myFileSystem.addFile("root/first_directory/new_file.txt");
            myFileSystem.addDir("root/second_directory/new_directory");
            myFileSystem.addFile("root/second_directory/new_directory/new_file.doc");
            myFileSystem.remove("root/second_directory/new_directory");

            System.out.println("\nThe paths that contains the 'new' string : \n");
            myFileSystem.search("new");

            myFileSystem.printFileSystem();
        }catch(Exception e) {
            System.out.println(e);
        }


    }

}
```

This is the remove method. It works correctly

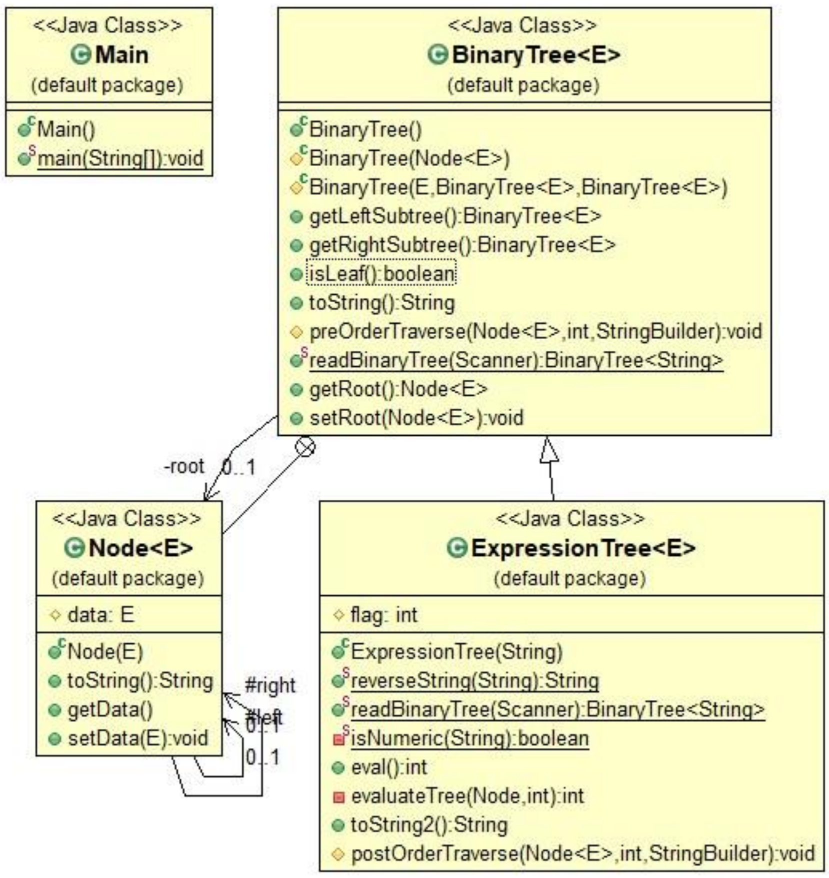
Commands

Actually I had a hard time with search method. Because I found node but I couldn't prints the path. But then I used a string to keep the path before the finding node.

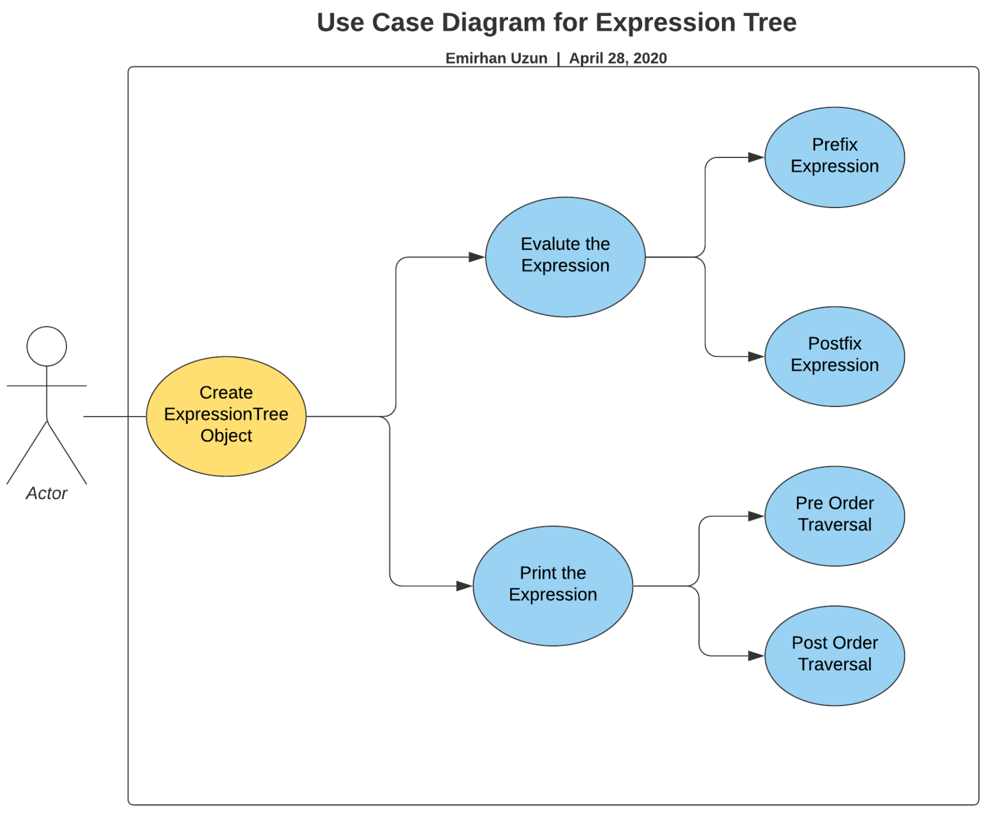
In this question I learnt that adding directories and files which means I learnt adding element to tree.

QUESTION 2

1) Class Diagram



2) Use Case Diagram



3) Problem Solution Approach

Firstly, I created a “flag” data field for the type of expression (Prefix or postfix). In constructor, I checked the string expression. If first character is not digit, then I determined this is prefix expression. But if it is digit, then I reverse the string. That means I did all operations according to the prefix expression. And then I send to the readBinaryTree methods to the create a tree.

Evaluate expression method works recursively. I have 2 base case. First of them if data is equal to null then return null. The second base case is if the node is leaf then return the integer type of node. I went left and right child in order and initialize them. Finally I did operations according to operator sign.

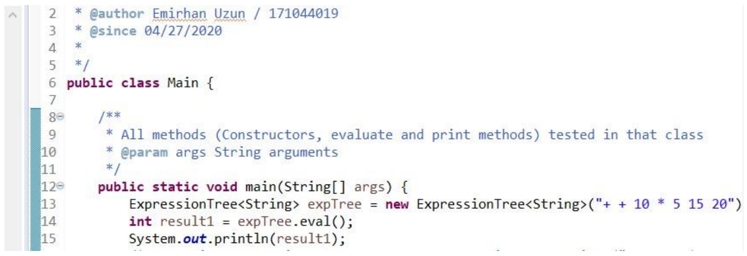
Moreover, I wrote the post order traverse method. It works recursively. It goes to the last left child. And then prints the right child and current node orderly.

4) Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Check the constructor	1.Write the expression name 2.Create the object	Expression string	Object should be created and root name must be initialized	As Expected	Pass
T02	Check read binary tree method		-Root - Expression	The tree has to be initialize according to string	As Expected	Pass
T03	Check eval method		-Root	The result of expression must be return truely,but has to use tree	As Expected	Pass
T04	Check preOrderTraversal method	1.Send the parameters to the method	-Root -String Builder buffer	The tree has to be printed pre order traversal	As Expected	Pass
T05	Check postOrderTraversal method	1.Send the parameters to the method	-Root -String Builder buffer	The tree has to be printed post order traversal	As Expected	Pass

5) Running Comand and Results

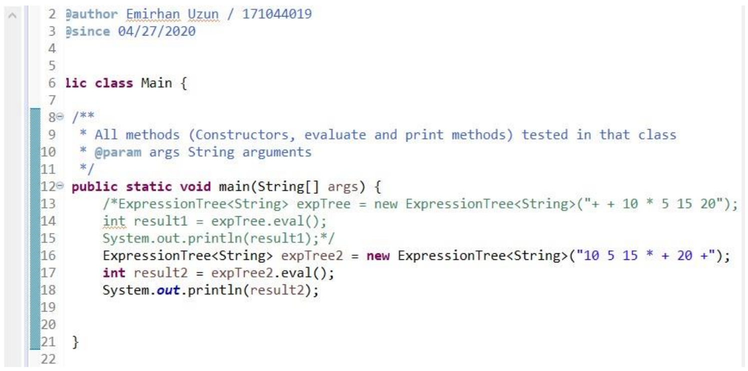
```
+ + 10 * 5 15 20
105
```



```
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 *
 */
public class Main {

    /**
     * All methods (Constructors, evaluate and print methods) tested in that class
     * @param args String arguments
     */
    public static void main(String[] args) {
        ExpressionTree<String> expTree = new ExpressionTree<String>("+ + 10 * 5 15 20")
        int result1 = expTree.eval();
        System.out.println(result1);
```

This is constructor, readBinaryTree and eval method test. It returns true result. That means the constructor and readBinaryTree method works correctly

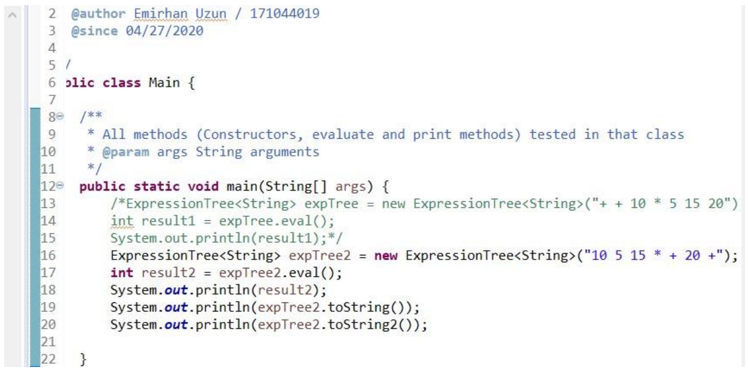
```
20 15 5 * 10 + +
105
```



```
@author Emirhan Uzun / 171044019
@since 04/27/2020


lic class Main {

 /**
   * All methods (Constructors, evaluate and print methods) tested in that class
   * @param args String arguments
   */
 public static void main(String[] args) {
     /*ExpressionTree<String> expTree = new ExpressionTree<String>("+ + 10 * 5 15 20");
     int result1 = expTree.eval();
     System.out.println(result1);*/
     ExpressionTree<String> expTree2 = new ExpressionTree<String>("10 5 15 * + 20 +");
     int result2 = expTree2.eval();
     System.out.println(result2);


 }
```

This is the postfix expression test. The eval method also work correctly.

```
20 15 5 * 10 + +
105
+ 20 + * 15 5 10
20 15 5 * 10 + +
```



```
 @author Emirhan Uzun / 171044019
 @since 04/27/2020

/
blic class Main {

  /**
    * All methods (Constructors, evaluate and print methods) tested in that class
    * @param args String arguments
    */
  public static void main(String[] args) {
      /*ExpressionTree<String> expTree = new ExpressionTree<String>("+ + 10 * 5 15 20")
      int result1 = expTree.eval();
      System.out.println(result1);*/
      ExpressionTree<String> expTree2 = new ExpressionTree<String>("10 5 15 * + 20 +");
      int result2 = expTree2.eval();
      System.out.println(result2);
      System.out.println(expTree2.toString());
      System.out.println(expTree2.toString2());

  }
```

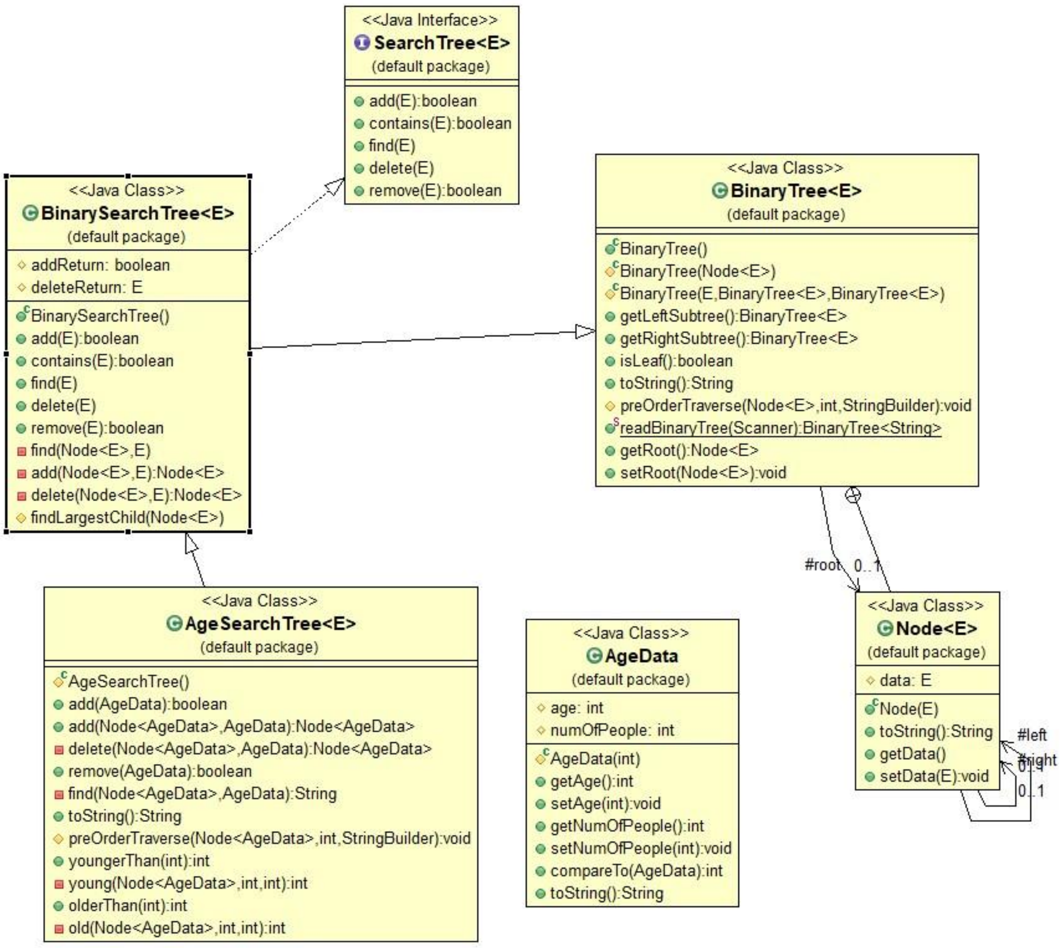
This is the post order traversal and pre order traversal test. I use toString() and toString2() method. So these methods work correctly

Commands

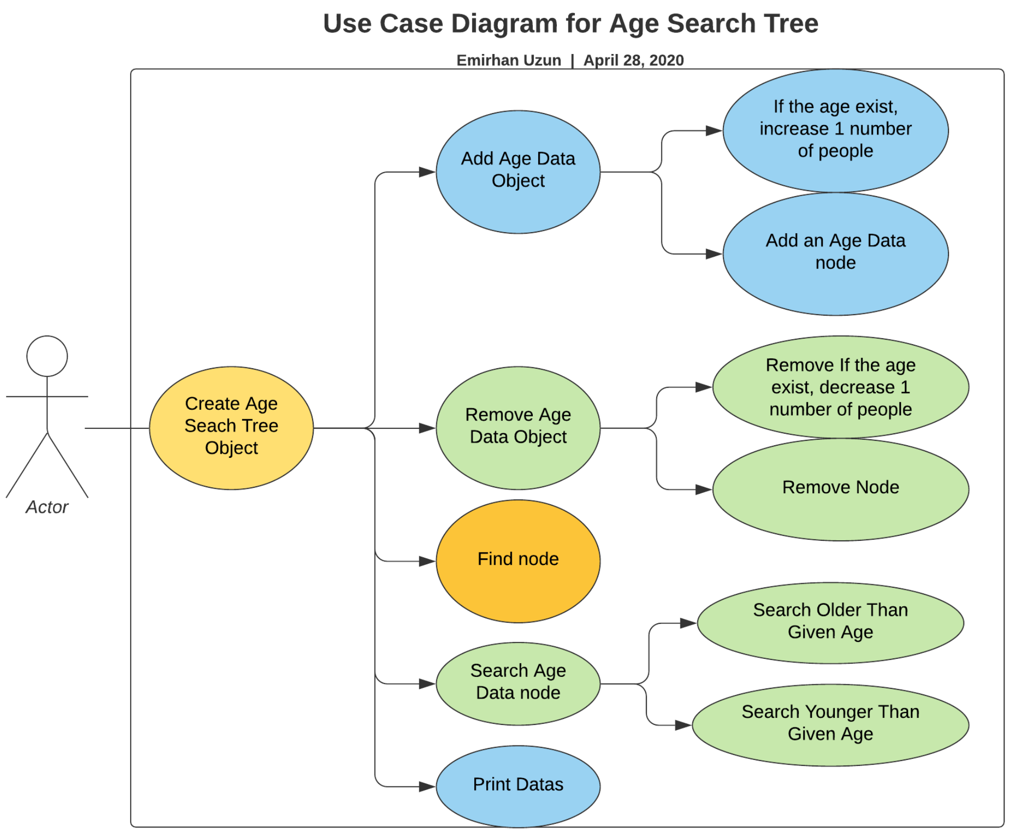
In this question, I learnt create a tree from given string. Also I used the evaluation of those nodes which means I can use the nodes whenever I need it

QUESTION 3

1) Class Diagram



2) Use Case Diagram


Use Case Diagram for Age Search Tree
Emirhan Uzun | April 28, 2020
Actor
Create Age Seach Tree Object
Add Age Data Object
If the age exist, increase 1 number of people
Add an Age Data node
Remove Age Data Object
Remove If the age exist, decrease 1 number of people
Remove Node
Find node
Search Age Data node
Search Older Than Given Age
Search Younger Than Given Age
Print Datas

3) Problem Solution Approach

In this question, AgeData class is my node class. That means, AgeSearchTree class' nodes are consist of AgeData objects. The age data class holds the age of people and people number.

In Age Search Tree class' constructor initializes the root to null. In add method, if the age is exist in tree, it just increases the people number 1. In remove method, if the age is exist, then check the people number of that node. If it is greater than 1, it decreases people number 1. Otherwise it removes the node. After the remove, it finds the largest child and initializes that to the removing node's place.

In younger than method, it goes to the smaller age from given age. And counter increases that ages' people number.

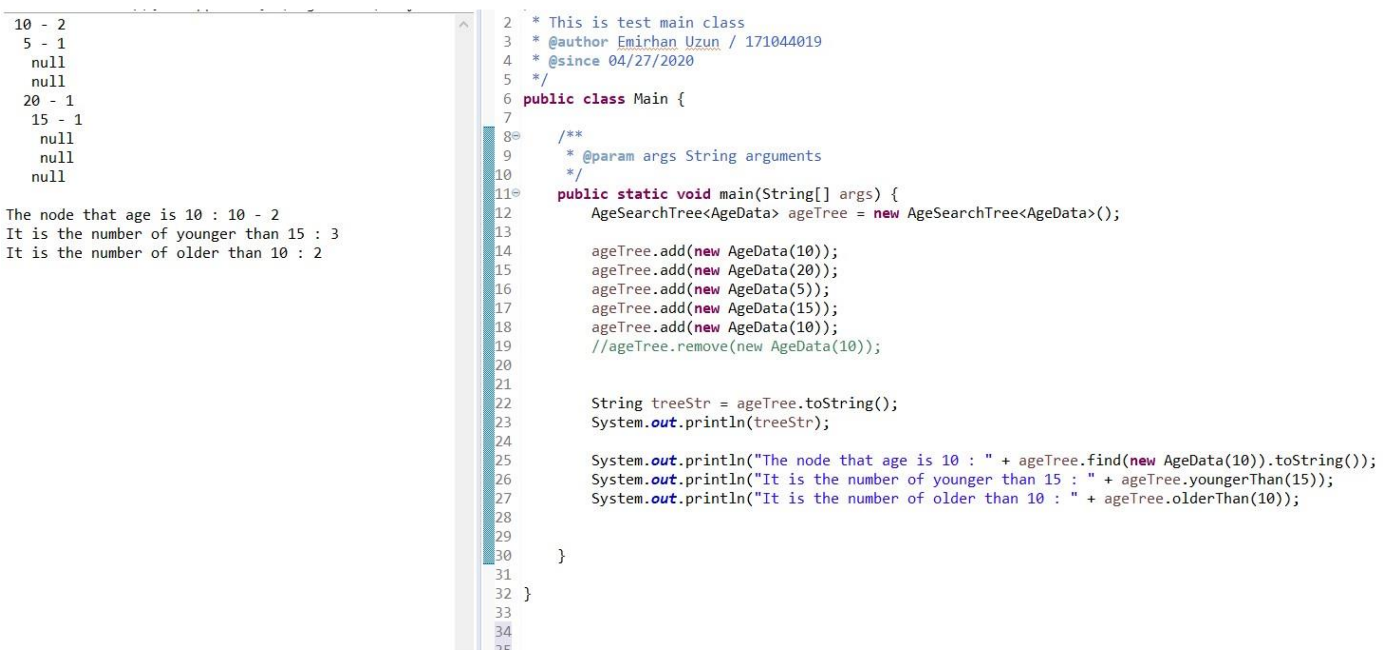
In older than method, it goes to the bigger age from given age. And counter increases that ages' people number.

To string method, prints the age and number of people that node. It works recursively. First it prints the current node. And then left and right child in order.

4) Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Check the constructor	1.Create the object	Root	Object should be created and root must be initialized to null	As Expected	Pass
T02	Check add method(increase the number of people)	1.Create a age data node and add it to list	-List - Age Data node	The node should be created and if that age is equal to the nodes of list, increase the people number	As Expected	Pass
T03	Check add method	1.Create a age data node and add it to tree	-Root -AgeData node	The node should be created and added to root	As Expected	Pass
T04	Check remove method (decrease the people number)	1.Create a new age data node 2.Compare this age with tree nodes	-Root -Age Data node	The node should be created and if the age is equal to the nodes of tree,decrease the people number	As Expected	Pass
T05	Check remove method	1.Create a new age data node 2.Compare this age with tree nodes	-Root -Age Data node	The node should be created and if the age is equal to the nodes of tree, remove it from tree	As Expected	Pass
T06	Check find method	1.Create a new age data node 2.Compare this age with tree nodes	-Root -Age Data node	The node should be created and if tree contains that age, prints the age and number of people	As Expected	Pass
T07	Check youngerThan method	1.Write to age which we find younger than that	-Root -Searching age	If the nodes have younger than that age, it prints the count of that people	As Expected	Pass
T08	Check olderThan method	1.Write to age which we find younger than that	-Root, -Searching age	If the nodes have older than that age, it prints the count of that people	As Expected	Pass
T09	Check toString method	1.Prints the nodes	-Root	It prints the nodes of tree with age and people number.It uses pre order traversal method	As Expected	Pass

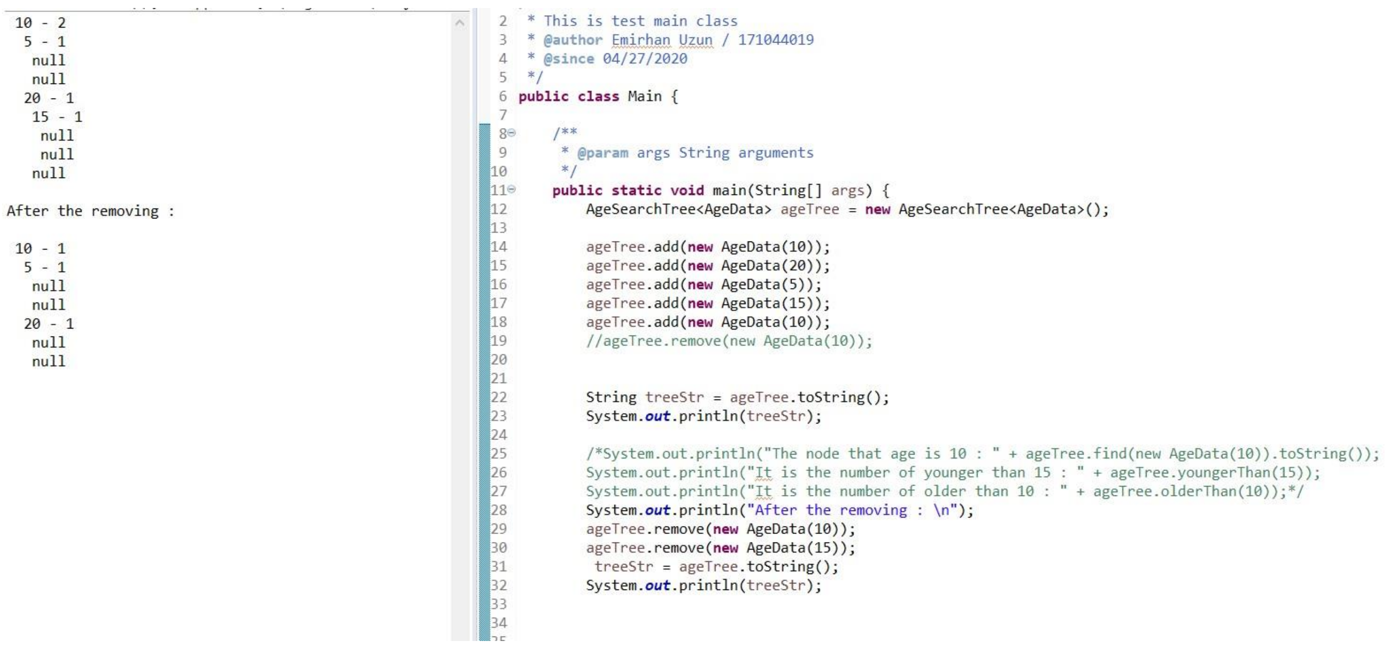
5) Running Command and Results



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

The node that age is 10 : 10 - 2
It is the number of younger than 15 : 3
It is the number of older than 10 : 2
```

```java
 * This is test main class
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     */
    public static void main(String[] args) {
        AgeSearchTree<AgeData> ageTree = new AgeSearchTree<AgeData>();

        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.remove(new AgeData(10));


        String treeStr = ageTree.toString();
        System.out.println(treeStr);

        System.out.println("The node that age is 10 : " + ageTree.find(new AgeData(10)).toString());
        System.out.println("It is the number of younger than 15 : " + ageTree.youngerThan(15));
        System.out.println("It is the number of older than 10 : " + ageTree.olderThan(10));


    }

}
```

This screenshot includes the add, add (increase), find, youngerThan, olderThan and toString method. All of them works correctly.



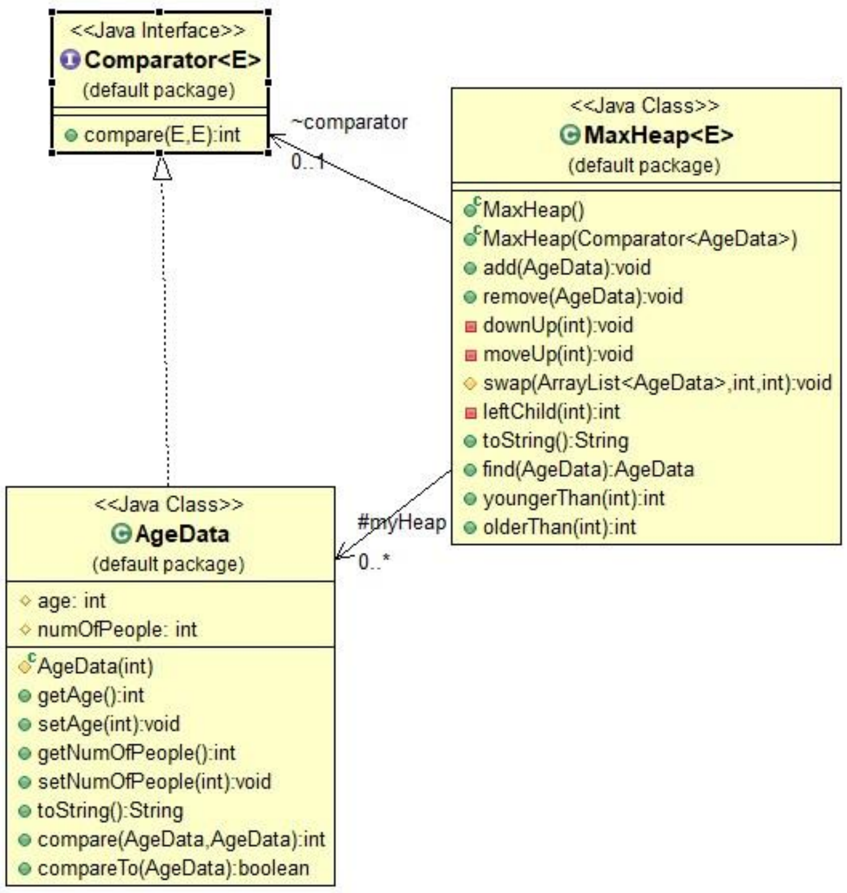
```
 10 - 2
  5 - 1
   null
   null
  20 - 1
   15 - 1
    null
    null
   null

After the removing :

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

```java
 * This is test main class
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 */
public class Main {

    /**
     * @param args String arguments
     */
    public static void main(String[] args) {
        AgeSearchTree<AgeData> ageTree = new AgeSearchTree<AgeData>();

        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.remove(new AgeData(10));


        String treeStr = ageTree.toString();
        System.out.println(treeStr);

        /*System.out.println("The node that age is 10 : " + ageTree.find(new AgeData(10)).toString());
        System.out.println("It is the number of younger than 15 : " + ageTree.youngerThan(15));
        System.out.println("It is the number of older than 10 : " + ageTree.olderThan(10));*/
        System.out.println("After the removing : \n");
        ageTree.remove(new AgeData(10));
        ageTree.remove(new AgeData(15));
         treeStr = ageTree.toString();
        System.out.println(treeStr);
```

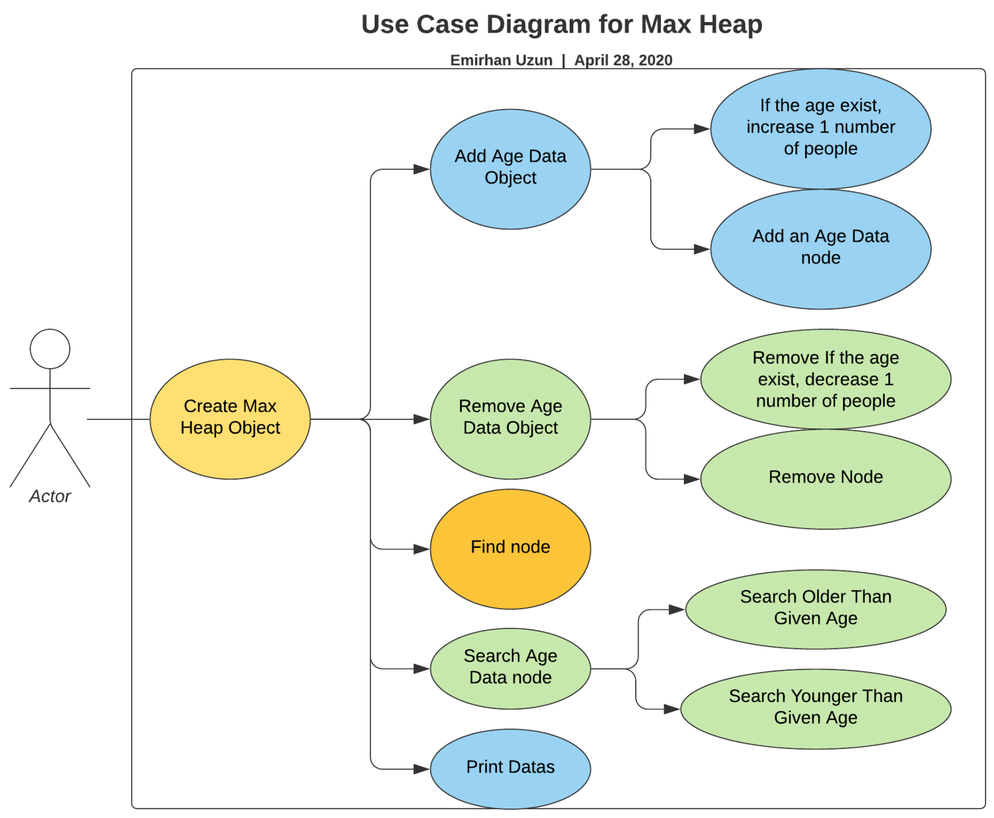
This screenshot includes the remove methods. All of them works correctly.

QUESTION 4

1) Class Diagram



2) Use Case Diagram



Use Case Diagram for Max Heap
Emirhan Uzun | April 28, 2020
Actor
Create Max Heap Object
Add Age Data Object
If the age exist, increase 1 number of people
Add an Age Data node
Remove Age Data Object
Remove If the age exist, decrease 1 number of people
Remove Node
Find node
Search Age Data node
Search Older Than Given Age
Search Younger Than Given Age
Print Datas

3) Problem Solution Approach

This question is very similar with question 3. The differences are, this uses the array list for nodes and age with the highest number node will be at root.

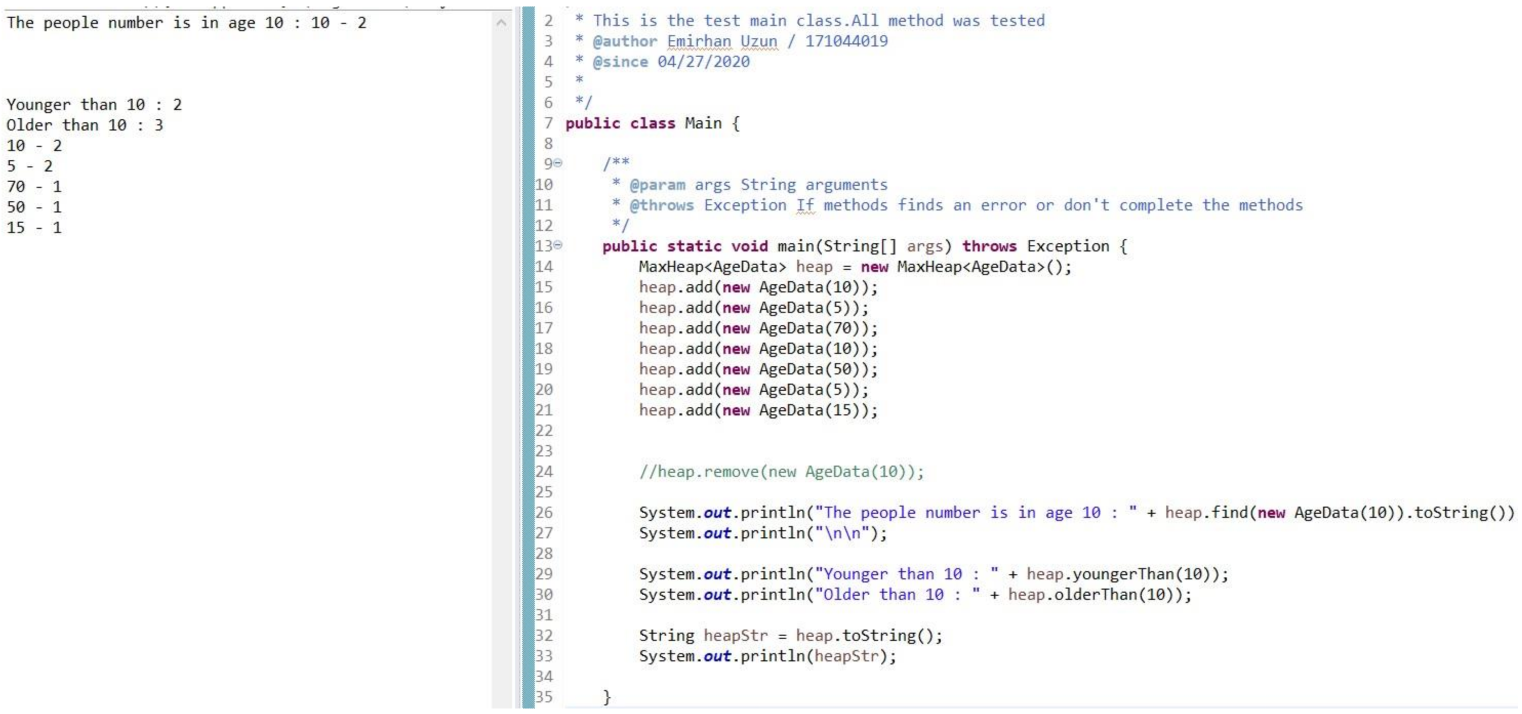
Array list keeps the ageData nodes. After every add and remove method, I check the number of people and shift the nodes up or down according to the situation.

The find method searchs the whole list and if it equals to given ageData method, then prints it.

4) Test Cases

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Check the constructor	1.Create the object	- List	Object should be created and list must be initialized to array list	As Expected	Pass
T02	Check add method(increase the number of people)	1.Create a age data node and add it to list	-List - Age Data node	The node should be created and if that age is equal to the nodes of list, increase the people number	As Expected	Pass
T03	Check add method	1.Create a age data node and add it to list	- Array List -AgeData node	The node should be created and added to list	As Expected	Pass
T04	Check remove method (decrease the people number)	1.Create a new age data node 2.Compare this age with list nodes	- Array List -Age Data node	The node should be created and if the age is equal to the nodes of list,decrease the people number	As Expected	Pass
T05	Check remove method	1.Create a new age data node 2.Compare this age to list nodes	- Array List -Age Data node	The node should be created and if the age is equal to the nodes of list, remove it from tree	As Expected	Pass
T06	Check find method	1.Create a new age data node 2.Compare this age to list nodes	- Array List -Age Data node	The node should be created and if list contains that age, prints the age and number of people	As Expected	Pass
T07	Check youngerThan method	1.Write to age which we find younger than that	- Array List -Searching age	If the list has younger than that age, it prints the count of that people	As Expected	Pass
T08	Check olderThan method	1.Write to age which we find younger than that	- Array List, -Searching age	If the list has older than that age, it prints the count of that people	As Expected	Pass
T09	Check toString method	1.Prints the nodes (Root has max people number)	-Array List	It prints the nodes of list with age and people number.	As Expected	Pass

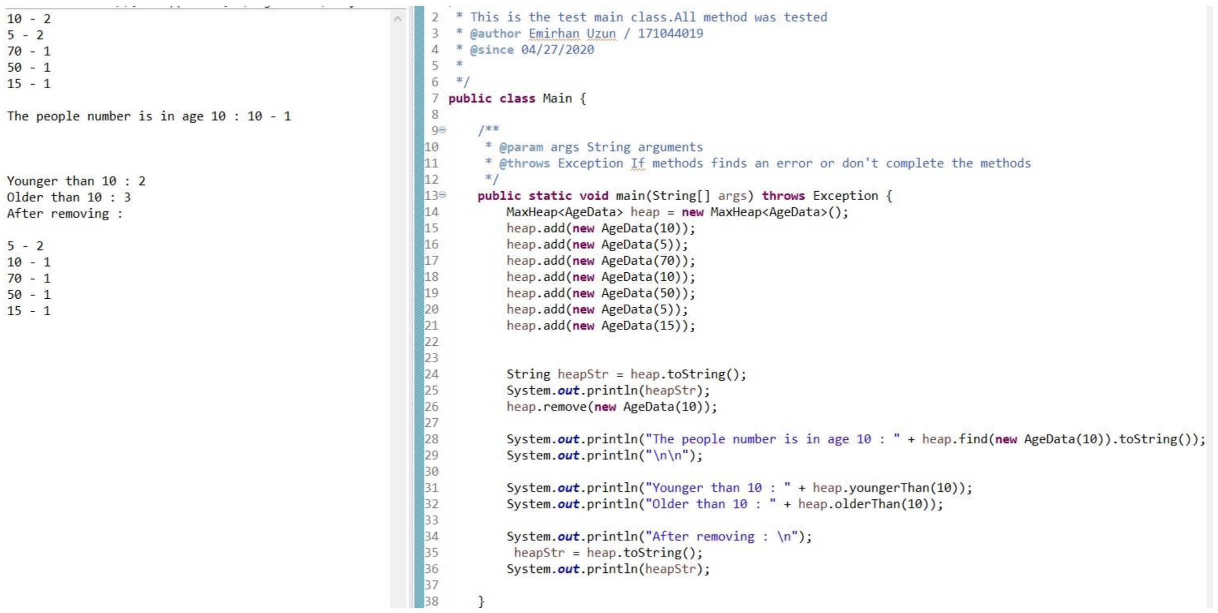
5) Running Command and Results



```
The people number is in age 10 : 10 - 2



Younger than 10 : 2
Older than 10 : 3
10 - 2
5 - 2
70 - 1
50 - 1
15 - 1
```

```java
 * This is the test main class.All method was tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 *
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or don't complete the methods
     */
    public static void main(String[] args) throws Exception {
        MaxHeap<AgeData> heap = new MaxHeap<AgeData>();
        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("The people number is in age 10 : " + heap.find(new AgeData(10)).toString())
        System.out.println("\n\n");

        System.out.println("Younger than 10 : " + heap.youngerThan(10));
        System.out.println("Older than 10 : " + heap.olderThan(10));

        String heapStr = heap.toString();
        System.out.println(heapStr);

    }
```

This screenshot includes the add, add(increase), olderThan, youngerThan, find methods. All of them works correctly.



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

The people number is in age 10 : 10 - 1



Younger than 10 : 2
Older than 10 : 3
After removing :

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

```java
 * This is the test main class.All method was tested
 * @author Emirhan Uzun / 171044019
 * @since 04/27/2020
 *
 */
public class Main {

    /**
     * @param args String arguments
     * @throws Exception If methods finds an error or don't complete the methods
     */
    public static void main(String[] args) throws Exception {
        MaxHeap<AgeData> heap = new MaxHeap<AgeData>();
        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);
        heap.remove(new AgeData(10));

        System.out.println("The people number is in age 10 : " + heap.find(new AgeData(10)).toString());
        System.out.println("\n\n");

        System.out.println("Younger than 10 : " + heap.youngerThan(10));
        System.out.println("Older than 10 : " + heap.olderThan(10));

        System.out.println("After removing : \n");
         heapStr = heap.toString();
        System.out.println(heapStr);

    }
```

This screenshot includes the remove method. Also it works correctly.

Commands

In this part, I learnt the usage of list with nodes and compare with child and parent nodes. Because though the list is in order, parent and children are not in order. So we have to find children or parent and compare them. And also I learnt what comparator is and I used it.