Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

FERHAT ŞİRİN
161044080

Course Assistant:

1 INTRODUCTION

1.1 Problem Definition

Part 1)

General tree can have multiple child tree which can vary from each other. To represent a general tree we need a binary tree which first child is left node and others is right nodes of that left node.

Part 2)

Multidimensional search tree is a tree which every node is compared to others node related to their dimension. Left nodes is for smaller element at that dimension and right nodes for bigger element at that dimension. For each level down dimension increases.

1.2 System Requirements

Part 1)

GeneralTree clas do not have any exception. Javadoc is written for all method about how to use them.

Part 2)

MultiDimenTree class is used with data types that implements List interface. (Vector, ArrayList, LinkedList..) Comparator class is needed for searching algorithm.

Class can be used like that.

CompareHelper<Integer> comp =new CompareHelper<>();

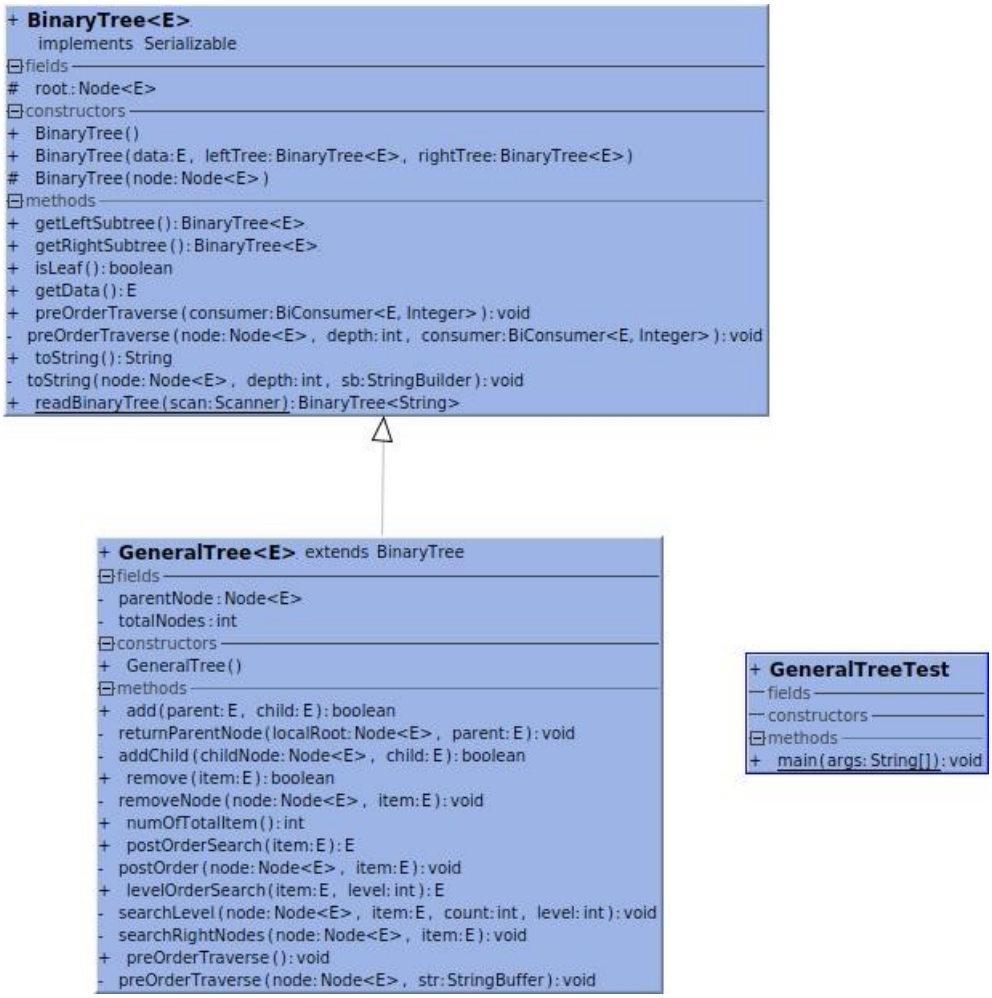
Comparator class for Integer type. This will be used in the constructor of MultiDimenTree class.

```
MultiDimenTree<Vector<Integer>> tree1 = new MultiDimenTree<>(comp);
Vector<Integer>[] arrList =new Vector[15];
arrList[0] =new Vector<>(Arrays.asList(new Integer[]{40,45,50}));
arrList[1] =new Vector<>(Arrays.asList(new Integer[]{20,70,60}))

Arrays.asList return a list with given array. This is used to make
adding operation easy.

Tree1.add(arrList[0]); tree1.add(arrList[1]);
```

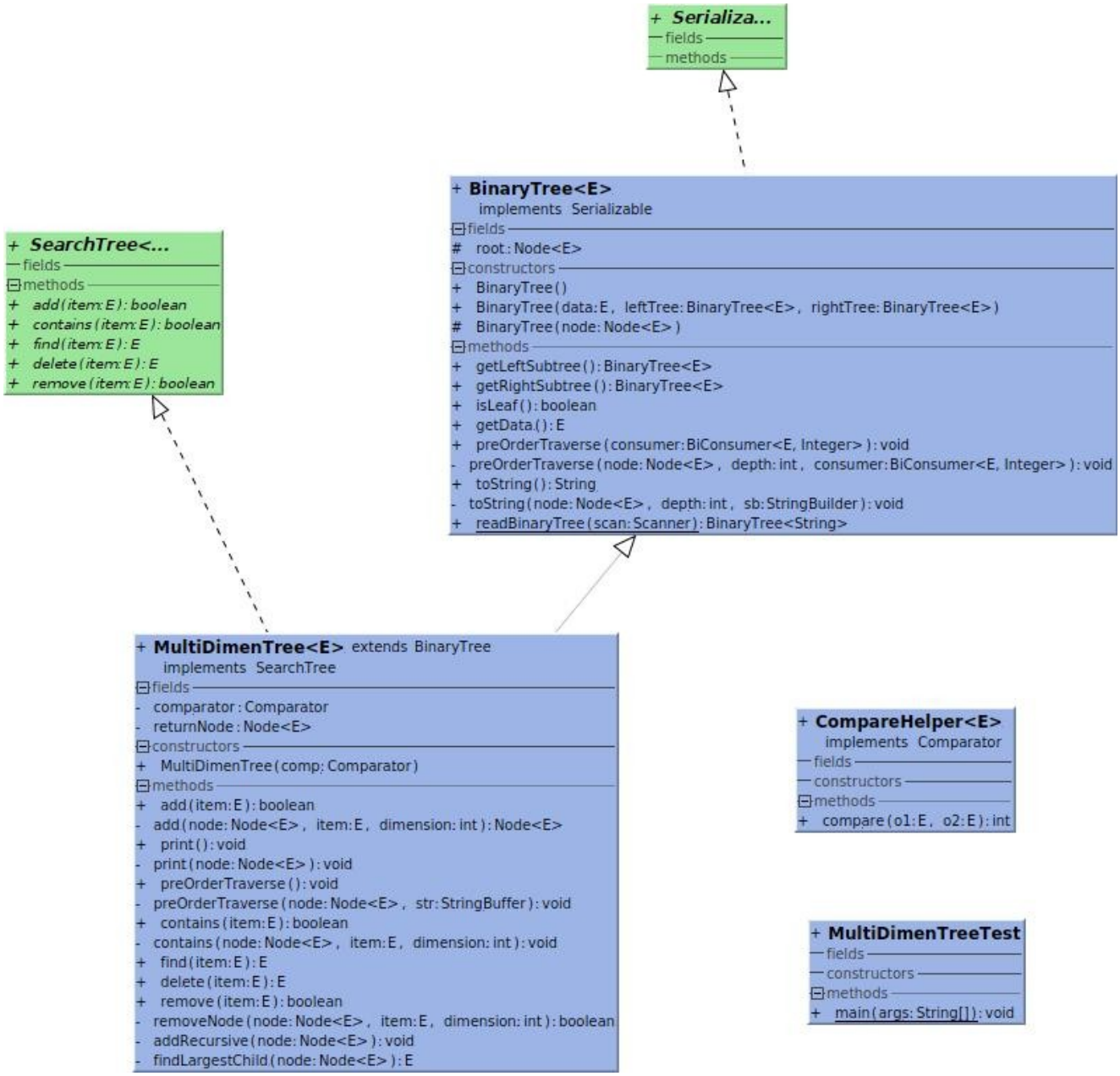
2 METHOD

2.1 Class Diagrams

Part 1 :

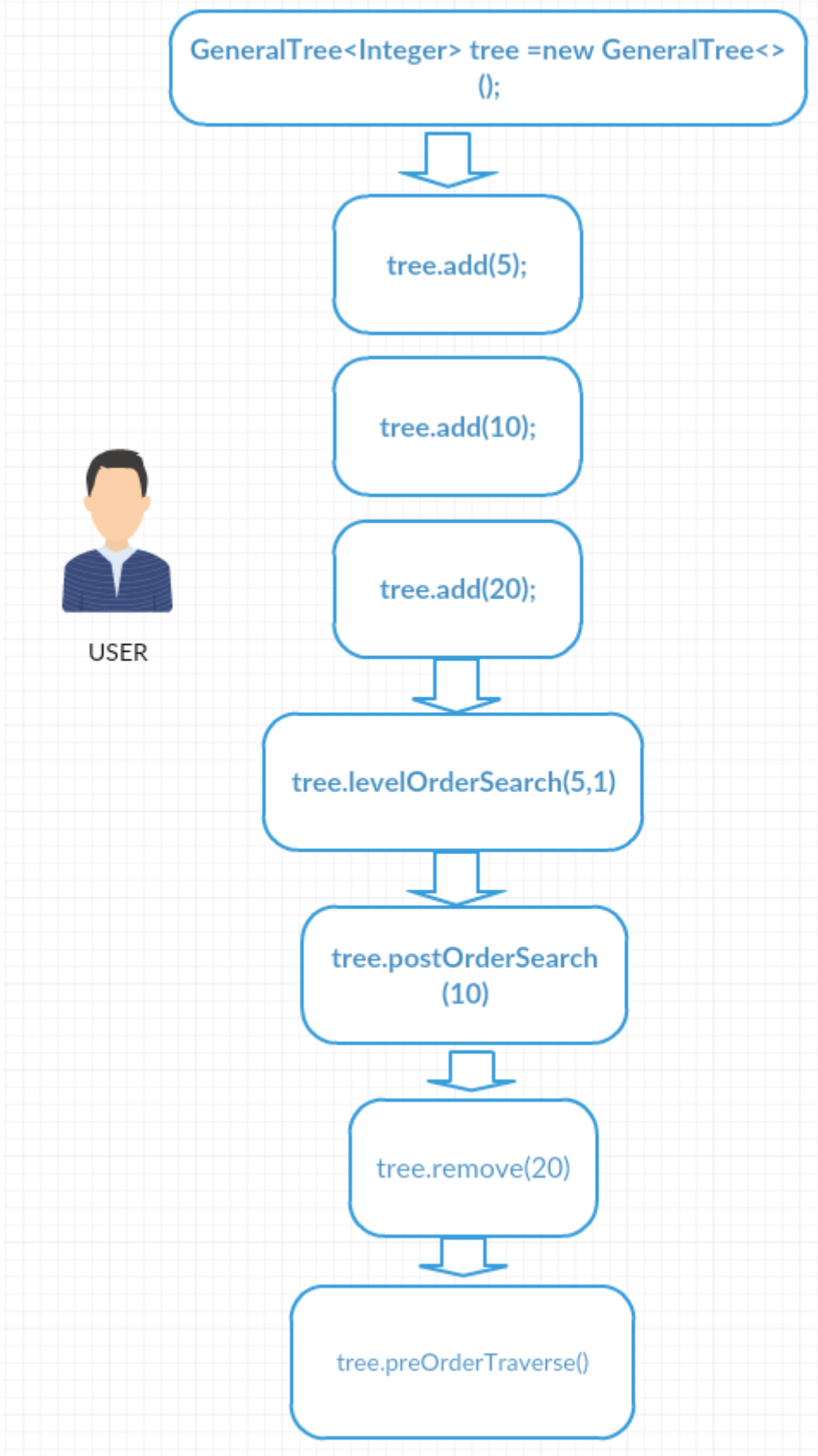


Part 2:


+ Serializa...
fields
methods
+ BinaryTree<E>
implements Serializable
fields
root : Node<E>
constructors
+ BinaryTree ( )
+ BinaryTree (data: E , leftTree: BinaryTree<E> , rightTree: BinaryTree<E>)
# BinaryTree (node: Node<E>)
methods
+ getLeftSubtree ( ) : BinaryTree<E>
+ getRightSubtree ( ) : BinaryTree<E>
+ isLeaf ( ) : boolean
+ getData ( ) : E
+ preOrderTraverse (consumer: BiConsumer<E, Integer>) : void
- preOrderTraverse (node: Node<E> , depth: int , consumer: BiConsumer<E, Integer>) : void
+ toString ( ) : String
- toString (node: Node<E> , depth: int , sb: StringBuilder) : void
+ readBinaryTree (scan: Scanner) : BinaryTree<String>
+ SearchTree<...
fields
methods
+ add (item: E) : boolean
+ contains (item: E) : boolean
+ find (item: E) : E
+ delete (item: E) : E
+ remove (item: E) : boolean
+ MultiDimenTree<E> extends BinaryTree
implements SearchTree
fields
- comparator : Comparator
- returnNode : Node<E>
constructors
+ MultiDimenTree (comp: Comparator)
methods
+ add (item: E) : boolean
- add (node: Node<E> , item: E , dimension: int) : Node<E>
+ print ( ) : void
- print (node: Node<E>) : void
+ preOrderTraverse ( ) : void
- preOrderTraverse (node: Node<E> , str: StringBuffer) : void
+ contains (item: E) : boolean
- contains (node: Node<E> , item: E , dimension: int) : void
+ find (item: E) : E
+ delete (item: E) : E
+ remove (item: E) : boolean
- removeNode (node: Node<E> , item: E , dimension: int) : boolean
- addRecursive (node: Node<E>) : void
- findLargestChild (node: Node<E>) : E
+ CompareHelper<E>
implements Comparator
fields
constructors
methods
+ compare (o1: E , o2: E) : int
+ MultiDimenTreeTest
fields
constructors
methods
+ main (args: String[]) : void

2.2 Use Case Diagrams

Part 1 :



Part 2 :


MultiDimenTree<Vector<Integer>> tree =new MultiDimenTree<>(new CompareHelper<Integer>());
tree.add(new Vector<>(Arrays.asList(new Integer[]{40,45,50})));
tree.add(new Vector<>(Arrays.asList(new Integer[]{10,25,48})));
USER
tree.contains(new Vector<>(Arrays.asList(new Integer[]{40,45,51})));
tree.find(new Vector<>(Arrays.asList(new Integer[]{40,45,51})));
tree.delete(new Vector<>(Arrays.asList(new Integer[]{40,45,51})));
tree.remove(new Vector<>(Arrays.asList(new Integer[]{40,45,51})));
tree.preOrderTraverse()

2.3 Problem Solution Approach

Part 1)

All function uses recursive methods to solve problem. They have helper function to use recursive method with Node<E> class. These function private because Node<E> is private and root can not be accessed outside. Add function is used with a parent item to specify which parent child node is added to. Child node can be null when adding first node.

Remove method removes the leaf nodes because tree is general.

Part 2)

Multidimensional search tree is implemented in this part. For data type for this class should extend List interface. Every class implementing List interface can be used with this class. (Vector , ArrayList, LinkedList...) and a proper Comparator class should be given for MultiDimenTree class. For Comparator class ComparatorHelper class can be used. All methods have helper function inside to solve the problem recursively.

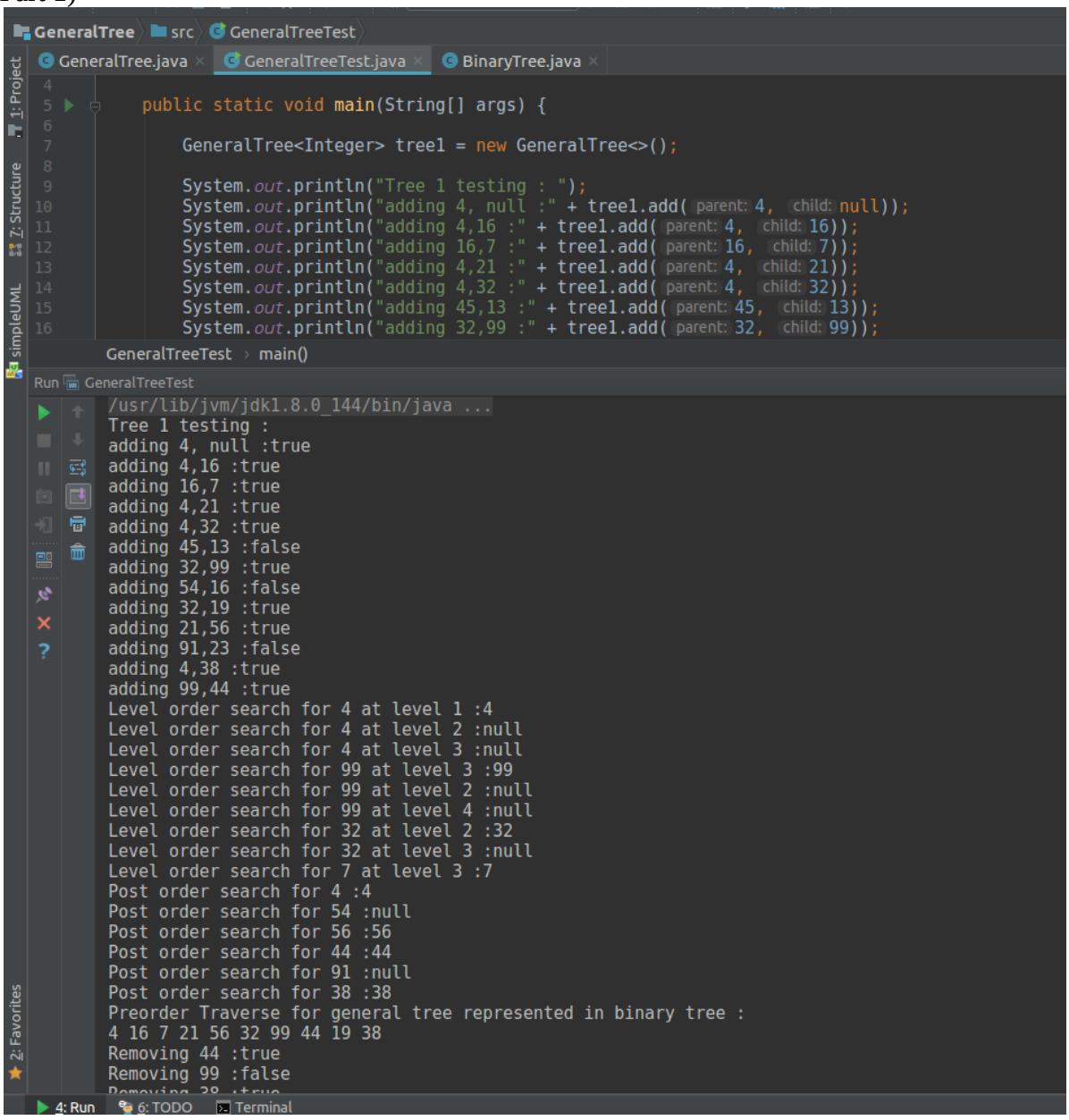
3 RESULT

3.1 Test Cases

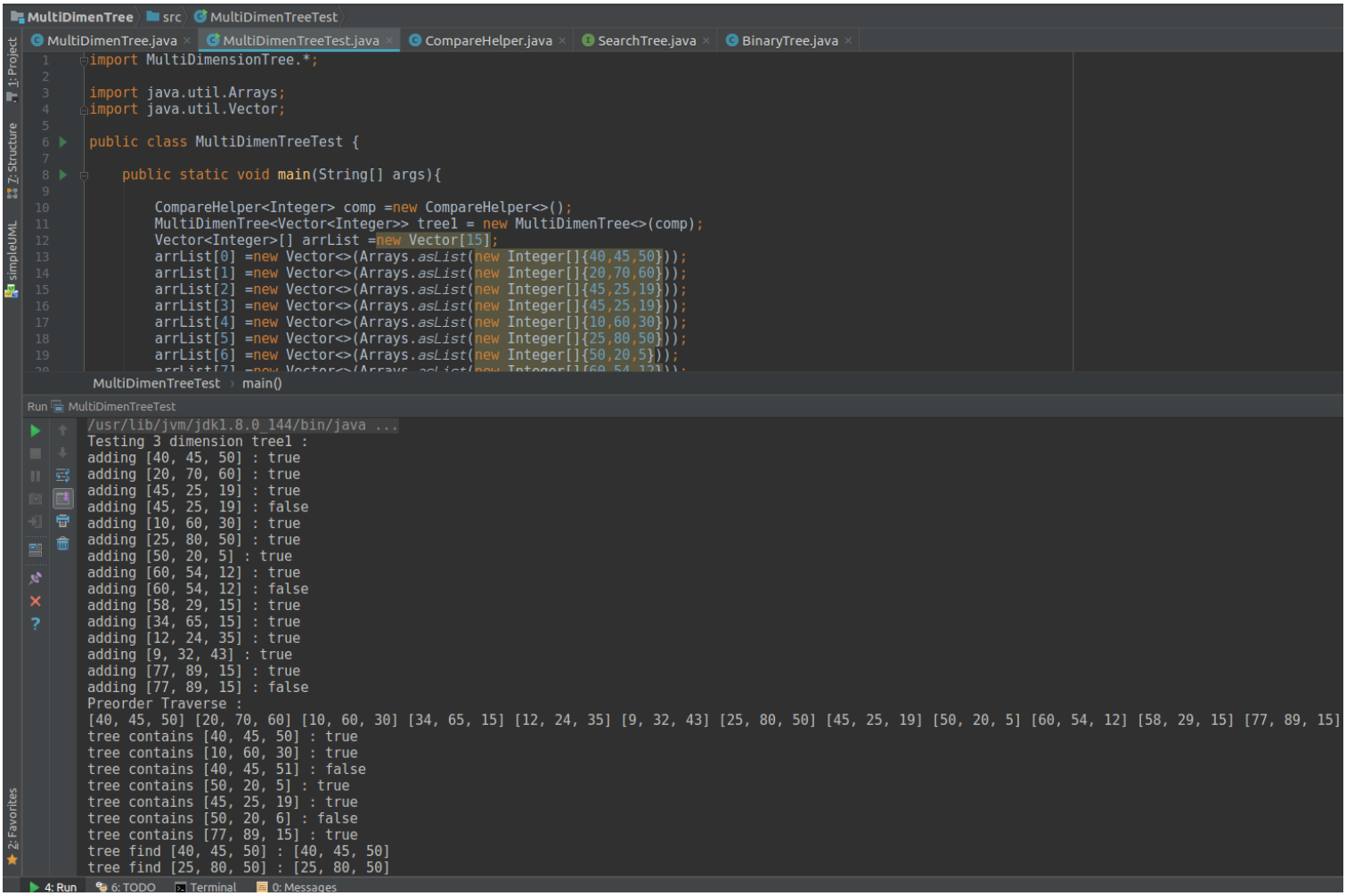
All test cases are written to the main function of the programs.

3.2 Running Results

Part 1)



Part 2)


MultiDimenTree
src
MultiDimenTreeTest
MultiDimenTree.java
MultiDimenTreeTest.java
CompareHelper.java
SearchTree.java
BinaryTree.java
import MultiDimensionTree.*;
import java.util.Arrays;
import java.util.Vector;
public class MultiDimenTreeTest {
public static void main(String[] args){
CompareHelper<Integer> comp =new CompareHelper<>();
MultiDimenTree<Vector<Integer>> tree1 = new MultiDimenTree<>(comp);
Vector<Integer>[] arrList =new Vector[15];
arrList[0] =new Vector<>(Arrays.asList(new Integer[]{40,45,50}));
arrList[1] =new Vector<>(Arrays.asList(new Integer[]{20,70,60}));
arrList[2] =new Vector<>(Arrays.asList(new Integer[]{45,25,19}));
arrList[3] =new Vector<>(Arrays.asList(new Integer[]{45,25,19}));
arrList[4] =new Vector<>(Arrays.asList(new Integer[]{10,60,30}));
arrList[5] =new Vector<>(Arrays.asList(new Integer[]{25,80,50}));
arrList[6] =new Vector<>(Arrays.asList(new Integer[]{50,20,5}));
MultiDimenTreeTest › main()
Run MultiDimenTreeTest
/usr/lib/jvm/jdk1.8.0_144/bin/java ...
Testing 3 dimension tree1 :
adding [40, 45, 50] : true
adding [20, 70, 60] : true
adding [45, 25, 19] : true
adding [45, 25, 19] : false
adding [10, 60, 30] : true
adding [25, 80, 50] : true
adding [50, 20, 5] : true
adding [60, 54, 12] : true
adding [60, 54, 12] : false
adding [58, 29, 15] : true
adding [34, 65, 15] : true
adding [12, 24, 35] : true
adding [9, 32, 43] : true
adding [77, 89, 15] : true
adding [77, 89, 15] : false
Preorder Traverse :
[40, 45, 50] [20, 70, 60] [10, 60, 30] [34, 65, 15] [12, 24, 35] [9, 32, 43] [25, 80, 50] [45, 25, 19] [50, 20, 5] [60, 54, 12] [58, 29, 15] [77, 89, 15]
tree contains [40, 45, 50] : true
tree contains [10, 60, 30] : true
tree contains [40, 45, 51] : false
tree contains [50, 20, 5] : true
tree contains [45, 25, 19] : true
tree contains [50, 20, 6] : false
tree contains [77, 89, 15] : true
tree find [40, 45, 50] : [40, 45, 50]
tree find [25, 80, 50] : [25, 80, 50]
4: Run
6: TODO
Terminal
0: Messages