

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

**BURAK DEMİRCİ
141044091**

Course Assistant: Mehmet Burak KOCA

1 INTRODUCTION

1.1 Problem Definition

Part1: Bu bölümde Binary Tree Class'ını extend ederek General Tree yapısında bir sistem elde etmemiz bekleniyor ek olarak;

- levelOrderSearch () ,verilen elemanı tree yapısında arayıp bulunduğunda (true) bulamadığında (false) döndürür.
- postOrderSearch() ,verilen elemanı tree yapısında arayıp bulunduğunda Node<E> tree yapısının bir birimi (elemanın tutulma biçimi) şeklinde bulamadığında boş (null) döndürür.
- preOrderTraverse() ,verilen elemandan başlayarak ağacı (tree) dolaşp elemanların verisini (data) bir string yapısına ekler.

preOrderTraverse() yapısının Override edilmesi bekleniyor fakat Binary Tree içerisinde bulunan bu method private olarak tanımlanmıştır private methodu Override edemediğimiz için bununla alakalı farklı bir çözüm getirdim.

Part2: Bu bölümde Binary Tree Class'ı nı extend ederek ve Search Interfacesi ni implemnts ederek Multidimention obje alan bir MultidimentionSearchTree gerçeklememiz (imlement) bekleniyor. Multidimention Class'ı;

Multidimention (x,y,z) ve Multidimention (count) şeklinde iki constructoru olan bir class implement etmemiz ve bu classın objelerini (multidimensional search tree) yapısında tutmamız ve üzerinde işlem yapmamız bekleniyor.

MultidimensionSearchTree üzerinde ise aşağıdaki methodları gerçeklemememiz gerekiyor.

- boolean add(E item) : Tree yapısına yeni eleman ekler
- boolean contains(E target) : Tree yapısında verilen elemanın olup olmadığını sorgular
- E find(E target) : Verilen elemanı tree yapısında arar
- E delete(E target) : Verilen elemanı tree üzerinden siler
- boolean remove(E target) : Verilen elemanı tree'den kaldırır

Verilen elemanı ekleme işleminde ise x,y,z eksenlerine bakılarak büyüklüklere göre Tree yapısına homojen olarak dağılması gerekmektedir. Arama ve silme işlemleri de bu mantığa göre ilerlemelidir.

Silme işlemi gerçekleştirirken silinen elemanın çocukları silinme işleminden sonra aynı sistemle tekrardan tree üzerine yerleştirilir.

1.2 System Requirements

Part1: Bu bölümde Binary Tree Classı extends edileceği için bu Classın ders kitabından alınmıştır (@author Koffman and Wolfgang).

- levelOrderSearch () , methodu => boolean levelOrderSearch (Node<E> root , E target)
- postOrderSearch() , methodu => Node<E> postOrderSearch(Node<E> root , E target)
- preOrderTraverse() , methodu => void preOrderTraverse(Node<E> root,StringBuffer str)

Yukarıdaki sistemler için gerekli olan parametreler verildiği gibi tanımlanmıştır. Sistemlerin test edilebilmesi için MainPart1.java Classı implement edilmiştir.

Part2: Bu bölümde Binary Tree Classı extends edilecek ve SearchTree Interface'si implements edileceği için bu yapılar ders kitabından alınmıştır (@author Koffman and Wolfgang).

Search Tree Interfacesinde bulunan metodlar Override edilmelidir

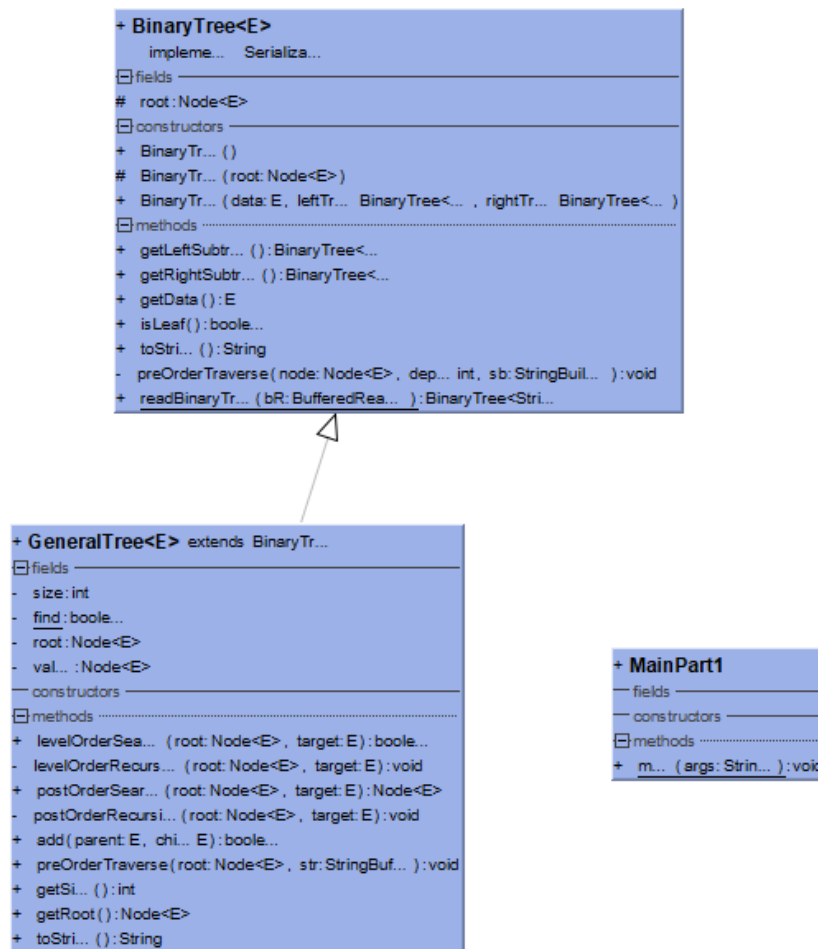
- boolean add(E item) : Tree yapısına yeni eleman ekler
- boolean contains(E target) : Tree yapısında verilen elemanın olup olmadığını sorgular
- E find(E target) : Verilen elemanı tree yapısında arar
- E delete(E target) : Verilen elemanı tree üzerinden siler
- boolean remove(E target) : Verilen elemanı tree'den kaldırır

MultidimensionSearchTree'nin objesi olarak kullanılacak Multidimensional Classı ve bunun Comparable olması için compareTo() methodunun imlement edilmesi gerekmektedir.

2 METHOD

2.1 Class Diagrams

Part1: Bu bölümde GeneralTree Class'ı Binary Class'ını extends ederek kullanılmıştır.



Part2: Bu bölümde MultidimSearchTree Classı BinaryTree Class'ını extends ederek ve SearchTree Interfacesini implements ederek kullanmıştır. MultidimSearchTree Classı Multidimensional Class'ının objelerini kullanark yapıyı oluşturduğu için Multidimensional Class'ı da Comparable olmalıdır bu yüzden Comparable Interfacesini implement etmiştir.



2.2 Problem Solution Approach

Part1: Bu bölümde BinaryTree yapısı extends edilerek GeneralTree yapısını BinaryTree gibi kullanılması istenilmiştir. BinaryTree yapısı extends edileceği için bu Class SystemRequirementte de belirttiğim gibi ders kitabından alınmıştır. Bu bölümde bizden istenilen ise aşağıdaki methodları GeneralTree mantığında BinaryTree yi extends eden bir Class üzerinde gerçekleştirecek metodlar aşağıdaki gibidir.

- levelOrderSearch () ,verilen elemanı tree yapısında arayıp bulduğunda (true) bulamadığında (false) döndürür.
- postOrderSearch() ,verilen elemanı tree yapısında arayıp bulduğunda Node<E> tree yapısının bir birimi (elemanın tutulma biçimi) şeklinde bulamadığında boş (null) döndürür.
- preOrderTraverse() ,verilen elemandan başlayarak ağacı (tree) dolaşip elemanların verisini (data) bir string yapısına ekler.

Bu methodlar Recursive olarak çalışan helper methodlar kullanılarak çözülmüştür ve bu methodların parametreleri aşağıdaki gibi belirlenmiştir.

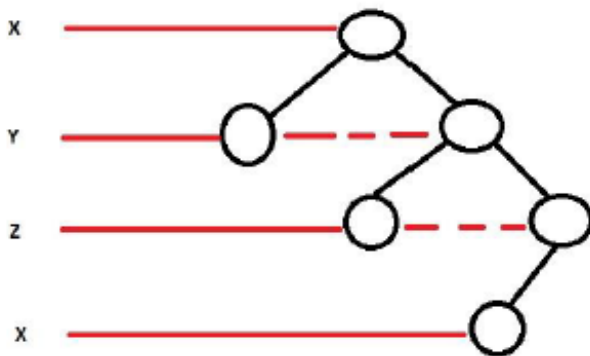
- public boolean levelOrderSearch (Node<E> root , E target)
- public Node<E> postOrderSearch(Node<E> root , E target)
- public void preOrderTraverse(Node<E> root,StringBuffer str)

Ödevde verilen bir kuralda preOrderTraverse() methodu Override edilmesi istenilmiştir fakat bu method BinaryTree Classında private tanımlandığı için bu işlemi gerçekleştirmek için Override edilmeyip preOrderTraverse() methodu normal bir method gibi tanımlanmıştır.

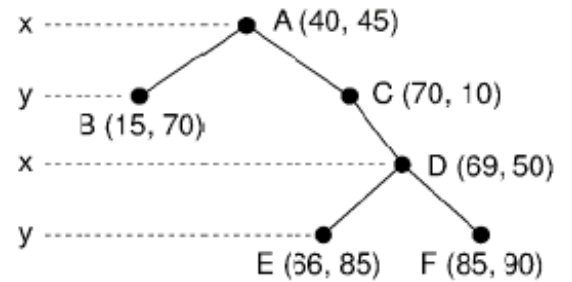
Part2: Bu bölümde Binary Tree Classı extends edilecek ve SearchTree Interface'si implements edileceği için bu yapılar ders kitabından alınmıştır.Bu yapılar kullanılacağı için Bu Tree yapısında kullanılan objenin Comparable olması gerekmektedir bu nedenle Multidimensional Classı Comparable Interfacesini implement etmiştir.

Aşağıda çözümümle alakalı gösterim ve açıklama verilmiştir.

BENİM YAPTIĞIM



ÖDEVDE VERİLEN



Yukarıda verilen örnekte sadece (x , y)eksenlerine göre tree ye dağıtıyor fakat son gönderdiğiniz Explanation2 ve benim çıkarımımla 3 boyutlu veriler ile çalışacağız ve tree ye daha iyi dağılması için yukarıda resimde de görüldüğü gibi (x , y , z) eksenlerine göre tree ye ekleme yapıldı ve methodlar bu sisteme uygun şekilde gerçekleştirildi

3 RESULT

3.1 Test Cases

Part1: JUnit Test for GeneralTree

- levelOrderSearch() ,verilen elemanı tree yapısında arayıp bulduğunda (true) bulamadığında (false) döndürür.
- postOrderSearch() ,verilen elemanı tree yapısında arayıp bulduğunda Node<E> tree yapısının bir birimi (elemanın tutulma biçimi) şeklinde bulamadığında boş (null) döndürür.
- preOrderTraverse() ,verilen elemandan başlayarak ağacı (tree) dolaşip elemanların verisini (data) bir string yapısına ekler.
- add() , yapıya eleman ekler

The screenshot displays the GeneralTreeTest.java file in an IDE. The file contains four JUnit tests for the GeneralTree class:

```
class GeneralTreeTest {
    GeneralTree<String> generalTree = new GeneralTree<>();

    @Test
    void levelOrderSearch() {
        generalTree.add( parent: "", child: "1" );
        generalTree.add( parent: "1", child: "2");
        generalTree.add( parent: "1", child: "3");
        assertEquals( expected: true, generalTree.levelOrderSearch(generalTree.getRoot(), target: "3"));
    }

    @Test
    void postOrderSearch() {
        generalTree.add( parent: "", child: "1" );
        generalTree.add( parent: "1", child: "2");
        generalTree.add( parent: "1", child: "3");
        assertEquals( expected: "2", generalTree.postOrderSearch(generalTree.getRoot(), target: "2").data);
    }

    @Test
    void add() {
        assertEquals( expected: true, generalTree.add( parent: "", child: "1" ));
    }

    @Test
    void preOrderTraverse() {
        generalTree.add( parent: "", child: "1" );
        generalTree.add( parent: "1", child: "2");
        generalTree.add( parent: "1", child: "3");
        StringBuffer str = new StringBuffer();
        generalTree.preOrderTraverse(generalTree.getRoot(), str);
        assertEquals( expected: "1 2 3 ", str.toString());
    }
}
```

The Run window at the bottom shows the test results for GeneralTreeTest:

Test Case	Duration	Status
GeneralTreeTest	21 ms	Passed
postOrderSearch()	17 ms	Passed
preOrderTraverse()	1 ms	Passed
levelOrderSearch()	2 ms	Passed
add()	1 ms	Passed

Tests passed: 4 of 4 tests - 21 ms

Process finished with exit code 0

Part2: JUnit Test for Multidimensional Class methods

- `public int compareToX(Multidimensional c)` : x eksenlerini karşılaştırır
- `public int compareToY(Multidimensional c)` : y eksenlerini karşılaştırır
- `public int compareToZ(Multidimensional c)` : z eksenlerini karşılaştırır
- `public int compareTo(Multidimensional o)` : verilen objlerin (0,0,0)'a uzaklığını karşılaştırır
- `public boolean equals(Object o)` : verilen obje ile kendi objemiz aynı mı değil mi

The screenshot displays an IDE with the `MultidimensionalTest.java` file open. The left sidebar shows the project structure, including the `Test` directory where the test file is located. The main editor shows the following code:

```
1 package Part2;
2
3 import ...
4
5
6
7 class MultidimensionalTest {
8     Multidimensional multi = new Multidimensional( x: 7, y: 2, z: 3);
9     @Test
10    void compareToX() {
11        assertEquals( expected: 3, multi.compareToX(new Multidimensional( x: 4, y: 3, z: 6)));
12    }
13
14    @Test
15    void compareToY() {
16        assertEquals( expected: -1, multi.compareToY(new Multidimensional( x: 4, y: 3, z: 6)));
17    }
18
19    @Test
20    void compareToZ() {
21        assertEquals( expected: -3, multi.compareToZ(new Multidimensional( x: 4, y: 3, z: 6)));
22    }
23
24    @Test
25    void compareTo() {
26        assertEquals( expected: 0, multi.compareTo(new Multidimensional( x: 4, y: 3, z: 6)));
27    }
28
29    @Test
30    void equals() {
31        assertEquals( expected: true, multi.equals(new Multidimensional( x: 7, y: 2, z: 3)));
32    }
33 }
```

The bottom panel shows the test results for `MultidimensionalTest`. All tests passed:

Test Method	Duration
MultidimensionalTest	23 ms
compareTo()	16 ms
equals()	4 ms
compareToX()	1 ms
compareToY()	1 ms
compareToZ()	1 ms

The console output shows: "C:\Program Files\Java\jdk1.8.0_144\bin\java.exe" ... Process finished with exit code 0.

JUnit Test for MultidimSearchTree Class methods

Aşağıdaki methodların testi yapıldı.

- boolean add(E item) : Tree yapısına yeni eleman ekler
- boolean contains(E target) : Tree yapısında verilen elemanın olup olmadığını sorgular
- E find(E target) : Verilen elemanı tree yapısında arar
- E delete(E target) : Verilen elemanı tree üzerinden siler
- boolean remove(E target) : Verilen elemanı tree'den kaldırır

The screenshot shows an IDE with the following components:

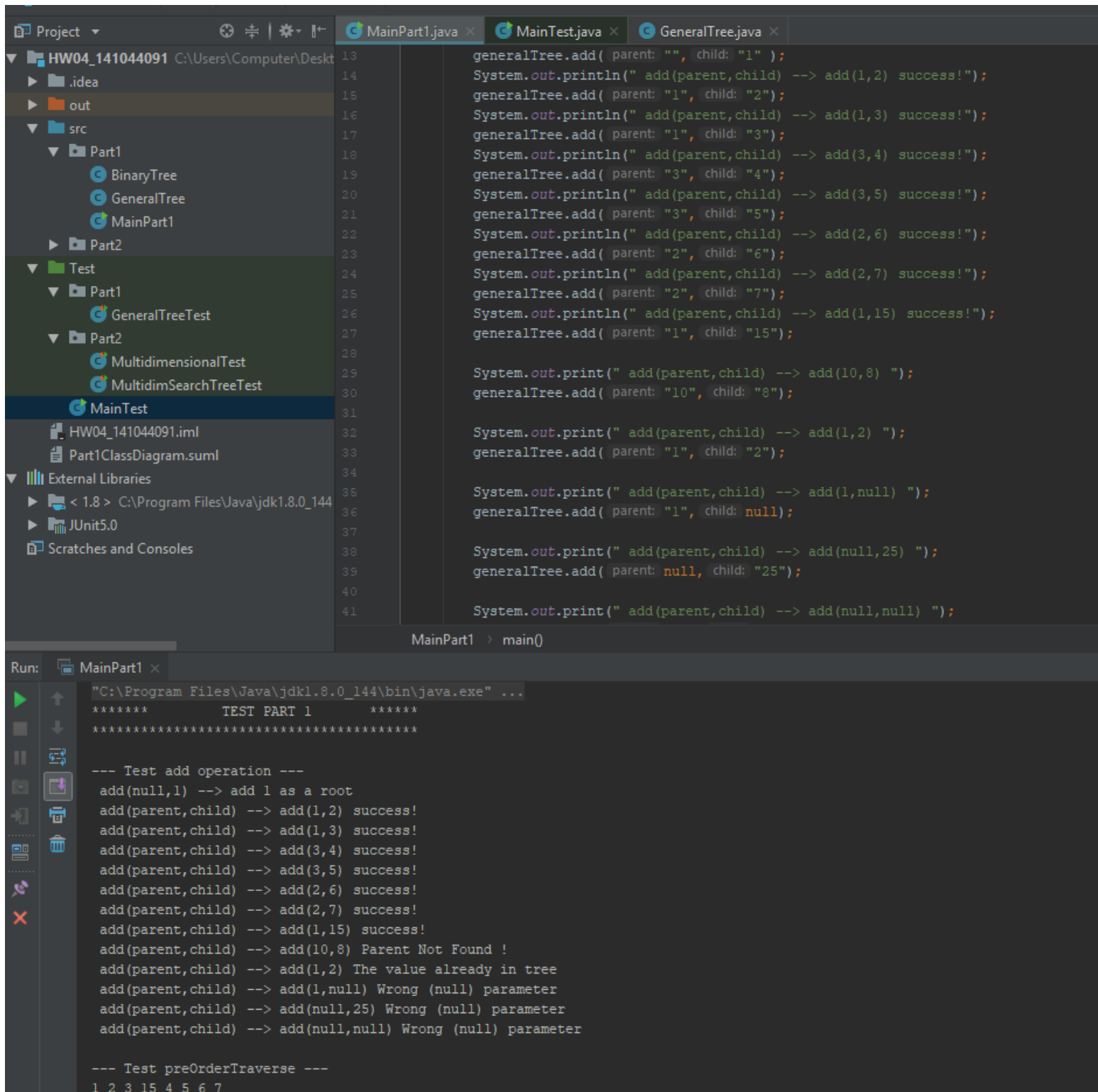
- Project Explorer:** Shows the project structure with folders like .idea, out, src, and Test. The Test folder contains MultidimSearchTreeTest.
- Code Editor:** Displays the MultidimSearchTreeTest.java file with the following code:

```
class MultidimSearchTreeTest {  
    MultidimSearchTree gst = new MultidimSearchTree();  
    @Test  
    void add() {  
        assertEquals( expected: true, gst.add(new Multidimensional( x: 1, y: 2, z: 0)));  
    }  
    @Test  
    void contains() {  
        gst.add(new Multidimensional( x: 1, y: 2, z: 0));  
        gst.add(new Multidimensional( x: 2, y: 3, z: 5));  
        gst.add(new Multidimensional( x: 1, y: 7, z: 9));  
        assertEquals( expected: true, gst.contains(new Multidimensional( x: 1, y: 7, z: 9)));  
    }  
    @Test  
    void find() {  
        gst.add(new Multidimensional( x: 1, y: 2, z: 0));  
        gst.add(new Multidimensional( x: 2, y: 3, z: 5));  
        gst.add(new Multidimensional( x: 1, y: 7, z: 9));  
        gst.add(new Multidimensional( x: 16, y: 85, z: 23));  
        assertEquals( expected: "(1,7,9)", gst.find(new Multidimensional( x: 1, y: 7, z: 9)).toString());  
    }  
    @Test  
    void delete() {  
        gst.add(new Multidimensional( x: 1, y: 2, z: 0));  
        gst.add(new Multidimensional( x: 2, y: 3, z: 5));  
        gst.add(new Multidimensional( x: 1, y: 7, z: 9));  
        gst.add(new Multidimensional( x: 16, y: 85, z: 23));  
        assertEquals( expected: "(1,7,9)", gst.delete(new Multidimensional( x: 1, y: 7, z: 9)).toString());  
    }  
    @Test  
    void remove() {  
        gst.add(new Multidimensional( x: 1, y: 2, z: 0));  
        gst.add(new Multidimensional( x: 2, y: 3, z: 5));  
        gst.add(new Multidimensional( x: 1, y: 7, z: 9));  
        gst.add(new Multidimensional( x: 16, y: 85, z: 23));  
        assertEquals( expected: true, gst.remove(new Multidimensional( x: 1, y: 7, z: 9)));  
    }  
}
```
- Run:** Shows the test results for MultidimSearchTreeTest. The tests are all passing.

Test	Time
MultidimSearchTreeTest	19 ms
delete()	17 ms
remove()	1 ms
contains()	1 ms
add()	
find()	

3.2 Running Results

Part1: Aşağıdaki resimde implement edilen GeneralTree Classını üzerinde objesini kullanarak istenilen methodların çalıştığı gösterilmiştir.



The screenshot displays an IDE with three open Java files: `MainPart1.java`, `MainTest.java`, and `GeneralTree.java`. The `GeneralTree.java` file contains the implementation of the `GeneralTree` class, which includes methods for adding nodes and traversing the tree. The `MainTest.java` file contains the test logic, which creates a `GeneralTree` object and performs various operations on it. The `MainPart1.java` file is also visible, showing the `main` method.

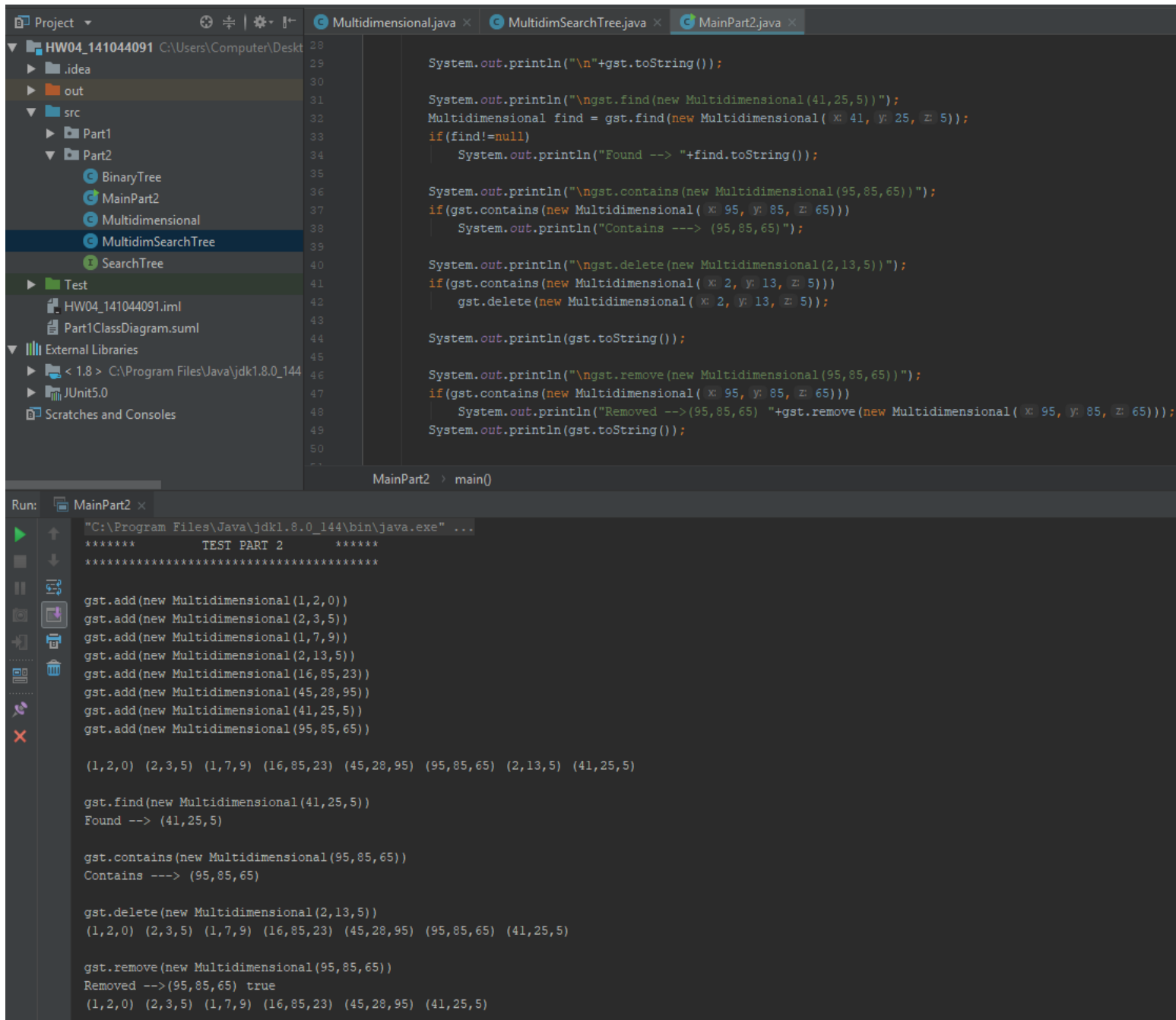
The Run window at the bottom shows the output of the program, which includes the following text:

```
***** TEST PART 1 *****
----- Test add operation -----
add(null,1) --> add 1 as a root
add(parent,child) --> add(1,2) success!
add(parent,child) --> add(1,3) success!
add(parent,child) --> add(3,4) success!
add(parent,child) --> add(3,5) success!
add(parent,child) --> add(2,6) success!
add(parent,child) --> add(2,7) success!
add(parent,child) --> add(1,15) success!
add(parent,child) --> add(10,8) Parent Not Found !
add(parent,child) --> add(1,2) The value already in tree
add(parent,child) --> add(1,null) Wrong (null) parameter
add(parent,child) --> add(null,25) Wrong (null) parameter
add(parent,child) --> add(null,null) Wrong (null) parameter

----- Test preOrderTraverse -----
1 2 3 15 4 5 6 7
```

Part2: Aşağıdaki resimde MultidimSearchTree Class'ının istenilen metodları kullanılarak test edilmiştir.

- boolean add(E item) : Tree yapısına yeni eleman ekler
- boolean contains(E target) : Tree yapısında verilen elemanın olup olmadığını sorgular
- E find(E target) : Verilen elemanı tree yapısında arar
- E delete(E target) : Verilen elemanı tree üzerinden siler
- boolean remove(E target) : Verilen elemanı tree'den kaldırır



4 ANALYSIS

Part1: Aşağıdaki resimde GeneralTree Analiz edilmiştir

```
1 package Part1;
2
3 public class GeneralTree<E> extends BinaryTree<E> {
4
5     private int size = 0;
6     private static boolean find=false;
7     private Node<E> root,value;
8     /**
9
10    public boolean levelOrderSearch (Node<E> root , E target){
11        if(root==null||target==null)
12            return false;
13        levelOrderRecursive(root,target);
14        boolean findVal = find;
15        find=false;
16        return findVal;
17    }
18
19    /**
20
21    private void levelOrderRecursive(Node<E> root , E target){
22
23        if(root.data==target){
24            find=true;
25            return;
26        }
27        if(root.left!=null)
28            levelOrderRecursive(root.left,target);
29        if(root.right!=null)
30            levelOrderRecursive(root.right,target);
31    }
32
33    /**
34
35    public Node<E> postOrderSearch(Node<E> root , E target){
36
37        postOrderRecursive(root,target);
38        if(!find)
39            return null;
40        find=false;
41        return this.value;
42    }
43
44    /**
45
46    private void postOrderRecursive(Node<E> root , E target){
47
48        if(root==null||target==null)
49            return;
50        if(root.data.equals(target)){
51            value=root;
52            find=true;
53            return;
54        }
55        if(root.left!=null)
56            postOrderRecursive(root.left,target);
57        if(root.right!=null)
58            postOrderRecursive(root.right,target);
59    }
60
61    /**
62
63    public boolean add(E parent, E child){
64
65        if(size==0) {
66            this.root = new Node<>(child);
67            size++;
68            return true;
69        }
70        if(parent==null || child ==null) {
71            System.out.println("Wrong (null) parameter ");
72            return false;
73        }
74        if(!levelOrderSearch(this.root,parent)){
75            System.out.println("Parent Not Found !");
76            return false;
77        }
78        if(levelOrderSearch(this.root,child)){
79            System.out.println("The value already in tree");
80            return false;
81        }
82
83        Node<E> parentNode = postOrderSearch(this.root,parent);
84        Node<E> childNode = new Node<>(child);
85
86        if(parentNode.right==null){
87            parentNode.right = childNode;
88            size++;
89            return true;
90        }
91        else {
92            Node<E> sibling = parentNode.right;
93            while (sibling.left!=null){
94                sibling= sibling.left;
95            }
96            sibling.left=childNode;
97        }
98        return false;
99    }
100
101    /**
102
103    public void preOrderTraverse(Node<E> root,StringBuffer str){
104        if(root==null)
105            return;
106        str.append(root.toString()+" ");
107        if(root.left!=null)
108            preOrderTraverse(root.left,str);
109        if(root.right!=null)
110            preOrderTraverse(root.right, str);
111    }
112
113    public int getSize(){return size;}
114
115    public Node<E> getRoot() {
116        return root;
117    }
118
119    @Override
120    public String toString(){
121        StringBuffer str = new StringBuffer();
122        preOrderTraverse(root,str);
123        return String.valueOf(str);
124    }
125
126 }
```

Complexity Analysis:

- levelOrderSearch:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- levelOrderRecursive:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- postOrderSearch:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- postOrderRecursive:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- add (Initial Node):** $O(1)$ Constant
- add (Search Parent):** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- add (Search Child):** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- add (Insertion):** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- add (Sibling Insertion):** $O(1)$ Constant
- Total:** $O(\log(n))$
- preOrderTraverse:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$
- getSize:** $O(1)$ Constant
- getRoot:** $O(1)$ Constant
- toString:** $T(n) = T(n/2) + T(n/2)$, $O(\log(n))$

Part2: Aşağıdaki resimde Multidimensional Class'ı Analiz edilmiştir

[illegible]

Aşağıdaki resimlerde ise MultidimSearchTree Class'ı Analiz edilmiştir.

```
3
4 import java.util.ArrayList;
5 import java.util.List;
6
7 public class MultidimSearchTree extends BinaryTree implements SearchTree<Multidimensional> {
8
9
10 private Node<Multidimensional> root = null;
11 private Node<Multidimensional> parent = null;
12 private boolean finder = false;
13 private boolean addFlag = false;
14 /**
15  *
16  * @Override
17  * public boolean add(Multidimensional item) {
18  *     if(item==null)
19  *         return false;
20  *     if(root==null){
21  *         root = new Node<>(item);
22  *         parent = root;
23  *         return true;
24  *     }
25  *     addHelper(root, item,0);
26  *     addFlag=false;
27  *     return true;
28  * }
29
30 /**
31  * private void addHelper(Node<Multidimensional> root,Multidimensional item,int depth){
32  *
33  *     if(addFlag)
34  *         return;
35  *     if(depth%3==0){ // Compare x axis
36  *
37  *         if(root.data.compareToX(item)>0){
38  *             if(root.right==null){
39  *                 root.right = new Node<>(item);
40  *                 addFlag=true;
41  *                 return;
42  *             }
43  *             else
44  *                 addHelper(root.right,item,++depth);
45  *         }
46  *         if(root.data.compareToX(item)<=0){
47  *             if(root.left==null){
48  *                 root.left= new Node<>(item);
49  *                 addFlag=true;
50  *                 return;
51  *             }
52  *             else
53  *                 addHelper(root.left,item,++depth);
54  *         }
55  *     }
56  *     if(depth%3==1){ //compare y axis
57  *
58  *         if(root.data.compareToY(item)>0){
59  *             if(root.right==null){
60  *                 root.right = new Node<>(item);
61  *                 addFlag=true;
62  *                 return;
63  *             }
64  *             else
65  *                 addHelper(root.right,item,++depth);
66  *         }
67  *         if(root.data.compareToY(item)<=0){
68  *             if(root.left==null){
69  *                 root.left = new Node<>(item);
70  *                 addFlag=true;
71  *                 return;
72  *             }
73  *             else
74  *                 addHelper(root.left,item,++depth);
75  *         }
76  *     }
77  *     if(depth%3==2){ //compare z axis
78  *
79  *         if(root.data.compareToZ(item)>0){
80  *             if(root.right==null){
81  *                 root.right = new Node<>(item);
82  *                 addFlag=true;
83  *                 return;
84  *             }
85  *             else
86  *                 addHelper(root.right,item,++depth);
87  *         }
88  *         if(root.data.compareToZ(item)<=0){
89  *             if(root.left==null){
90  *                 root.left = new Node<>(item);
91  *                 addFlag=true;
92  *                 return;
93  *             }
94  *             else
95  *                 addHelper(root.left,item,++depth);
96  *         }
97  *     }
98  * }
99
100 /**
101  * @Override
102  * public boolean contains(Multidimensional target) {
103  *     finder=false;
104  *     if(find(target)!=null)
105  *         return true;
106  *     return false;
107  * }
108
109 /**
110  * @Override
111  * public Multidimensional find(Multidimensional target) {
112  *     if(target==null)
113  *         return null;
114  *     Node<Multidimensional> finding = null;
115  *     findHelper(root,target,0, finding);
116  *
117  *     if(finder)
118  *         return target;
119  *     finder = false;
120  *     return null;
121  * }
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
```

Complexity Analysis:

- add(Multidimensional item):** $O(1)$ Constant for the initial check, $O(\log(n))$ for the recursive calls. **Total: $O(\log(n))$**
- contains(Multidimensional target):** $O(\log(n))$
- find(Multidimensional target):** $O(\log(n))$

Recurrence Relation:

$$T(n) = T(n/6) + T(n/6) + T(n/6) + T(n/6) + T(n/6) + T(n/6)$$

Time Complexity: $T(n) = \log(n)$

Total: $O(\log(n))$

```

138  /**
145  private void findHelper(Node<Multidimensional> root, Multidimensional item, int depth, Node<Multidimensional> find) {
146  if (finder)
147  return;
148  if (depth%3==0) { // Compare x axis
149
150      if (root.data.compareToX(item)>0){
151
152          if (root.data.equals(item)){
153              parent=find;
154              finder=true;
155              return;
156          }
157          else
158              findHelper(root.right,item,++depth,root);
159      }
160      if (root.data.compareToX(item)<=0){
161
162          if (root.data.equals(item)){
163              parent=find;
164              finder=true;
165              return;
166          }
167          else
168              findHelper(root.left,item,++depth,root);
169      }
170  }
171
172  if (depth%3==1) { //compare y axis
173
174      if (root.data.compareToY(item)>0){
175
176          if (root.data.equals(item)){
177              parent=find;
178              finder=true;
179              return;
180          }
181          else
182              findHelper(root.right,item,++depth,root);
183      }
184      if (root.data.compareToY(item)<=0){
185          if (root.data.equals(item)){
186              parent=find;
187              finder=true;
188              return;
189          }
190          else
191              findHelper(root.left,item,++depth,root);
192      }
193  }
194
195  if (depth%3==2) { //compare z axis
196
197      if (root.data.compareToZ(item)>0){
198          if (root.data.equals(item)){
199              parent=find;
200              finder=true;
201              return;
202          }
203          else
204              findHelper(root.right,item,++depth,root);
205      }
206      if (root.data.compareToZ(item)<=0){
207          if (root.data.equals(item)){
208              parent=find;
209              finder=true;
210              return;
211          }
212          else
213              findHelper(root.left,item,++depth,root);
214      }
215  }
216  }
217  }
218  /**
219  @Override
220  public Multidimensional delete(Multidimensional target) {
221
222      if (target==null)
223          return null;
224      finder=false;
225
226      Node<Multidimensional> child = null;
227      Node<Multidimensional> watch = null;
228      findHelper(this.root,target,0, watch);
229
230      if (!finder)
231          return null;
232      finder = false;
233
234      if (parent.left!=null)
235          if (parent.left.data.equals(target))
236              child=parent.left;
237      if (parent.right!=null)
238          if (parent.right.data.equals(target))
239              child=parent.right;
240
241      List<Multidimensional> data = new ArrayList<>();
242      preOrderTraverse(child,data);
243
244      if (parent.left!=null)
245          if (parent.left.data.equals(target))
246              parent.left = null;
247      if (parent.right!=null)
248          if (parent.right.data.equals(target))
249              parent.right = null;
250
251      for (int i=1; i<data.size(); i++){
252          add(data.get(i));
253      }
254      return target;
255  }
256  /**
257  private void preOrderTraverse(Node<Multidimensional> root, List<Multidimensional> list) {
258  if (root!=null){
259      list.add(root.data);
260      preOrderTraverse(root.left, list);
261      preOrderTraverse(root.right, list);
262  }
263  }
264  /**
265  @Override
266  public boolean remove(Multidimensional target) {
267  if (delete(target)!=null)
268      return true;
269  return false;
270  }
271  /**
272  @Override
273  public String toString(){
274  String str = "";
275  List<Multidimensional> list = new ArrayList<>();
276  preOrderTraverse(root,list);
277  for (int i=0; i<list.size(); i++)
278      str += list.get(i).toString()+" ";
279  return str;
280  }

```

$$T(n) = T(n/6) + T(n/6) + T(n/6) + T(n/6) + T(n/6) + T(n/6)$$

$$T(n) = \log(n)$$

$$\text{Total : } O(\log(n))$$

$$O(\log(n))$$

Total

$$O(n \cdot \log(n))$$

$$O(\log(n))$$

$$T(n) = n \cdot \log(n)$$

$$O(n \cdot \log(n))$$

$$T(n) = T(n/2) + T(n/2)$$

$$T(n) = \log(n)$$

$$O(\log(n))$$

$$O(\log(n))$$

$$O(\log(n))$$

$$O(n)$$

$$\text{Total : } O(n)$$