

**Gebze Technical University
Computer Engineering**

CSE 222 - 2018 Spring

HOMEWORK 4 REPORT

STUDENT NAME: YAĞMUR KARAMAN

STUDENT NUMBER: 141044016

Course Assistant: Burak Koca

1 INTRODUCTION

1.1 Problem Definition

PART 1:

General Tree'lerin Binary Tree olarak implement edilmesi gerekiyordu. Kitapta general tree ile ilgili bir örnek vardı ve ona bakarak implement edilmesi istendi. General Tree'deki bir node'un ilk çocuğu onun soluna, varsa diğer çocukları da ilk çocuğun sağına koyulmalı.

Bunun için BinaryTree class'ı implement edildi, methodlar ve kodlar kitaptan alındı.

Implement edilmesi gereken methodlar: add(E parent, E child), levelOrderSearch(), postOrderSearch(), preOrderTraverse().

PART 2:

General Search Tree yapısı implement edilmesi gerekiyordu. Tree'deki node'lar multidimensional itemlardan oluşmalıydı.

BinarySearch classı implement edilmeliydi, bu class SearchTree interface classını implement etmeliydi. Kendi yazdığım Part2_MultidimensionalTree classım da BinarySearch classını extend etmeliydi.

Kendi yazdığım classta add(...) methodunu implement ettim.

1.2 System Requirements

PART 1:

add(E parent, E child): Parent'a child'ı ekler.

levelOrderSearch(E parent, Node<E> root): Gelen parent item'ı ağaç üzerinde level order olarak arar.

postOrderSearch(E parent, Node<E> root): Gelen parent item'ı ağaç üzerinde post order olarak arar.

preOrderTraverse(Node<E> root): Ağaç üzerinde pre order gezer ve node'ların datalarını print eder.

PART 2:

addHelper(ArrayList<Integer> node): Helper method olarak kullanılır.

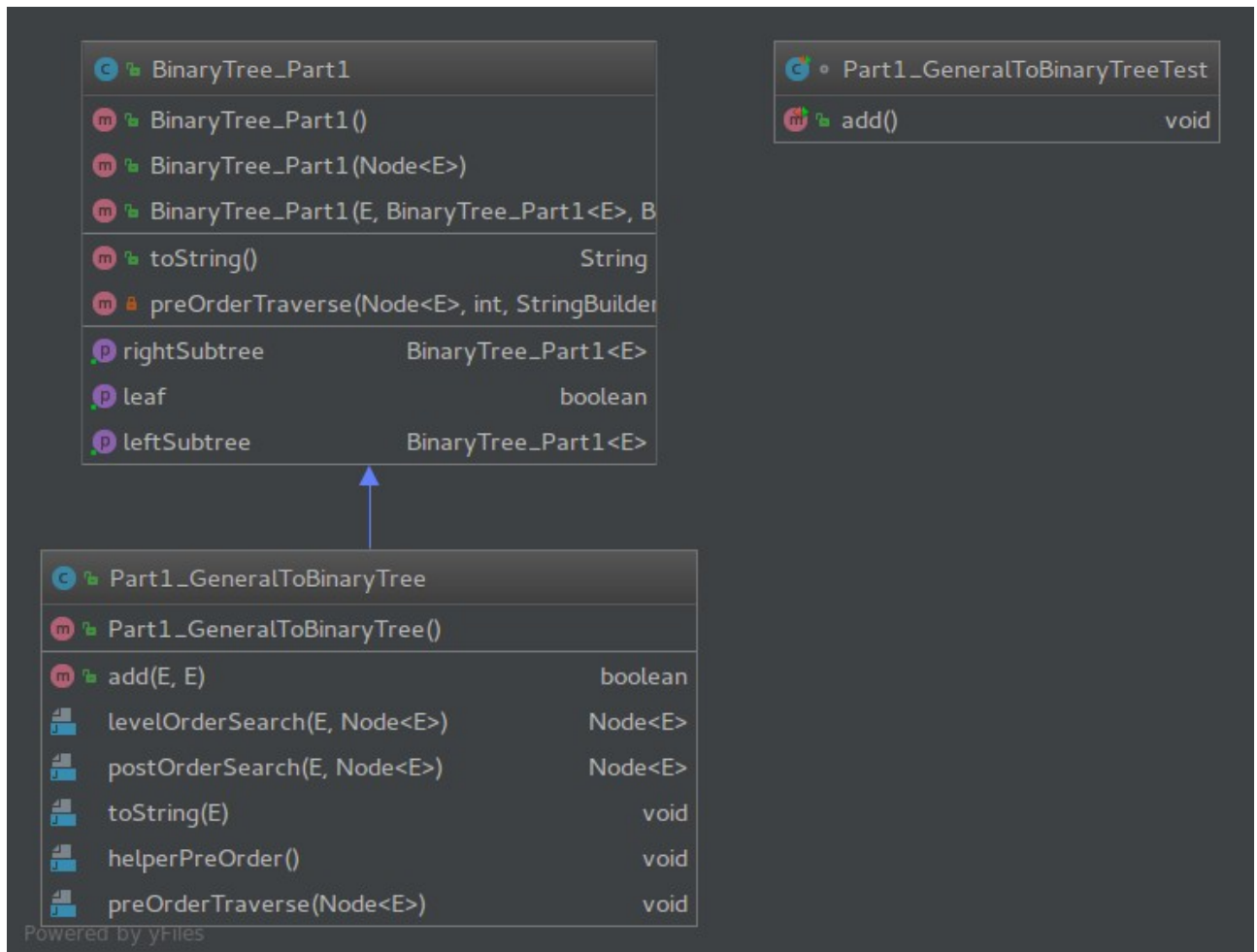
İçerisinde add methodunu çağırır.

add(Node root, ArrayList<Integer> node, ArrayList<Integer> dimen, int index): Gelen node'u ağacın uygun yerine ekler.

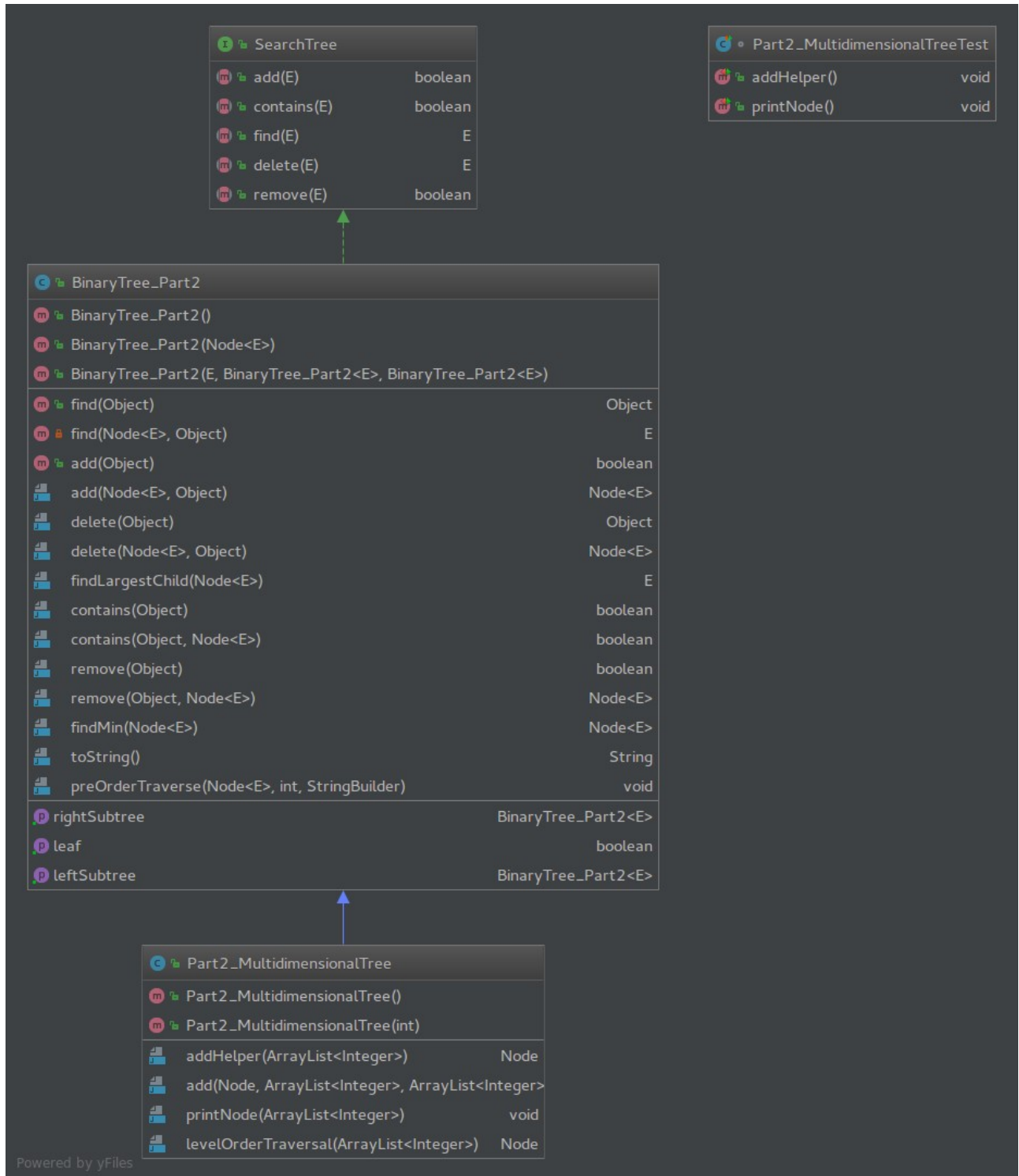
2 METHOD

2.1 Class Diagrams

PART 1:



PART 2:



2.2 Problem Solution Approach

PART 1: Part1 için `BinaryTree_Part1` classı implement edildi. `Part1_GeneralToBinaryTree.java` isimli classımda da `BinaryTree_Part1` classını extend ettim. Ödev için implement etmem gereken methodlar `add()`, `levelOrderSearch()`, `postOrderSearch()` ve `preOrderTraverse()` methodlarıydı.

add(E parentItem, E childItem): Root null ise, gelen parent'ı root'a ekler ve child'ı da root'un soluna ekler. Eğer root null değilse, parent node'u ağaçta arar, bulursa solunu kontrol eder, solu boşsa child'ı oraya ekler, boş değilse child'ın sağ'ı null olana kadar gider ve null olduğu zaman ekler. Ekleme başarılı şekilde olursa true return eder. Eğer parent item ağaçta yoksa ve ağaç boş değilse false return eder. **(O(n))**

levelOrderSearch(E parentItem, Node<E> root): Gelen parentItem'ı ağaçta level-order olarak arar. Bulursa bulduğu node'u return eder. Bulamazsa null return eder. **(O(n))**

postOrderSearch(E parentItem, Node<E> root): Gelen parentItem'ı ağaçta post-order olarak arar. Bulursa bulduğu node'u return eder. Bulamazsa null return eder. **(O(n))**

preOrderTraverse(Node<E> root): Ağacı pre-order olarak gezer ve node'ların datasını print eder. **(O(n))**

PART 2: Part2 için BinaryTree_Part2 classı ve SearchTree interface classı oluşturuldu. BinaryTree_Part2 classında SearchTree classı implement edildi. Part2_MultidimensionalTree.java classında da BinaryTree_Part2 classı extend edildi. Bu part için implement edilmesi gereken method temel olarak add methoduydu. Add methodu için addHelper isimli yardımcı method implement edildi.

addHelper(ArrayList<Integer> node): add methodu için helper method olarak implement edildi. İçerisinde oluşturduğu parametrelerle add methodunu çağırdı.

add(Node root, ArrayList<Integer> node, ArrayList<Integer> dimen, int index): index parametresi flag olarak kullanıldı. Dimen arraylisti ağaçta kaç'ıncı levelda(x, y, z...) olduğunu kontrol etmek ve ona göre gelen node'un kaç'ıncı elemanına bakılacağını anlamak için kullanıldı. Böylece gelen node ağaçta uygun yere eklendi.

3 RESULT

3.1 Test Cases

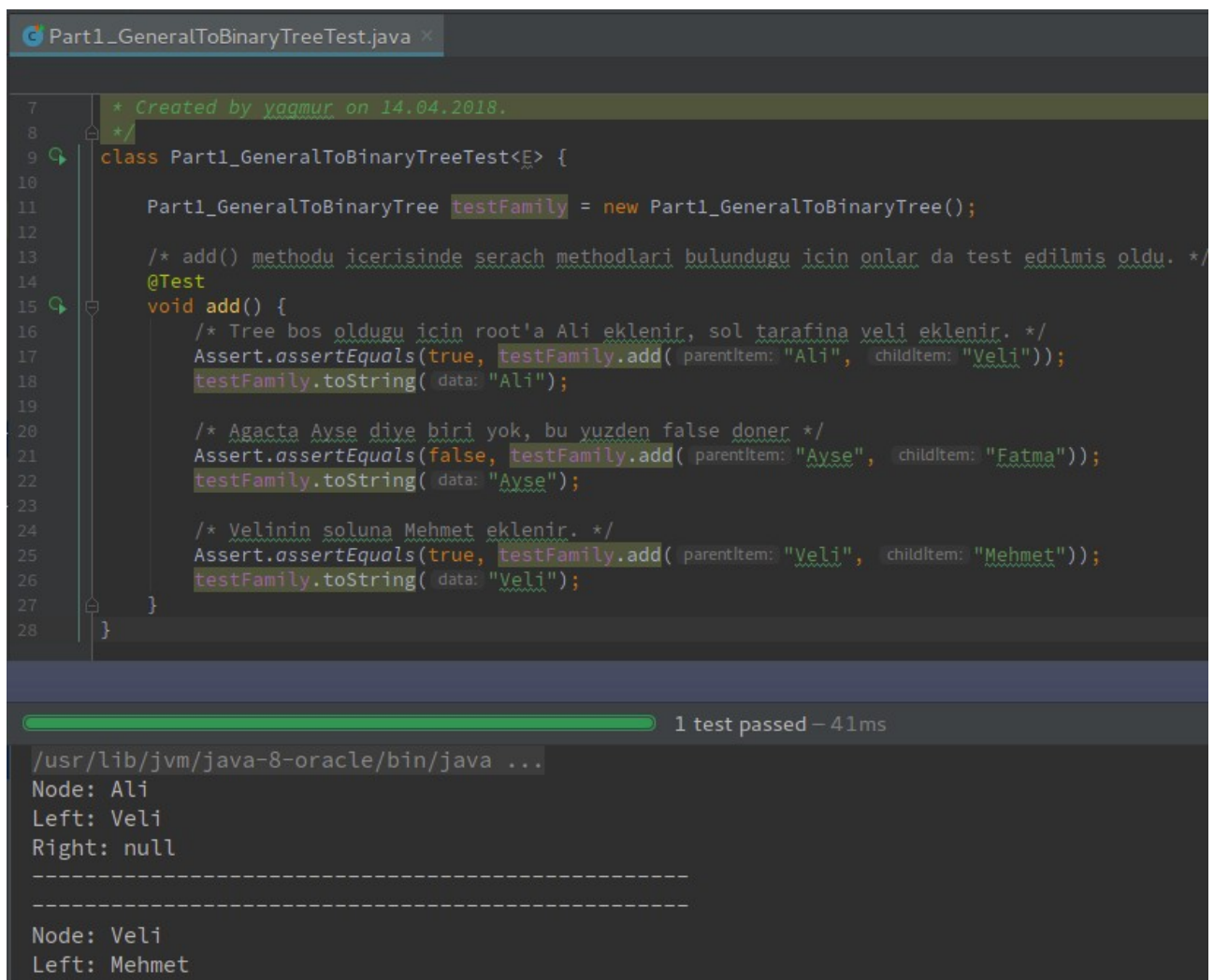
PART 1:

-add methodu test edildi. Önce boş olan ağaçtaki Ali parentına, Veli childı eklendi. Ağaç boş olduğu için root'a Ali, sol tarafına da Veli atandı. toString methodu ile Ali node'u, solu ve sağı print edildiğinde de sol tarafına Veli'nin eklendiği görüldü.

-Daha sonra bu ağaca Ayse parentına Fatma childı eklenmeye çalışıldı. Fakat ağaçta Ayse isimli bir node olmadığı için ve ağaç da boş olmadığı için false return etmesi beklendi. Ayse isimli node olmadığı için toString methodu da herhangi bir şey print edemedi.

-Son olarak da Veli parentına Mehmet childı eklenmeye çalışıldı. Ağaçta Veli isimli bir node bulunmaktaydı, bu node bulundu, sol tarafı boş olduğu için de sol tarafına Mehmet eklendi. toString methodu ile de print edildi.

-add methodu içinde search methodları kullanıldığı için, böylece onlar da test edilmiş oldu.



```
Part1_GeneralToBinaryTreeTest.java x
7  * Created by yagmur on 14.04.2018.
8  */
9  class Part1_GeneralToBinaryTreeTest<E> {
10
11      Part1_GeneralToBinaryTree testFamily = new Part1_GeneralToBinaryTree();
12
13      /* add() methodu içerisinde search methodlari bulunduğu için onlar da test edilmiş oldu. */
14      @Test
15      void add() {
16          /* Tree bos olduğu için root'a Ali eklenir, sol tarafına veli eklenir. */
17          Assert.assertEquals(true, testFamily.add( parentItem: "Ali", childItem: "Veli"));
18          testFamily.toString( data: "Ali");
19
20          /* Ağaçta Ayse diye biri yok, bu yüzden false döner */
21          Assert.assertEquals(false, testFamily.add( parentItem: "Ayse", childItem: "Fatma"));
22          testFamily.toString( data: "Ayse");
23
24          /* Velinin soluna Mehmet eklenir. */
25          Assert.assertEquals(true, testFamily.add( parentItem: "Veli", childItem: "Mehmet"));
26          testFamily.toString( data: "Veli");
27      }
28  }
```

1 test passed – 41ms

```
/usr/lib/jvm/java-8-oracle/bin/java ...
Node: Ali
Left: Veli
Right: null
-----
Node: Veli
Left: Mehmet
```

PART 2:

-add methodunu test etmek için addHelper methodu test edildi, çünkü kodlama sırasında add methodu için böyle bir helper methodu kullanıldı. addHelper methodu içerisinde add methodu çağırıldığı için bu şekilde test edilmiş oldu.

-Test için 2 boyutlu node kullanıldı. Bunun için boyut bilgisi alan constructor çağırıldı.

-[20,30] node'u ağaç boş olduğu için root'a eklendi.

-Daha sonra [25,13] node'u eklenmeye çalışıldı. Eklenirken x koordinatına bakıldı. Root'ta bulunan node'un x değeri 20, yeni eklenen node'taki değer ise 25. Kıyasladığımızda $25 > 20$ olduğu için yeni node root'un sağına eklenmeliydi.

-Son olarak [12,45] node'u eklenmeye çalışıldı. Yine x değerine baktığımızda $12 < 20$ olduğu için root'un sol tarafına eklenmeliydi.

-Altta da printNode methodunu test ettiğimizde, doğru print edildiğini gördüm.

```
Part2_MultidimensionalTreeTest.java x
Part2_MultidimensionalTreeTest addHelper()
12 class Part2_MultidimensionalTreeTest {
13
14     /* 2 boyutlu */
15     Part2_MultidimensionalTree test = new Part2_MultidimensionalTree(2);
16     ArrayList node1 = new ArrayList();
17     ArrayList node2 = new ArrayList();
18     ArrayList node3 = new ArrayList();
19
20     /* addHelper methodu içerisinde add methodunu çağırıldığı için o da test edilmiş ol
21     @Test
22     void addHelper() {
23         node1.add(20);
24         node1.add(30);
25         test.addHelper(node1);
26         Assert.assertEquals(test.root.data, node1);
27
28         node2.add(25);
29         node2.add(13);
30         test.addHelper(node2);
31         Assert.assertEquals(test.root.right.data, node2);
32
33         node3.add(12);
34         node3.add(45);
    }
}

All 2 tests passed – 25ms

/usr/lib/jvm/java-8-oracle/bin/java ...
Found: [20, 30]
Left: [12, 45]
Right: [25, 13]
-----
Process finished with exit code 0
```

3.2 Running Results

```
Main
/usr/lib/jvm/java-8-oracle/bin/java ...
*****
***** PART 1 *****
*****
##### TREE 1 - ENGLAND FAMILY #####
Tree is empty. Add -> Parent-William 1 and Child-Robert:
Node: William 1
Left: Robert
Right: null
-----
Parent-William 1 and Child-William 2:
Node: Robert
Left: null
Right: William 2
-----
Parent-William 1 and Child-Adela:
Node: William 2
Left: null
Right: Adela
-----
Parent-William 1 and Child-Henry 1:
Node: Adela
Left: null
Right: Henry 1
-----
Parent-Robert and Child-William:
Node: Robert
Left: William
Right: William 2
-----
```



```
Main
-----
Parent-Adela and Child-Stephen:
Node: Adela
Left: Stephen
Right: Henry 1
-----
Parent-Henry 1 and Child-William:
Node: Henry 1
Left: William
Right: null
-----
Parent-Henry 1 and Child-Matilda:
Node: William
Left: null
Right: null
-----
Parent-Matilda and Child-Henry 2:
Node: Matilda
Left: Henry 2
Right: null
-----
Parent-Henry 2 and Child-Henry:
Node: Henry 2
Left: Henry
Right: null
-----
Parent-Henry 2 and Child-Richard 1:
Node: Henry
Left: null
Right: Richard 1
-----
```



Parent-Henry 2 and Child-Geoffrey:

Node: Richard 1

Left: null

Right: Geoffrey

Parent-Henry 2 and Child-John:

Node: Geoffrey

Left: null

Right: John

Parent-John and Child-Henry 3:

Node: John

Left: Henry 3

Right: null

Parent-John and Child-Richard:

Node: Henry 3

Left: null

Right: Richard

Parent-Henry 3 and Child-Edward 1:

Node: Henry 3

Left: Edward 1

Right: Richard

Parent-Henry 3 and Child-Edmund:

Node: Edward 1

Left: null

Right: Edmund

```
Main
-----
Parent-Edward 1 and Child-Edward 2:
Node: Edward 1
Left: Edward 2
Right: Edmund
-----
Parent-Edward 1 and Child-Thomas:
Node: Edward 2
Left: null
Right: Thomas
-----
Parent-Edward 1 and Child-Edmund:
Node: Thomas
Left: null
Right: Edmund
-----
Parent-Edward 2 and Child-Edward 3:
Node: Edward 2
Left: Edward 3
Right: Thomas
-----
My PreOrderTraverse:
William 1 -> Robert -> William -> William 2 -> Adela -> Stephen -> Henr
```

Main

↑

↓

↕

📁

📄

🗑️

BinaryTreeClass PreOrderTraverse:
William 1
Robert
William
null
null
William 2
null
Adela
Stephen
null
null
Henry 1
William
null
Matilda
Henry 2
Henry
null
Richard 1
null
Geoffrey
null
John
Henry 3
Edward 1
Edward 2
Edward 3
null
null
Thomas
null

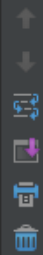
All files are up-to-date (2 minutes ago)

```
Main
##### TREE 2 #####
Tree is empty. Add -> Parent-Ahmet and Child-Ecrin:
Node: Ahmet
Left: Ecrin
Right: null
-----
Parent-Ahmet and Child-Mehmet:
Node: Ecrin
Left: null
Right: Mehmet
-----
Parent-Mehmet and Child-Ali:
Node: Mehmet
Left: Ali
Right: null
-----
Parent-Mehmet and Child-Veli:
Node: Ali
Left: null
Right: Veli
-----
Parent-Ecrin and Child-Esila:
Node: Ecrin
Left: Esila
Right: Mehmet
-----
Parent-Ahmet and Child-Ayşe:
Node: Mehmet
Left: Ali
Right: Ayşe
-----
```

```
Main
-----
Parent-Ecrin and Child-Yagmur:
Node: Esila
Left: null
Right: Yagmur
-----
Parent-Yagmur and Child-Erdi:
Node: Yagmur
Left: Erdi
Right: null
-----
Parent-Ahmet and Child-Filiz:
Node: Ayse
Left: null
Right: Filiz
-----
Parent-Ayse and Child-Fatma:
Node: Ayse
Left: Fatma
Right: Filiz
-----
Parent-Filiz and Child-Yıldız:
Node: Filiz
Left: Yıldız
Right: null
-----
My PreOrderTraverse:
Ahmet -> Ecrin -> Esila -> Yagmur -> Erdi -> Mehmet -> Ali -> Veli -> Ays
```

```
Main
Ahmet -> Ecrin -> Esila -> Yagmur -> Erdi -> Mehmet -> Ali -> Veli -> Ayse ->

BinaryTreeClass PreOrderTraverse:
Ahmet
  Ecrin
    Esila
      null
    Yagmur
      Erdi
        null
        null
        null
      Mehmet
        Ali
          null
          Veli
            null
            null
        Ayse
          Fatma
            null
            null
          Filiz
            Yıldız
              null
              null
              null
          null
```



```
*****  
***** PART 2 *****  
*****
```

```
##### TREE 1: 2 DIMENSION #####
```

```
Found: [40, 45]  
Left: null  
Right: null
```

```
-----  
Found: [40, 45]  
Left: [15, 70]  
Right: null
```

```
-----  
Found: [40, 45]  
Left: [15, 70]  
Right: [70, 10]
```

```
-----  
Found: [70, 10]  
Left: null  
Right: [69, 50]
```

```
-----  
Found: [69, 50]  
Left: [66, 85]  
Right: null
```

```
-----  
Found: [69, 50]  
Left: [66, 85]  
Right: [85, 90]  
-----
```




TREE 2: 3 DIMENSION

Found: [4, 5, 10]

Left: null

Right: null

Found: [4, 5, 10]

Left: [2, 20, 40]

Right: null

Found: [4, 5, 10]

Left: [2, 20, 40]

Right: [10, 15, 8]

Found: [10, 15, 8]

Left: [20, 13, 7]

Right: null

Found: [10, 15, 8]

Left: [20, 13, 7]

Right: [8, 18, 30]

Found: [8, 18, 30]

Left: null

Right: [15, 28, 40]

Process finished with exit code 0