

**Gebze Technical University  
Computer Engineering**

**CSE 222 - 2018 Spring**

**HOMEWORK 4 REPORT**

**VEFİK FIRAT AKMAN  
151044031**

# 1 INTRODUCTION

## 1.1 Problem Definition

### Part1:

**General Tree**'nin **Binary Tree** olarak temsil edilmesi istenmektedir.

Ayrıca bu temsil işlemini gerçekleştirecek program da kullanılacak sınıfın **Binary Tree** sınıfından **Extend** edilmesi istenmektedir.

### Part2:

**Yapamadım.**

## 1.2 System Requirements

Proje **Intelij IDE** üzerinde yazılmıştır. **Java 8** kullanılmıştır. **Ubuntu** ve **Windows 10** da çalıştırılabilir.

**Add()** fonksiyonu veriyi tutan yeni bir **node**'a (child) ve verinin **Tree** de yerleştirileceği konum için **Tree** için de bulunan **node**'a (**parent**) ihtiyacı vardır.

**postOrderSearch()** ve **levelOrderSearch()** fonksiyonları arama metotları olup **Tree** de ki **node**'lar da arayacakları veriye (**target**) ihtiyaç duymaktadır.

**preOrderTraverse()** fonksiyonu **preOrder()** fonksiyonunu çağırılmaktadır. Ve **preOrder()** fonksiyonu **Tree**'nin köküne ve **StringBuilder**'a ihtiyacı vardır.

# 2 METHOD

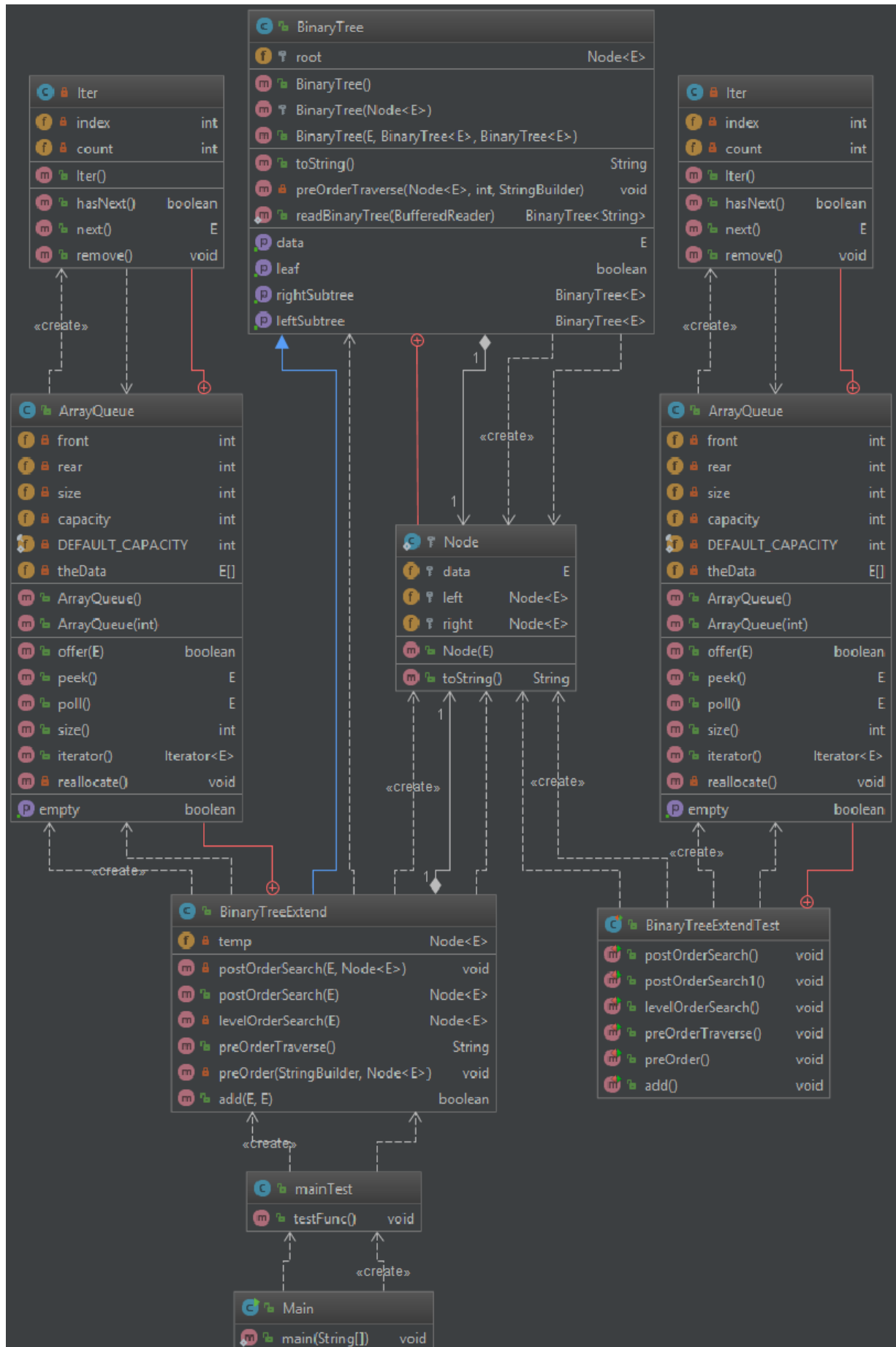
**Add()** fonksiyonu veriyi **Tree**'ye yeni bir **node** ekler. Yeni **node** **child**'tır. Ve **Child**'ın konumunu **parent node**'una göre belirler. Eğer ağaç boşsa **parent null** girilerek **child** olarak verilen **node** **root** olur.

**postOrderSearch()** fonksiyonu arama metodudur. Ve arama yaparken önce **child**'lara sonra **root**'a bakmaktadır. **Root**'un parametre olarak verildiği diğer **postOrderSearch()**'i çağırır.

**levelOrderSearch()** fonksiyonu arama metodudur. **Tree**'yi oluşturan **level**'lere göre ve ilk **level**'den son level'e göre arama yapmaktadır.

**preOrderTraverse()** fonksiyonu **preOrder()** fonksiyonunu çağırılmaktadır. Ve **preOrder()** fonksiyonu **Tree**'nin köküne ve **StringBuilder**'a ihtiyacı vardır.

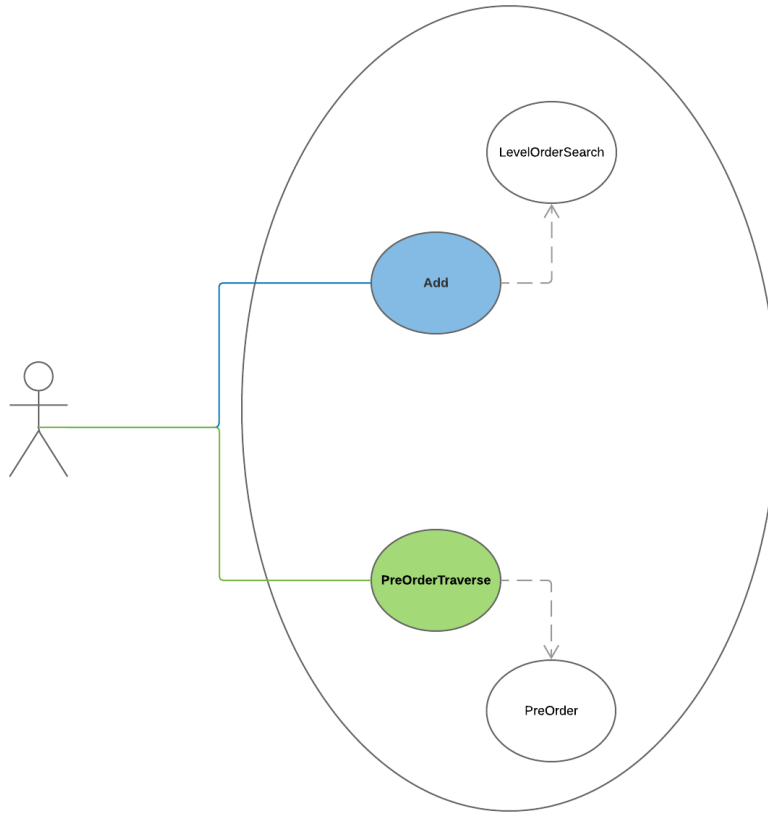
## 2.1 Class Diagrams



## 2.2 Use Case Diagrams

### BASIC USE CASE DIAGRAM

vfakman | April 16, 2018



## 2.3 Other Diagrams (optional)

*(Add other diagrams if required.)*

Başka bir diyagram gerekli gör(e)medim.

## 2.4 Problem Solution Approach

### Part1:

İlk olarak **General Tree** ve **Binary Tree** nedir? Ne özellikleri vardır? Birbirlerinden farkları nelerdir?

Araştırmalarını yaptım. Ardından problem için gerekli olan **Binary Tree** sınıfı ile **ArrayQueue** Sınıfını dersin kitabından aldım.

İstenilen metotların normal de **Binary Tree** de ve **General Tree** de olup olmadıklarını kontrol ettim. Bulduğum metot **implementation**'lerini inceledim. Ve kendi problemime uygun olanları kullandım veya modifiye ettim.

### Part2:

Part2 de istenileni tam olarak anlayamadım. Ve anlayabildiğim kısımlara çözüm üretemedim. Part1 de yaşadığım sorunlardan sonra da ödevin Part2'sini yapamayacağımı anlayıp yapmama kararı aldım.

## 3 RESULT

### 3.1 Test Cases

#### Main Test:

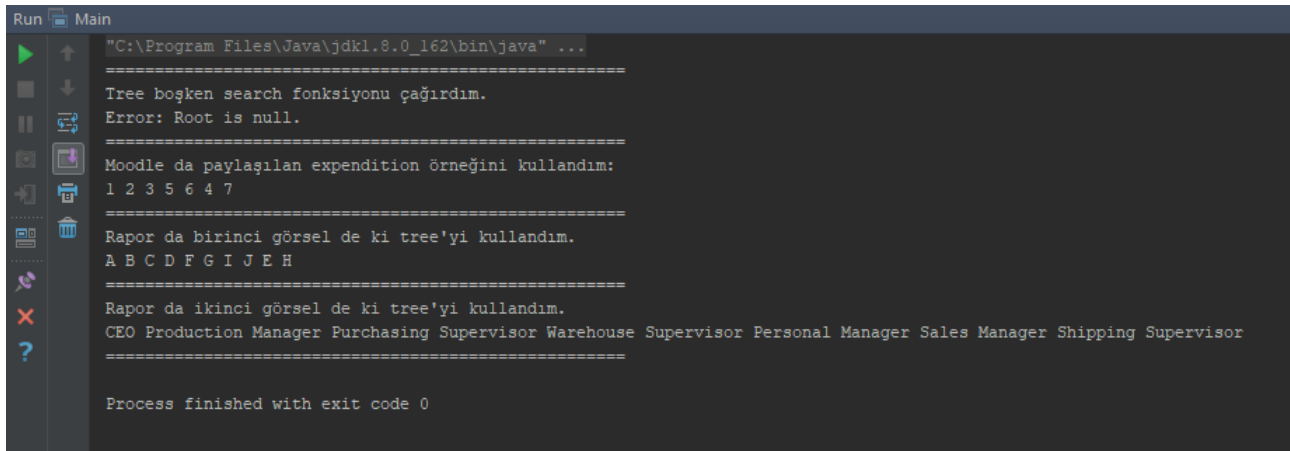
Main test de düzgün bir **output** formatıyla test **output** ekranında açıklanmıştır. Part1 test edilmektedir. **Output** ekranında her aşama kalın olarak ayrılmıştır.

#### Unit Test:

**BinaryTreeExtend** Sınıfının tüm metotları test edilmiştir.

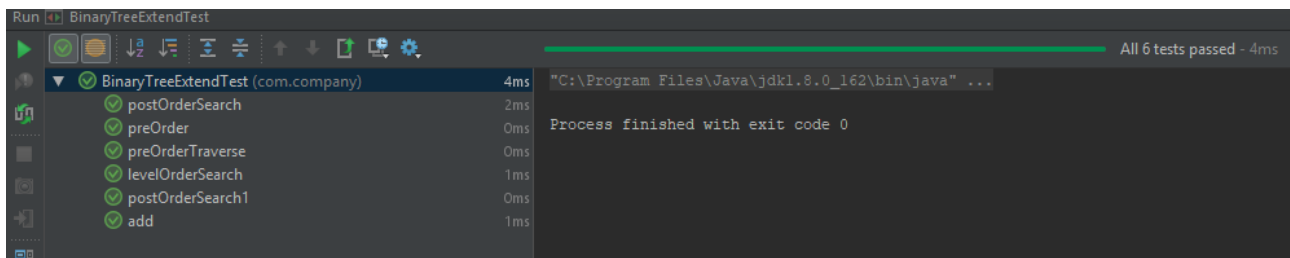
## 3.2 Running Results

### Main Test Running Result:



```
Run Main
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
=====
Tree boşken search fonksiyonu çağırdım.
Error: Root is null.
=====
Moodle da paylaşılan expenditure örneğini kullandım:
1 2 3 5 6 4 7
=====
Rapor da birinci görsel de ki tree'yi kullandım.
A B C D F G I J E H
=====
Rapor da ikinci görsel de ki tree'yi kullandım.
CEO Production Manager Purchasing Supervisor Warehouse Supervisor Personal Manager Sales Manager Shipping Supervisor
=====
Process finished with exit code 0
```

### Unit Test Running Result:



```
Run BinaryTreeExtendTest
All 6 tests passed - 4ms
BinaryTreeExtendTest (com.company) 4ms
  postOrderSearch 2ms
  preOrder 0ms
  preOrderTraverse 0ms
  levelOrderSearch 1ms
  postOrderSearch1 0ms
  add 1ms
"C:\Program Files\Java\jdk1.8.0_162\bin\java" ...
Process finished with exit code 0
```

## 4 Time Analyses

### preOrder():

$$T(f()) = O(n);$$

### preOrderSearch():

$$T_{\text{best}}(f()) = O(1) \text{ and } T_{\text{worst}}(f()) = T(\text{preOrder});$$

### postOrder():

$$T(f()) = O(n);$$

**postOrderSearch():**

$T_{\text{best}}(f()) = O(1)$  and  $T_{\text{worst}}(f()) = T(\text{postOrder})$ ;

**levelOrderSearch():**

$T_{\text{best}}(f()) = O(1)$  and  $T_{\text{worst}}(f()) = O(n)$ ;

**add():**

$T_{\text{best}}(f()) = O(1)$  and  $T_{\text{worst}} = T(\text{Search})$ ;

- Main titles -> 16pt , 2 line break
- Subtitles -> 14pt, 1.5 line break
- Paragraph -> 12pt, 1.5 line break