**Gebze Technical University**
**Computer Engineering**

# CSE 222 - 2018 Spring

# HOMEWORK 6 REPORT

**STUDENT NAME: YAĞMUR KARAMAN**
**STUDENT NUMBER: 141044016**

Course Assistant: Fatma Nur Esirci

# 1 Worst RedBlack Tree

## 1.1 Problem Solution Approach

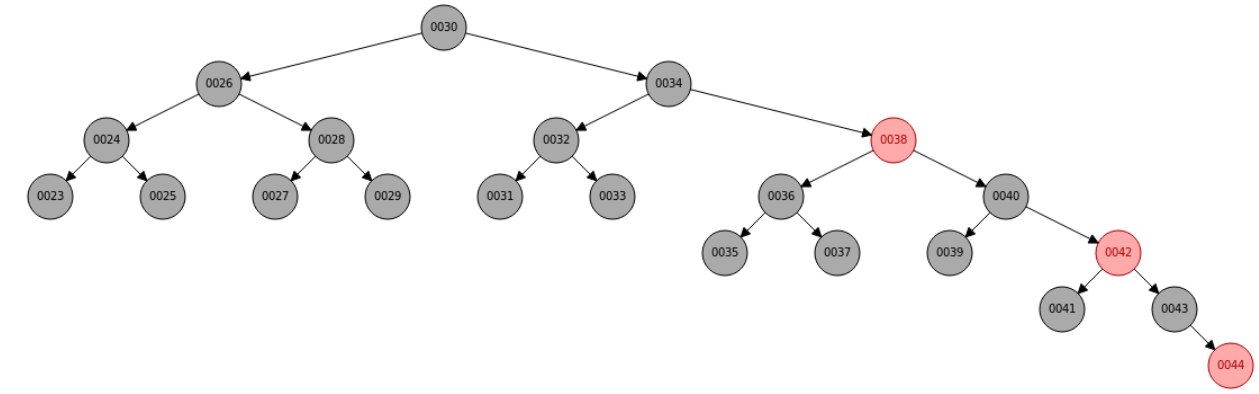
Benim RedBlack Tree'de worst case yaklaşımım, istenilen sayıdaki yüksekliğe en az sayıdaki node ile ulaşmaktı. Ödevde yükseklik olarak 6 belirlenmişti. Ben de bunu en az 22 node ile sağladım. Bu nodeların eklenme aşamalarının visualize edilmiş haline 1.3'e ekledim.

RedBlackTree classına createWorstRBTree() isimli yeni bir method etkledim.
**createWorstRBTree():** method oldukça basit bir method. Döngü içerisinde ağaca 22 tane eleman ekler. Öncelikle bir tane random sayı üretip sonra bu sayıyı birer birer artırarak ekleme yaptım. Böylece gelen elemanlar hep ağacın sağ tarafına eklenir, rebalance durum oluşursa da ağacı dengelemek için rotate işlemi uygulanır.

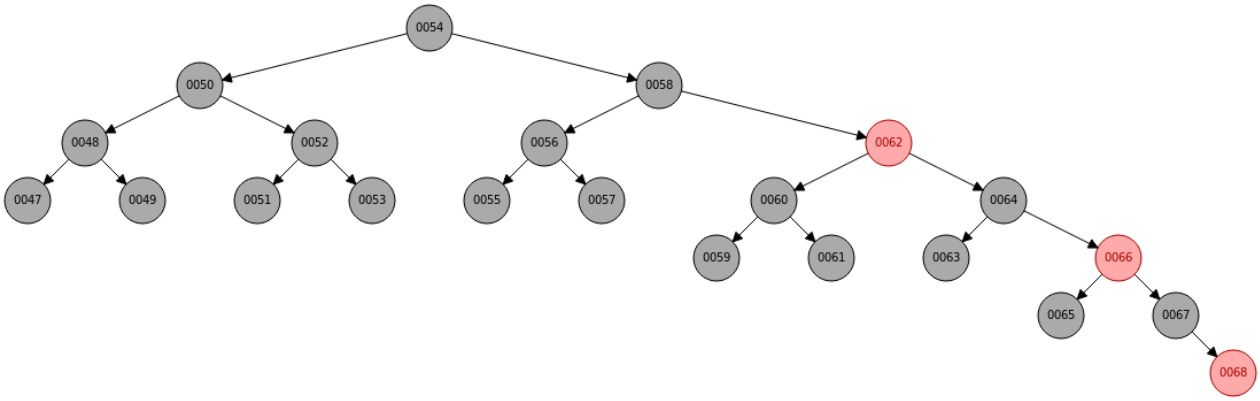
## 1.2 Test Cases

Main'de 2 tane ağaç objesi oluşturdum. Her biri için createWorstRBTree() methodunu çağırdım. Öncelikle her bir ağaç için 1-51 arasında random bir başlangıç sayısı generate etti. Daha sonra da 22 kere add(item) methodunu çağırdı ve her defasında item değerini 1 artırdı.
Test sonucu oluşan ağaçların görüntüsü aşağıdadır.

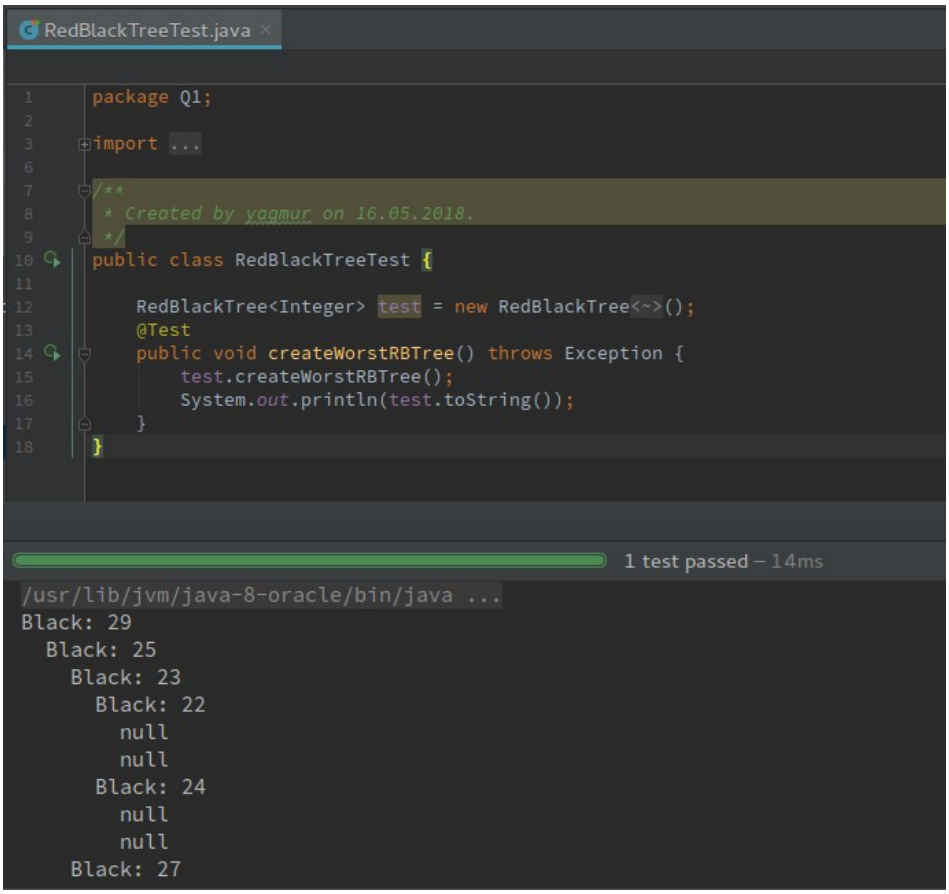
İlk ağaçta bulunan sayılar 23-44 arasındadır.




İkinci ağaçta bulunan sayılar 47-68 arasındadır.

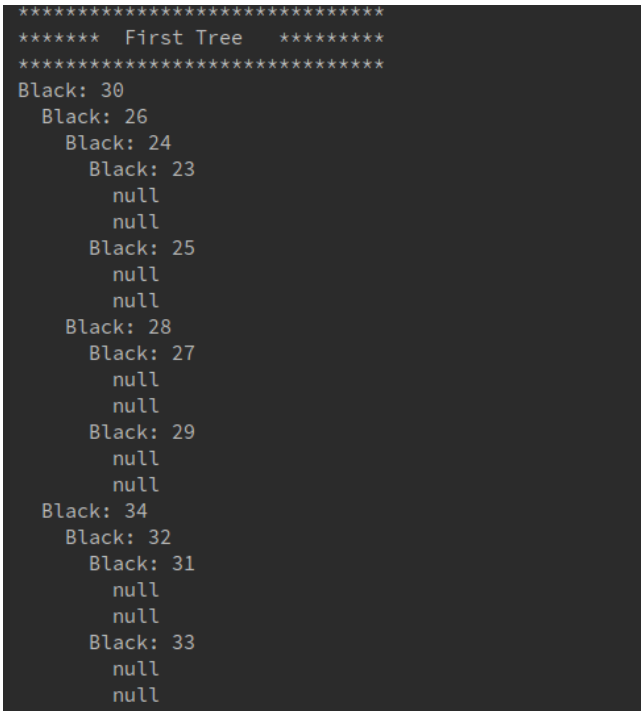


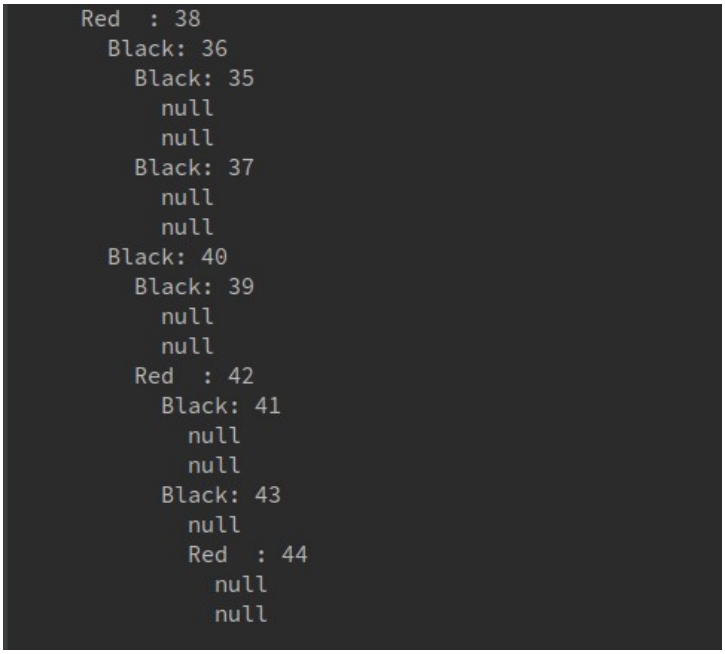
RedBlackTreeTest classı içinde de methodu tekrar test ettim.




## 1.3 Running Commands and Results

Yukarıda görselleştirilmiş ağaçların terminal çıktısı aşağıdadır.
İlk ağaç:



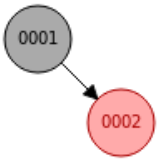
İkinci ağaç:

```
*****************************
*******  Second Tree   ********
*****************************
Black: 54
  Black: 50
    Black: 48
      Black: 47
        null
        null
      Black: 49
        null
        null
    Black: 52
      Black: 51
        null
        null
      Black: 53
        null
        null
  Black: 58
    Black: 56
      Black: 55
        null
        null
      Black: 57
        null
        null
```

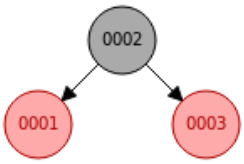
```
    Red  : 62
      Black: 60
        Black: 59
          null
          null
        Black: 61
          null
          null
      Black: 64
        Black: 63
          null
          null
        Red  : 66
          Black: 65
            null
            null
          Black: 67
            null
            Red  : 68
              null
              null
```

Örnek olarak 1-22 arasındaki sayılarda oluşan bir ağacın eleman ekleme aşamalarını, programın çalışma mantığının daha iyi anlaşılması için aşağıya ekledim. Öncelikle root'a 1 eklenir.

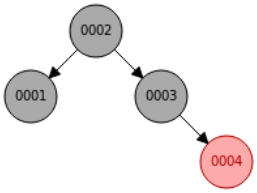
insert 2:



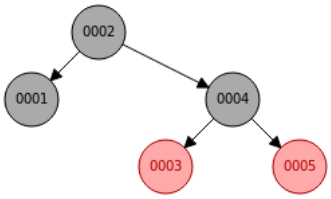

insert 3:



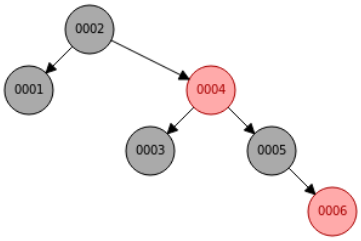

insert 4:



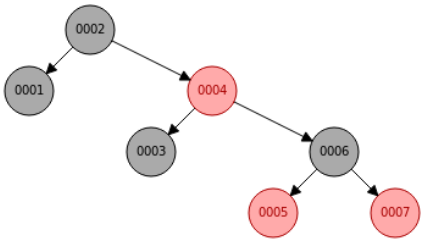

insert 5:



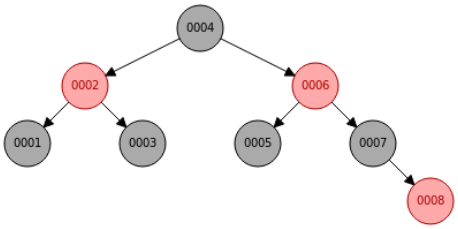

insert 6:



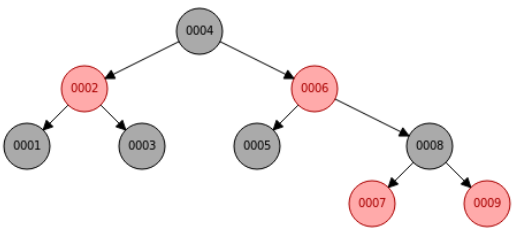

insert 7:



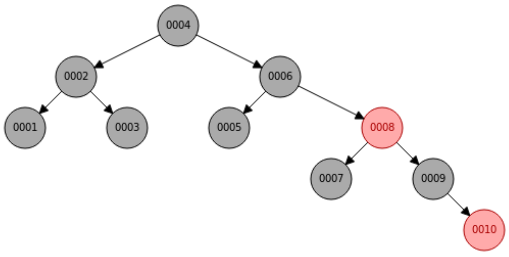

insert 8:



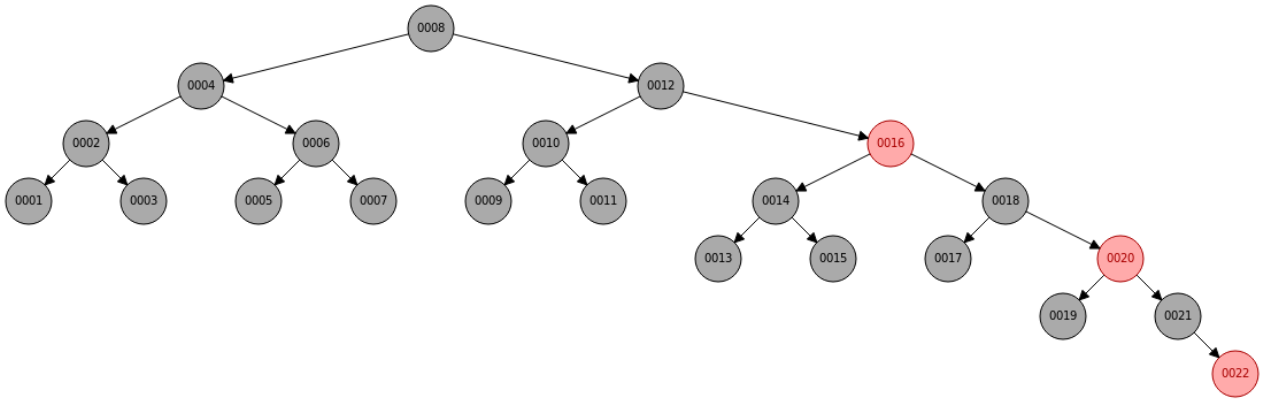

insert 9:




insert 10:



Diğer aşamalarda bu şekilde ilerler ve ağacın son hali aşağıdaki gibi olur.



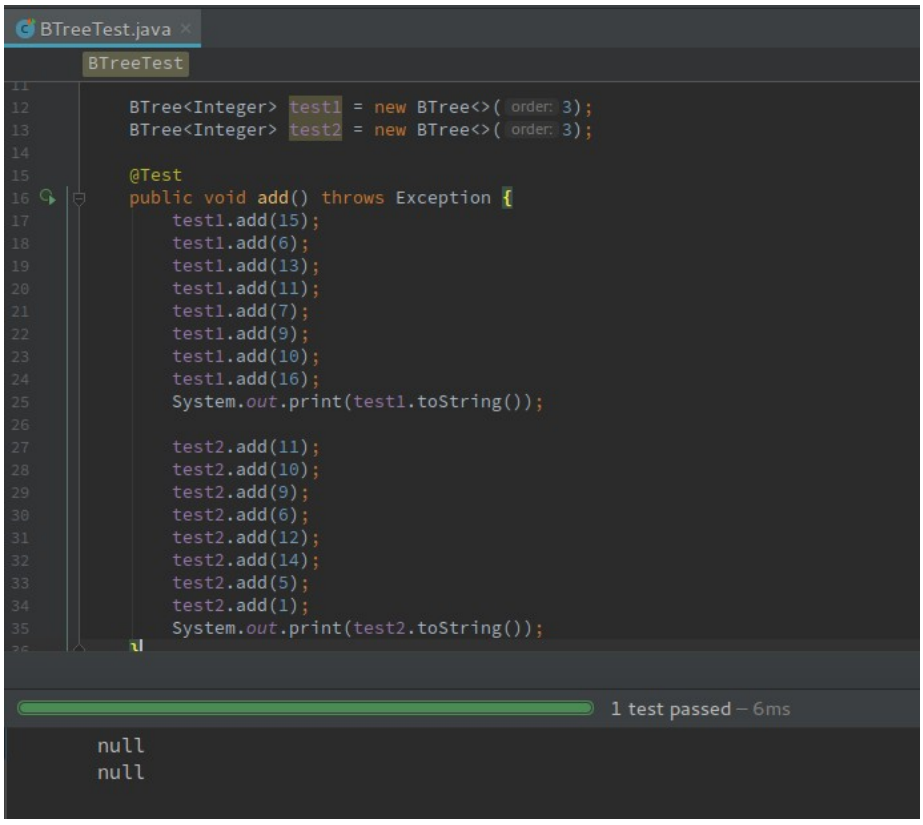
# 2 binarySearch method

## 2.1 Problem Solution Approach

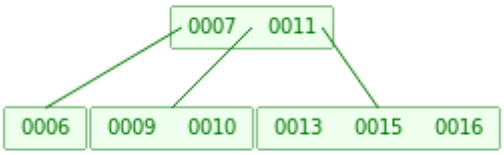
Btree classındaki binarySearch(...) methodunun implementinde, search edilecek elemanı, data arrayinde search ettim. Methodu iterative olarak implement ettim. Bunu yaparken 3 durumun kontrolünü yaptım:
1. Elemanın başta olması
2. Elemanın sonda olması
3. Elemanın arada bir yerde olması

## 2.2 Test Cases

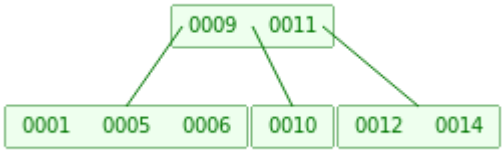
2 tane ağaç oluşturdum ve 8'er eleman ekledim.




test1 isimli ağacın visualize edilmiş hali:



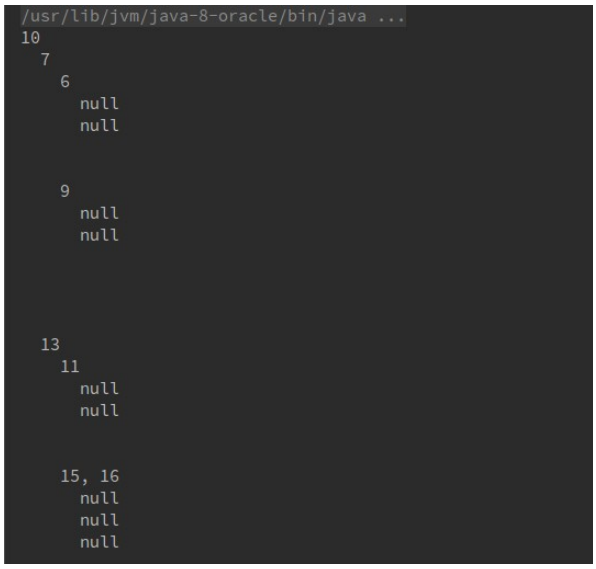

test2 isimli ağacın visualize edilmiş hali:



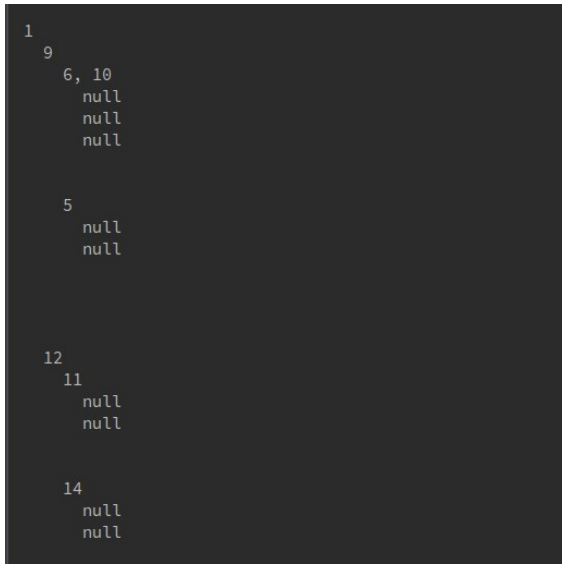
## 2.3 Running Commands and Results

test1 ve test2 isimli ağaçların beklenen görüntüsünü yukarıya eklemiştim. Şimdi de terminal çıktılarına bakalım.

test1 ağacının terminal çıktısı:



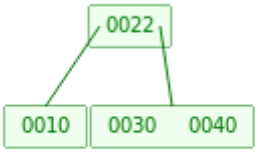

test2 ağacının terminal çıktısı:



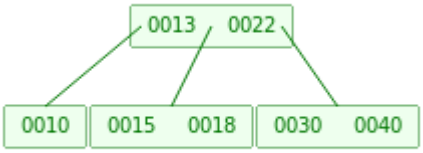

Mainde de aşağıdaki ağacı oluşturdum.

```
BTree<Integer> myBTree = new BTree<>( order: 3);
myBTree.add(10);
myBTree.add(22);
myBTree.add(30);
myBTree.add(40);
myBTree.add(13);
myBTree.add(15);
myBTree.add(18);
myBTree.add(20);
myBTree.add(5);
myBTree.add(7);
myBTree.add(8);
myBTree.add(26);
myBTree.add(27);
myBTree.add(32);
myBTree.add(35);
myBTree.add(38);
myBTree.add(42);
myBTree.add(46);

System.out.println(myBTree.toString());
```

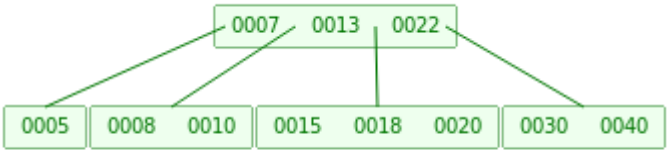
Mainde oluşturduğum ağacı visualize ettiğimde beklenen çıktı aşağıdaki gibidir.
10, 22 ve 30 yanyana eklendikten sonra 40 eklenmeye çalışılır. Fakat node dolu olduğu için aşağıdaki gibi split edilir.



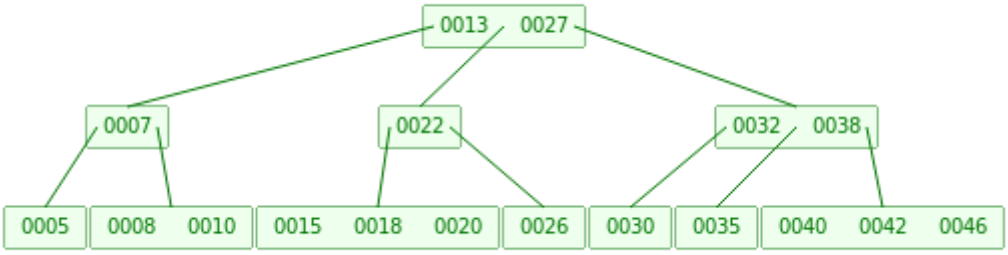

13, 22'nin soluna eklenir. 15, 13 ve 20'nin arasına eklenir. 18 de 15'in sağına eklenir.




20, 18'in sağına eklenir. 8, 10'un soluna eklenir. 5, 8'in soluna eklenir. 7 eklenmeye çalışıldığında 5 ve 8'in arasına gelmelidir. Fakat node dolu olduğu için 7 yukarıya 13'ün soluna gider, 5 de 7'nin soluna eklenir.




Diğer elemanların da eklenmesiyle ağacın son hali aşağıdaki gibi olur.



# 3 Project 9.5 in book

## 3.1 Problem Solution Approach

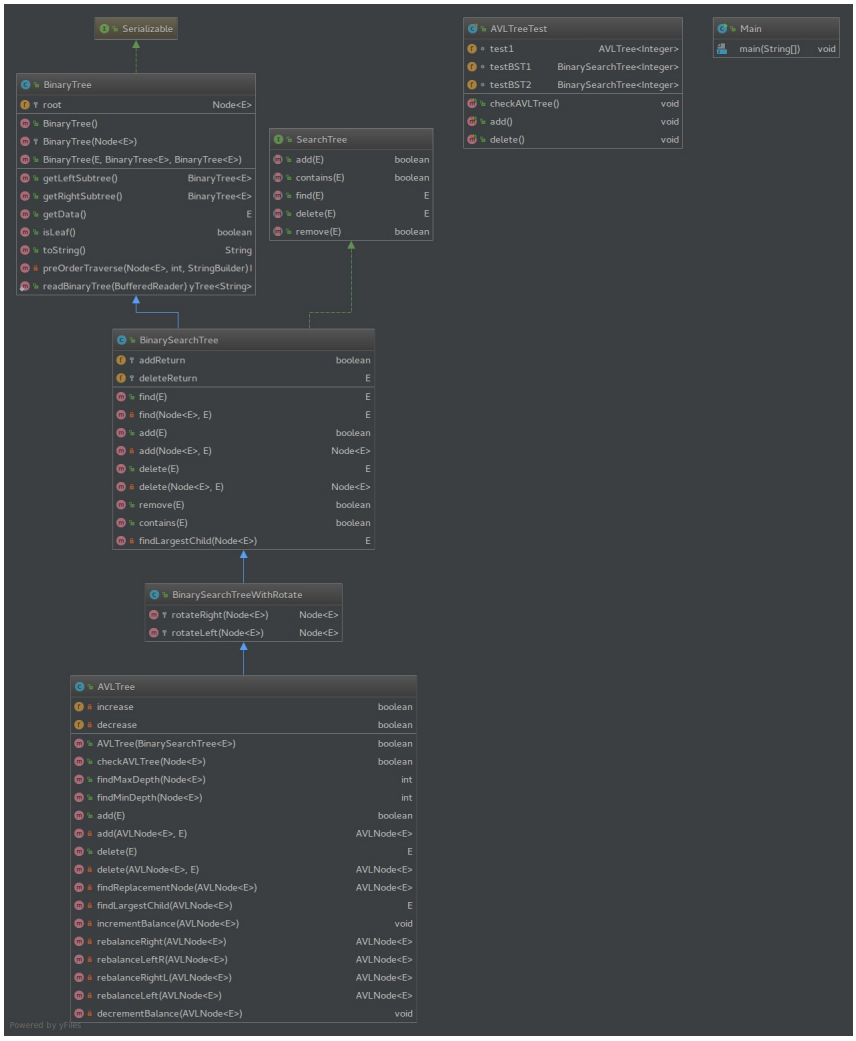
-Oluşturulan bir BST'nin AVLTree olup olmadığını anlamak için ağaçtaki tüm nodeları gezdim, her node'un left ve right ağaçlarının yüksekliğini buldum, ikisi arasındaki farkın -1, 0 veya 1 değerlerinden biri olup olmadığını kontrol ettim. Bunlardan biriyse checkAVLTree(root) methodu true, değilse false return eder.

***AVLTree(BinarySearchTree<E> bst):***
call checkAVLTree(root)

***checkAVLTree(root):***
call findMaxDepth(root) and calculate maximum depth of tree
call findMinDepth(root) and calculate minimum depth of tree
if these differences is less or equal than 1, return true.
else return false.

***findMaxDepth(root):***
calculate depth of right and left tree, return maximum of them

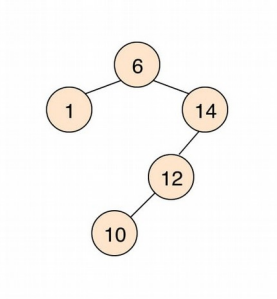
***findMinDepth(root):***
calculate depth of right and left tree, return minimum of them



## 3.2 Test Cases

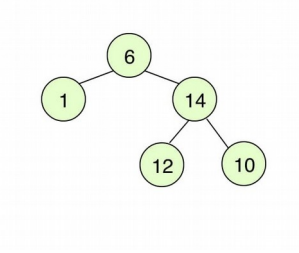
```java
public class AVLTreeTest {

    AVLTree<Integer> test1 = new AVLTree<Integer>();
    BinarySearchTree<Integer> testBST1 = new BinarySearchTree<Integer>();
    BinarySearchTree<Integer> testBST2 = new BinarySearchTree<Integer>();

    @Test
    public void checkAVLTree() throws Exception {
        testBST1.add(6);
        testBST1.add(14);
        testBST1.add(12);
        testBST1.add(1);
        testBST1.add(10);
        Assert.assertEquals( expected: false, test1.checkAVLTree(testBST1.root));

        testBST2.add(6);
        testBST2.add(14);
        testBST2.add(12);
        testBST2.add(1);
        testBST2.add(17);
        Assert.assertEquals( expected: true, test1.checkAVLTree(testBST2.root));
    }
```

Üstteki Test Casede 2 tane Binary Search Tree oluşturup AVL Tree olup olmadığını kontrol ettim. İlk testte testBST1 isimli ağacı oluşturdum, 5 tane eleman ekledim. Eklediğim sayılar ile oluşan BST:




Bu ağaçta 14 numaralı nodeun balance değeri -2 olduğu için AVL Tree değildir. Bu yüzden checkAVLTree() methodu çalıştırıldığında **false** return etmesi beklenmektedir.

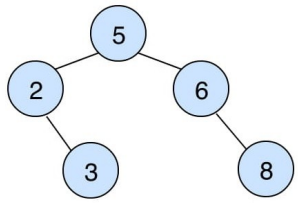
Son olarak da testBST2 isimli ağacı oluşturdum, yine 5 tane eleman ekledim. Oluşan BST:




Bu ağaçta 6 numaralı nodeun balance değeri 1, diğer nodeların balance değeri 0 olduğu için AVL Tree ağacının özelliklerini gösterir. Bu yüzden checkAVLTree() methodu çalıştırıldığında **true** return etmesi beklenir.

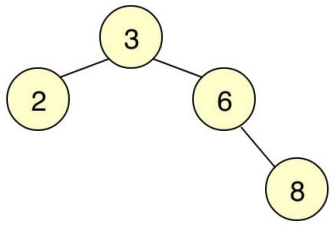
```java
    @Test
    public void add() throws Exception {
        test1.add(6);
        test1.add(5);
        test1.add(2);
        test1.add(3);
        test1.add(8);
        System.out.print(test1.toString());
    }
    @Test
    public void delete() throws Exception {
        test1.add(6);
        test1.add(5);
        test1.add(2);
        test1.add(3);
        test1.add(8);
        test1.delete( item: 5);
        System.out.print(test1.toString());
    }
}
```

Üstteki Test Casede add() methodunda test1 isimli AVLTree'ye 5 tane eleman eklendi. Oluşması beklenen AVL Tree:



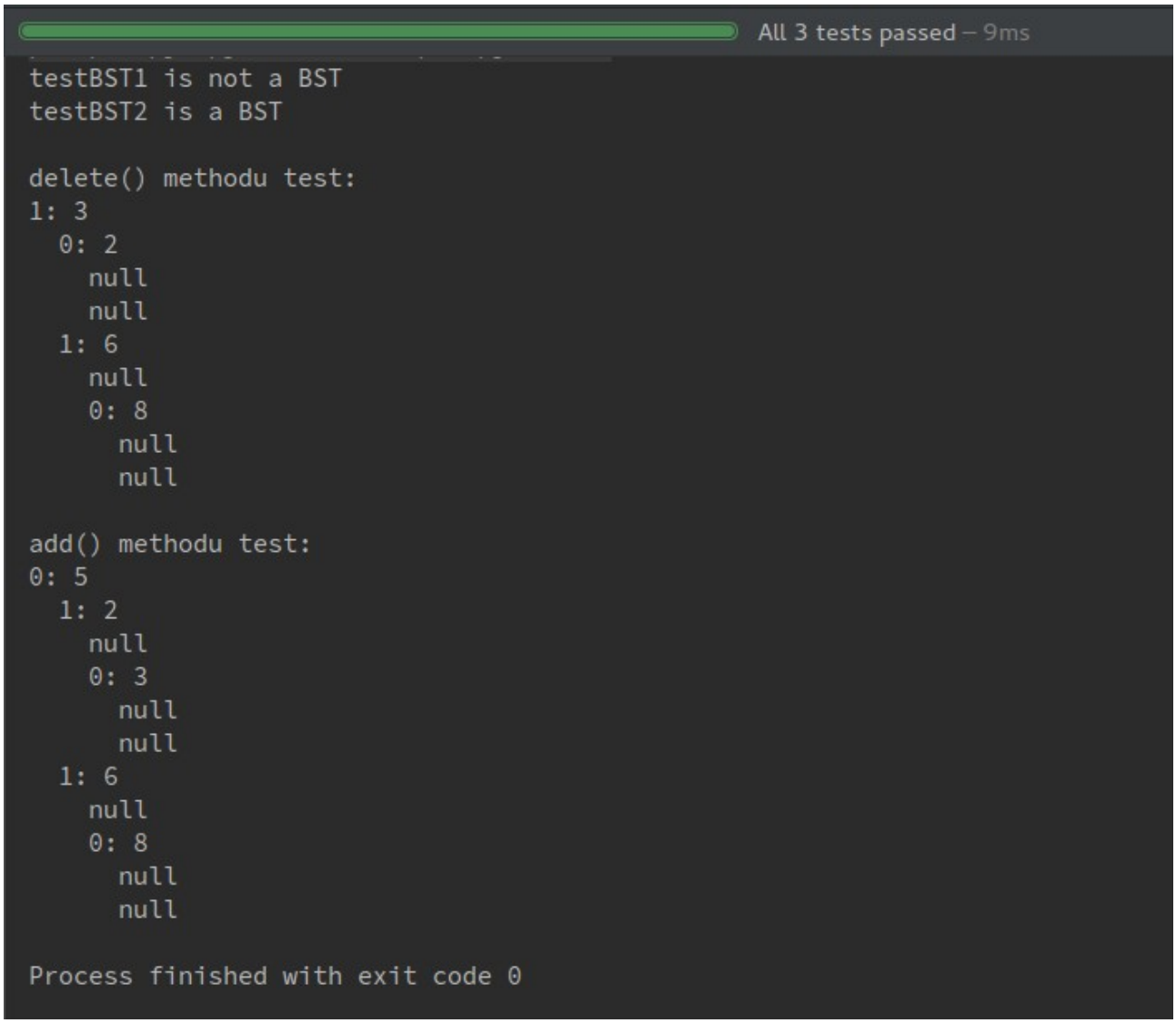

Burada 5 numaralı node'un balance değeri 0, 2 ve 6 numaralı nodeların balance değeri 1, 3 ve 8 numaralı nodeların balance değerleri de 0'dır.

delete() methodu çalıştırıldıktan sonra, üstteki ağaçtan 5 elemanı silinir. Yeni AVL Tree'nin aşağıdaki gibi olması beklenir:




Burada 3 numaralı node'un balance değeri 1, 2 ve 8 numaralı nodeların balance değeri 0, 6 numaları node'un balance değeri ise 1'dir.

## 3.3 Running Commands and Results




Üstteki çıktıda da görüldüğü gibi, Test Caselerin sonuçları beklendiği gibi geldi.