Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

CENGİZ TOPRAK
161044087

Course Assistant: Fatma Nur Esirci

1 Worst RedBlack Tree

This part about Question1 in HW6

1.1 Problem Solution Approach

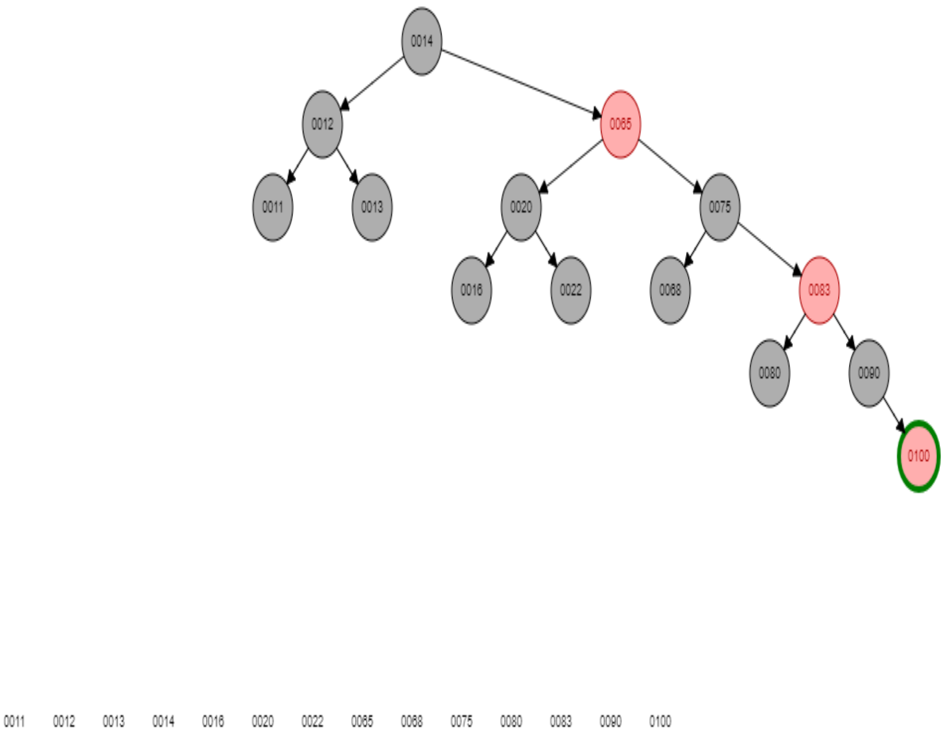
It is implemented absent method named rotateLeft in class BinarySearchWithRotate so that Red Black Tree works correctly.

Pseudocode for rotateLeft(Node<E> root):

```
1-) Recall the value of root.right ( temp = root.right)
2-) Set root.right to temp.left
3-) Set temp.left to root
4-) return temp
```

1.2 Test Cases

example tree 1

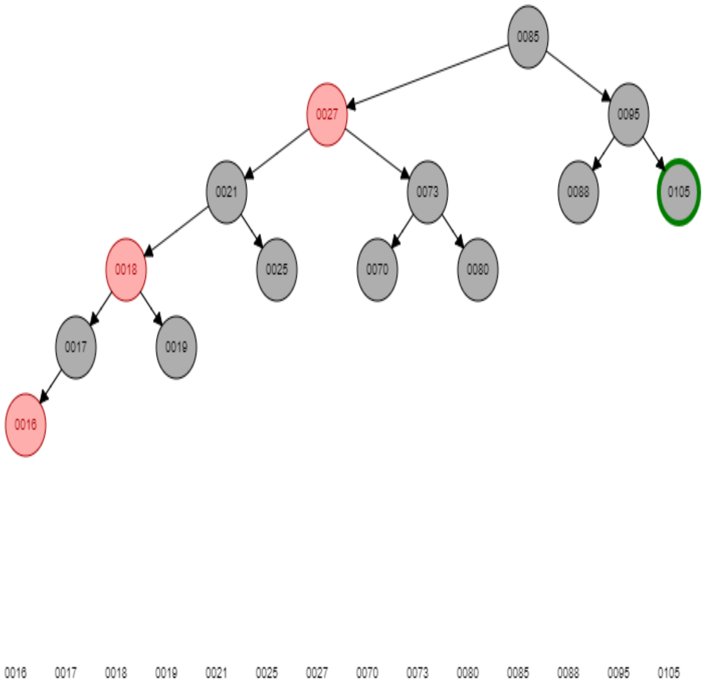


When items added to red black tree ,following operations performed.
Default colours of adding nodes is red.

Adding 11	No rotation and change colour of it to black.
Adding 12	No rotation and No colour changing.
Adding 13	Single rotate left and change colour of 12 to black.
Adding 14	No rotation and change colour of 12 and 13 to black.

Adding 16	Single rotate left ,change colour of 14 to black and 13 to red.
Adding 20	No rotation ,colours of 13 and 16 to black and 14 to red.
Adding 22	Single rotate left , change colour of 16 to red and 20 to black.
Adding 65	Single rotate left , change colour of 14,16 ,22 to black and 20 ,12 red
Adding 68	Single rotate left ,change colour of 65 to black and 22 to red.
Adding 75	No rotation,change colourof 22,68,12,20 to black and 65 to red.
Adding 80	Single rotate left ,change colour of 68 to red and 75 to black.
Adding 83	Single rotate left ,change colour of 65,68,80 to black , 75 and 20 to red.
Adding 90	Single rotate left ,change colour of 83 to black and 80 to red.
Adding 100	No rotation,change colour of 80,90 to black and 83 to red.

example tree 2



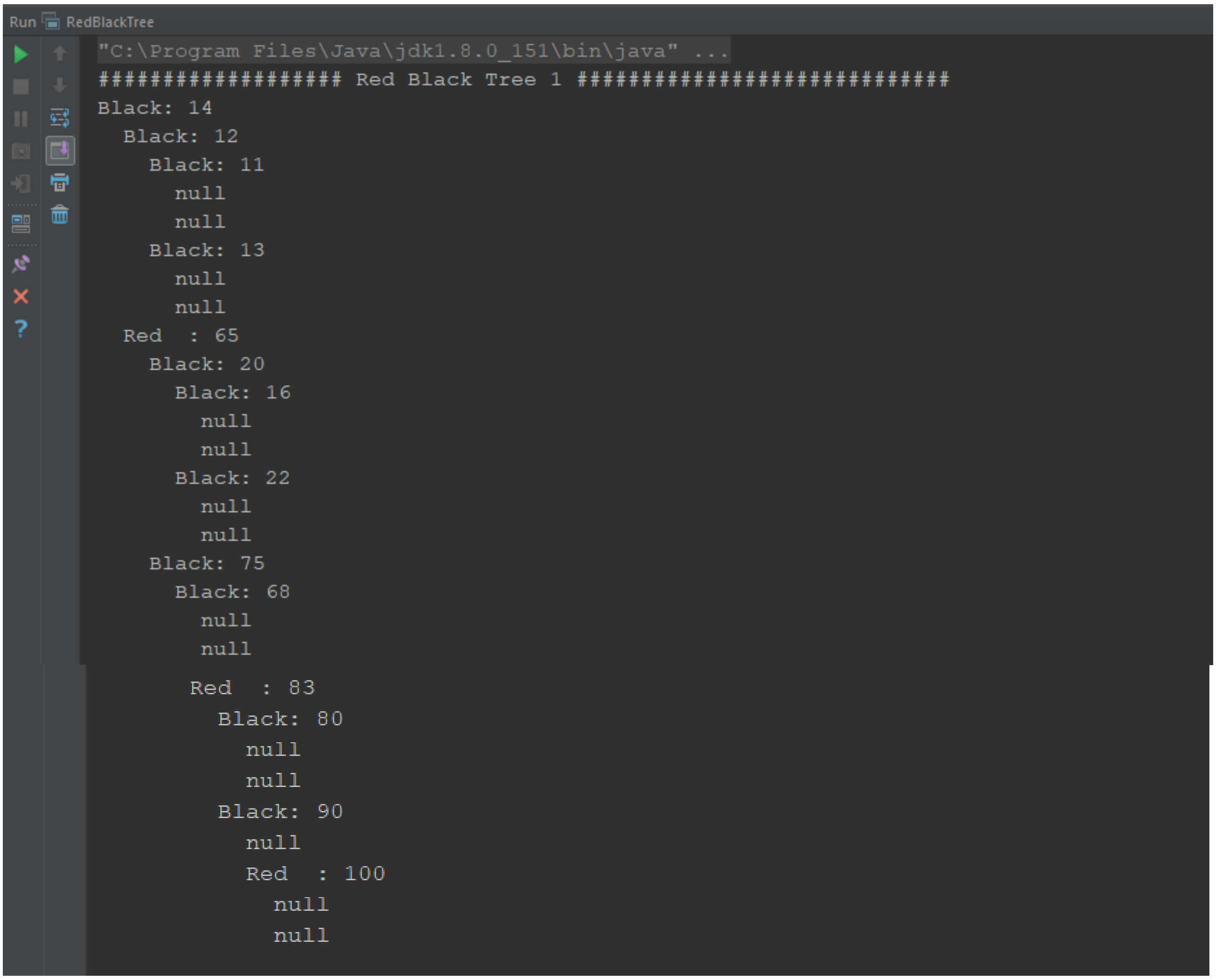
order of items that is added to red black tree is from tail to head.
When items added to red black tree ,following operations performed.
Default colours of adding nodes is red.

Adding 105	No rotation and change colour of it to black.
Adding 95	No rotation and No colour changing.
Adding 88	Single rotate right,change colour of 95 to black and 105 to red.
Adding 65	No rotation,change colour of 88 , 105 to black.
Adding 80	first rotate left then rotate right ,change colour of 80 to black and 88 to red.
Adding 73	No rotation,change colour of 65 and 88 to black and 80 to red.

Adding 70	first rotate right then rotate left , change colour of 70 to black and 65 to red.
Adding 27	change colour of 65 ,73 to black and 70 to red and then rotate right ,change colour of 80 to black and 95 to red.
Adding 25	Single rotate right,change colour of 27 to black and 65 to red.
Adding 21	change colour of 25,65,70,95 to black and 27 to red.
Adding 19	Single rotate right,change colour of 21 to black and 25 to red.
Adding 18	change colour of 19 ,25 to black and 21 to red and rotate right and change colour of 27 to black and 70 to red.
Adding 17	Single rotate right,change colour of 18 to black and 19 to red.
Adding 16	No rotation,change colour of 17 ,19 to black and 18 to red.

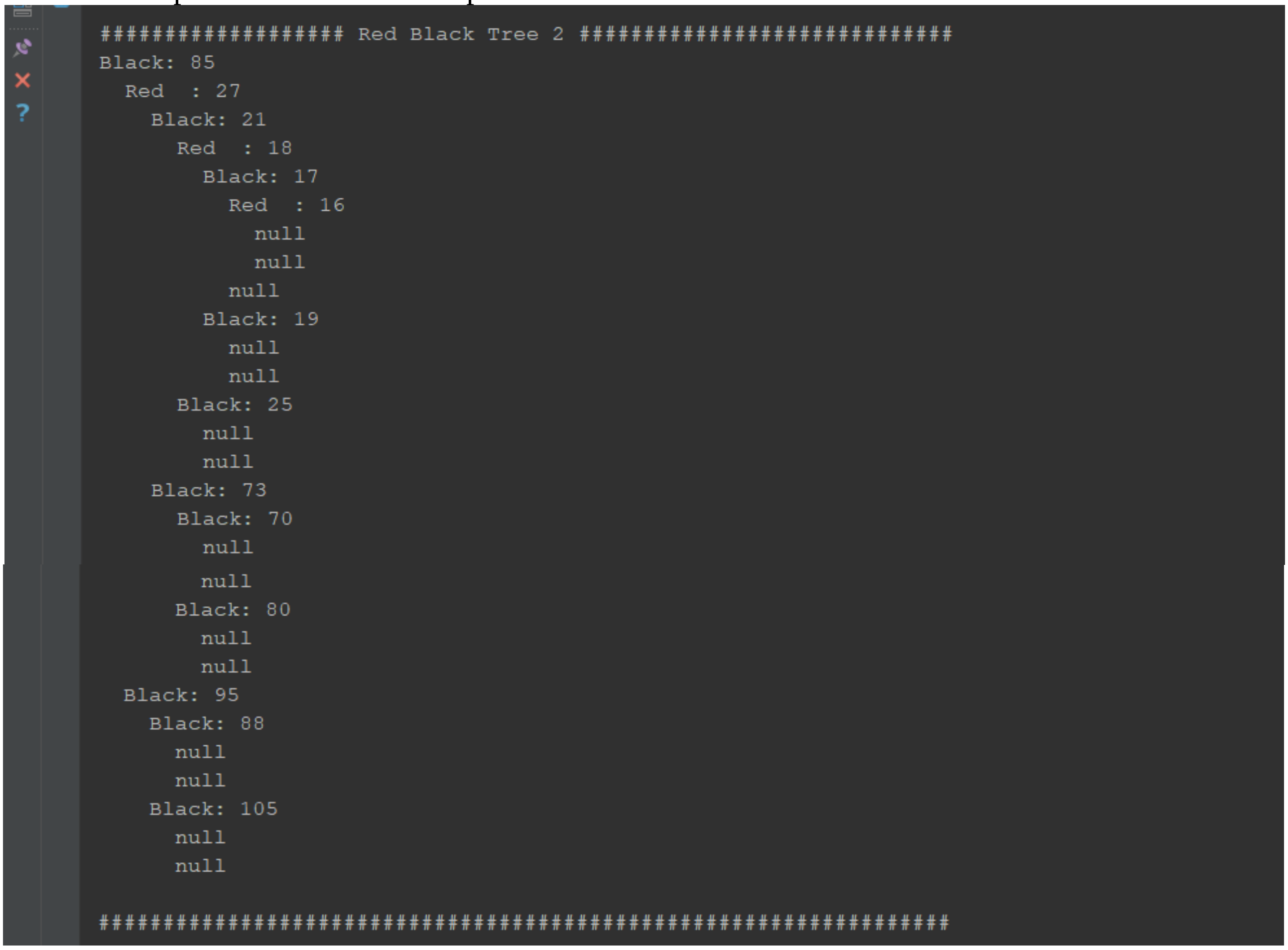
1.3 Running Commands and Results

Example red black tree 1 output.



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
################### Red Black Tree 1 #############################
Black: 14
  Black: 12
    Black: 11
      null
      null
    Black: 13
      null
      null
  Red  : 65
    Black: 20
      Black: 16
        null
        null
      Black: 22
        null
        null
    Black: 75
      Black: 68
        null
        null
      Red  : 83
        Black: 80
          null
          null
        Black: 90
          null
          Red  : 100
            null
            null
```

Example red black tree 2 output.



```
################### Red Black Tree 2 #############################
Black: 85
  Red  : 27
    Black: 21
      Red  : 18
        Black: 17
          Red  : 16
            null
            null
          null
        Black: 19
          null
          null
      Black: 25
        null
        null
    Black: 73
      Black: 70
        null
        null
      Black: 80
        null
        null
  Black: 95
    Black: 88
      null
      null
    Black: 105
      null
      null

##################################################################
```

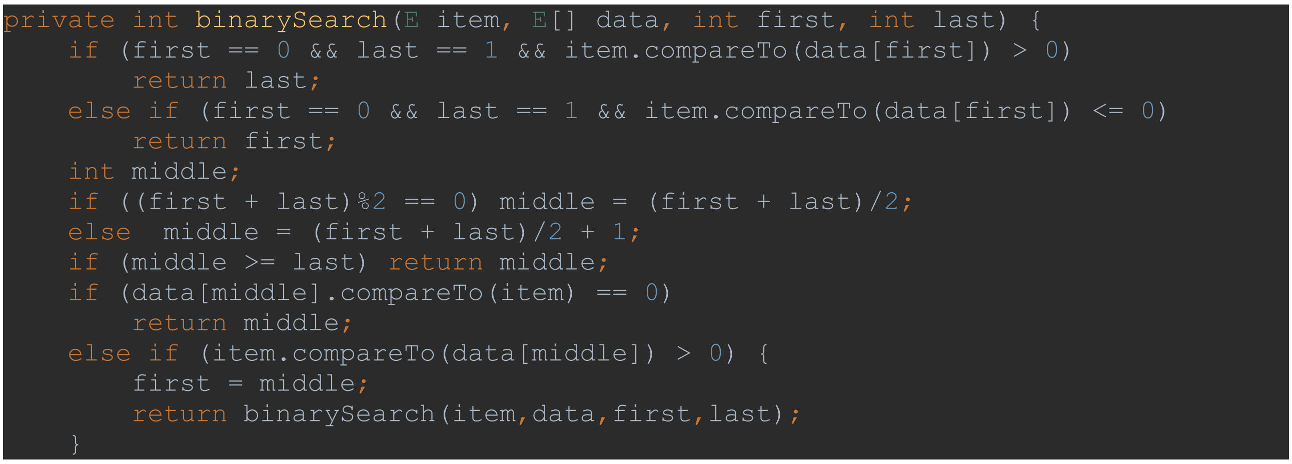
2 binarySearch method

This part about Question2 in HW6

2.1 Problem Solution Approach

It is implemented binarySearch method in order to add items to Btree correctly

Code for binarySearch method:



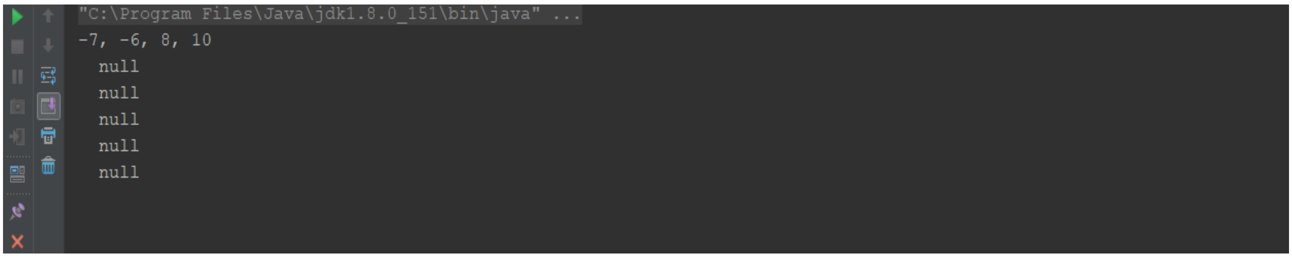
```java
private int binarySearch(E item, E[] data, int first, int last) {
    if (first == 0 && last == 1 && item.compareTo(data[first]) > 0)
        return last;
    else if (first == 0 && last == 1 && item.compareTo(data[first]) <= 0)
        return first;
    int middle;
    if ((first + last)%2 == 0) middle = (first + last)/2;
    else  middle = (first + last)/2 + 1;
    if (middle >= last) return middle;
    if (data[middle].compareTo(item) == 0)
        return middle;
    else if (item.compareTo(data[middle]) > 0) {
        first = middle;
        return binarySearch(item,data,first,last);
    }
    last = middle;
    return binarySearch(item,data,first,last);
}
```

Pseudocode for binarySearch(E item,E[] data,int first,int last):

```
1-)   if (first == 0 and last == 1 and  item > data[first] ) do
              return last.
2-) else if (first == 0 and last == 1 and item <= data[first])  do
           return first.
3-) decleare variable middle
4-) if ((first + last)%2 == 0) do
           set middle  to  (first + last)/2.
5-) else  do
           set middle  to (first + last)/2 + 1.
6-) if (middle >= last) do
           return middle.
7-) if (data[middle].compareTo(item) == 0) do
       return middle.
 8-) else if (item.compareTo(data[middle]) > 0)
        set first to middle.
       call  itself with parameters  item,data,first,last and return value of itself.
9-) set last to middle
10-) call  itself with parameters  item,data,first,last and return value of itself.
```

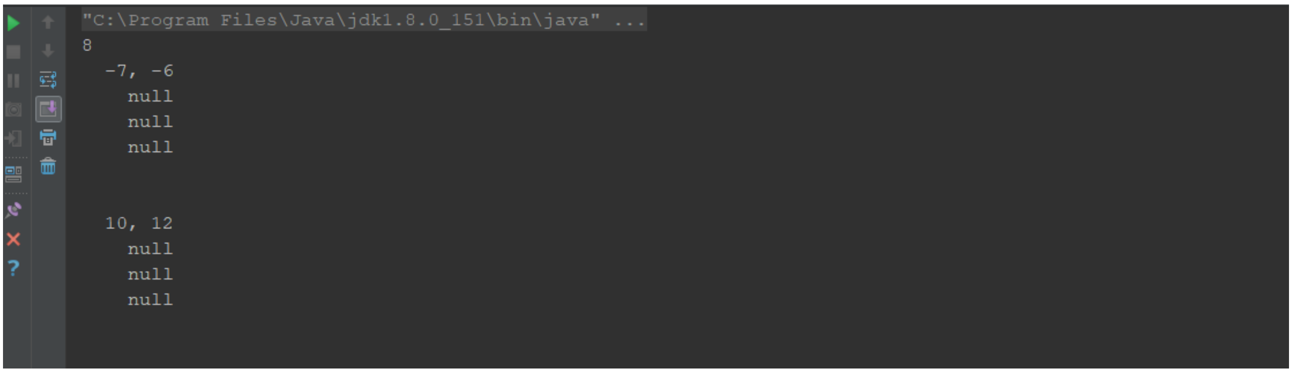
2.2 Test Cases

Btree order 5
Adding items to btree:
add -6 -7 8 10 to tree



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
-7, -6, 8, 10
  null
  null
  null
  null
  null
```

when add 12 to tree btree is splitted.



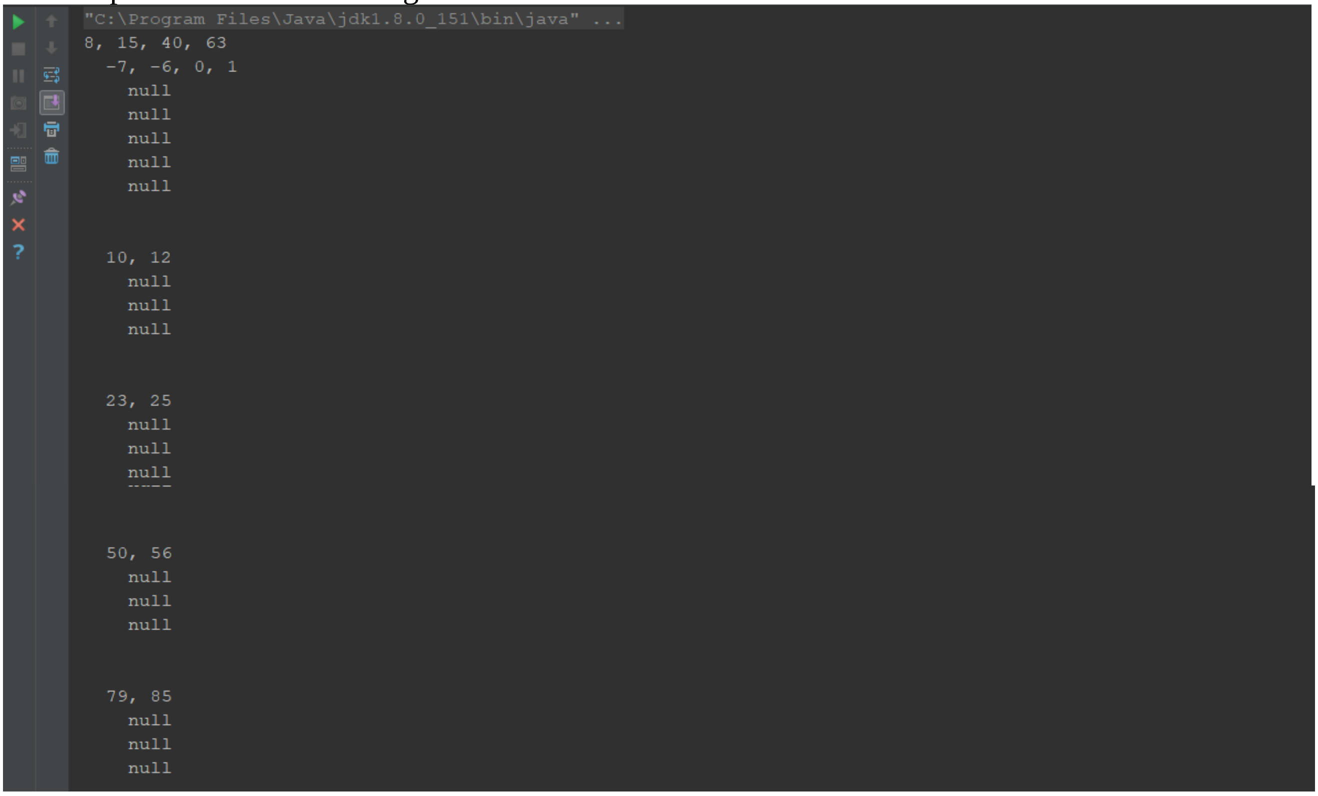
```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
8
  -7, -6
    null
    null
    null


  10, 12
    null
    null
    null
```

As same way,every time array is fulled , btree is splitted itself auotomatically.

Added elements ,in turn,is following order.

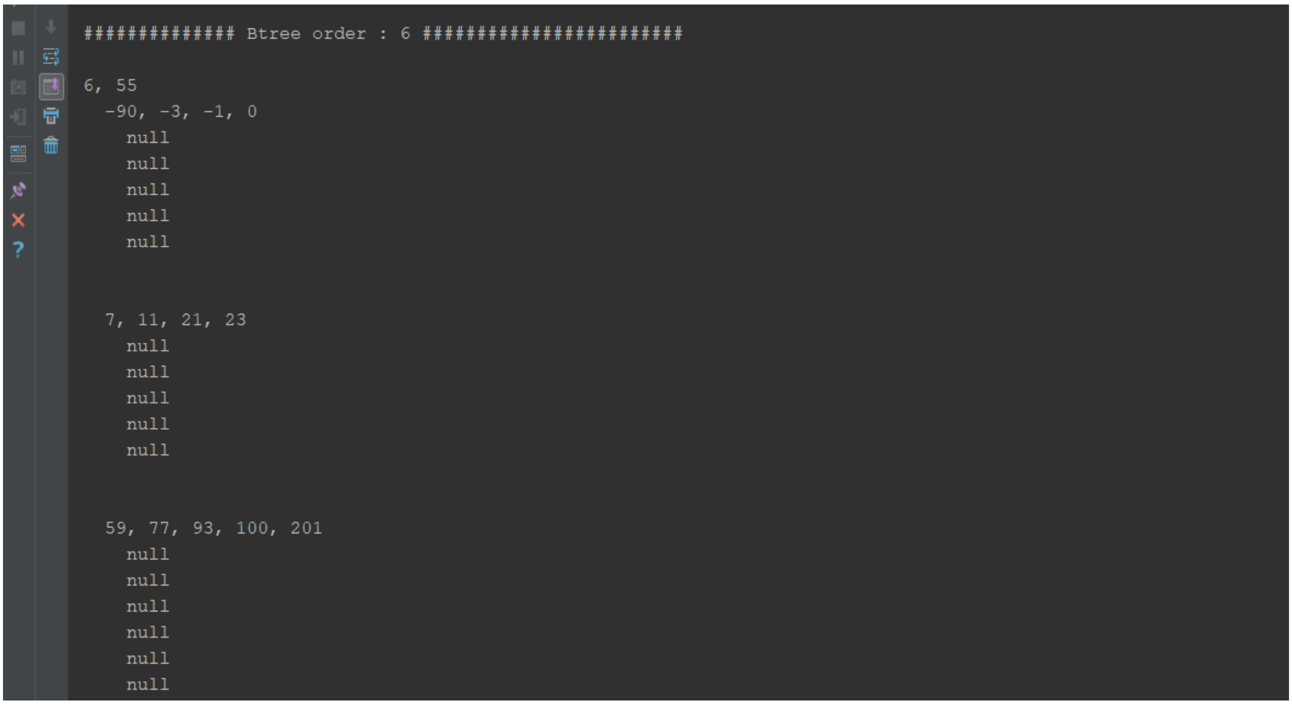
-6 -7 8 10 12 50 63 15 23 25 40 79 85 0 1 56

and output of btree is following.



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
8, 15, 40, 63
  -7, -6, 0, 1
    null
    null
    null
    null
    null


  10, 12
    null
    null
    null


  23, 25
    null
    null
    null


  50, 56
    null
    null
    null


  79, 85
    null
    null
    null
```

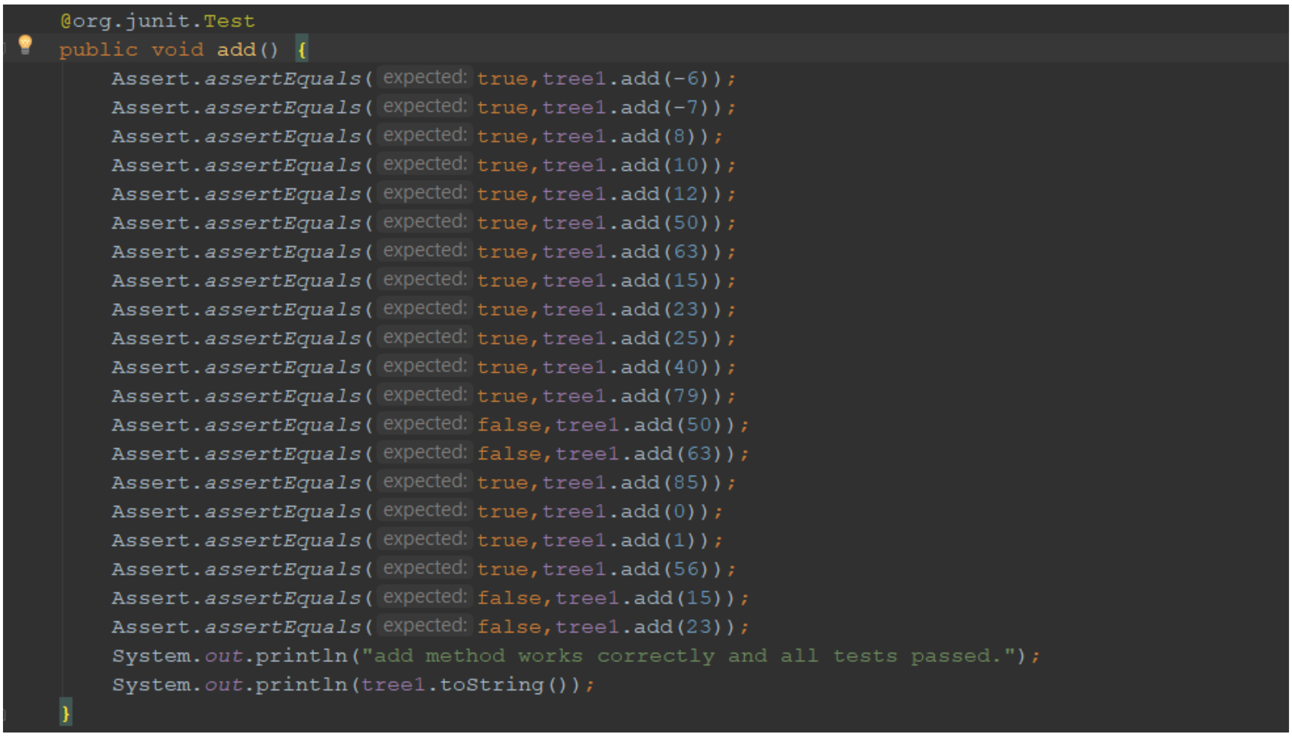
Btree order 6:
Added elements ,in turn,is -1, 6 ,11 ,100 ,55 ,-90 ,23 ,0 ,201 ,-3 ,59 ,77 ,21 ,93 ,7
output of btree is following:



```
############## Btree order : 6 ########################

6, 55
  -90, -3, -1, 0
    null
    null
    null
    null
    null


  7, 11, 21, 23
    null
    null
    null
    null
    null


  59, 77, 93, 100, 201
    null
    null
    null
    null
    null
    null
```

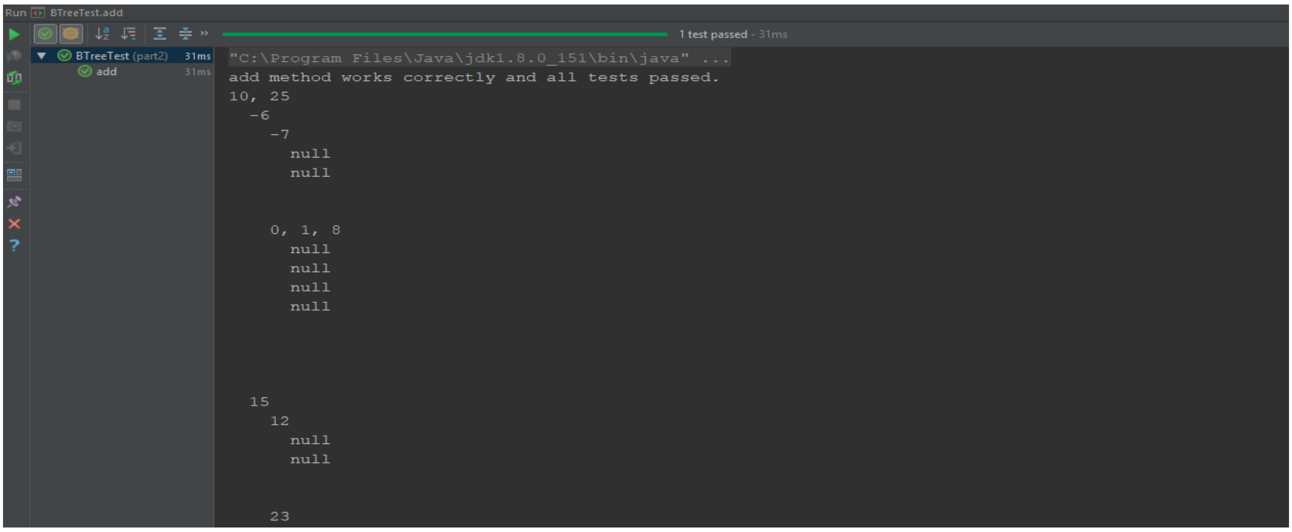
2.3 Running Commands and Results

Btree order 4:

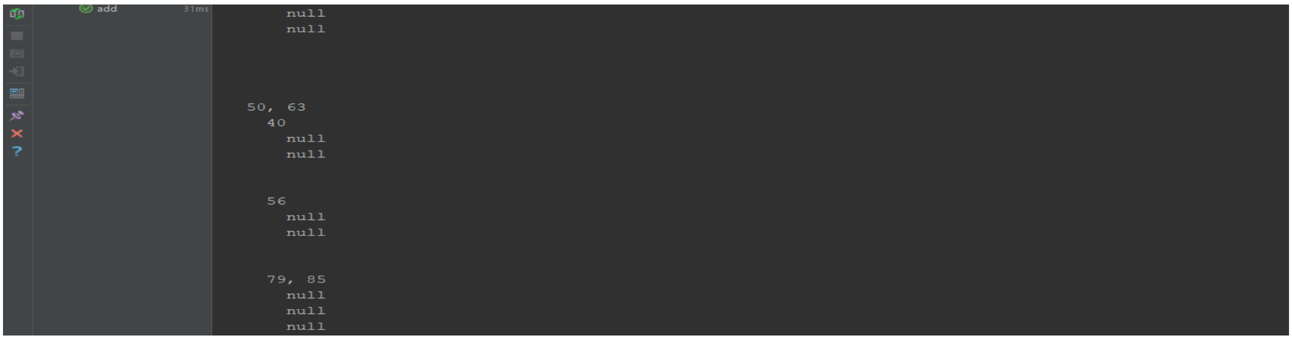


```
    @org.junit.Test
    public void add() {
        Assert.assertEquals( expected: true,tree1.add(-6));
        Assert.assertEquals( expected: true,tree1.add(-7));
        Assert.assertEquals( expected: true,tree1.add(8));
        Assert.assertEquals( expected: true,tree1.add(10));
        Assert.assertEquals( expected: true,tree1.add(12));
        Assert.assertEquals( expected: true,tree1.add(50));
        Assert.assertEquals( expected: true,tree1.add(63));
        Assert.assertEquals( expected: true,tree1.add(15));
        Assert.assertEquals( expected: true,tree1.add(23));
        Assert.assertEquals( expected: true,tree1.add(25));
        Assert.assertEquals( expected: true,tree1.add(40));
        Assert.assertEquals( expected: true,tree1.add(79));
        Assert.assertEquals( expected: false,tree1.add(50));
        Assert.assertEquals( expected: false,tree1.add(63));
        Assert.assertEquals( expected: true,tree1.add(85));
        Assert.assertEquals( expected: true,tree1.add(0));
        Assert.assertEquals( expected: true,tree1.add(1));
        Assert.assertEquals( expected: true,tree1.add(56));
        Assert.assertEquals( expected: false,tree1.add(15));
        Assert.assertEquals( expected: false,tree1.add(23));
        System.out.println("add method works correctly and all tests passed.");
        System.out.println(tree1.toString());
    }
```

Above picture show that which elements is will be adding to tree.
As seen above ,same elements should not be added to btree.
Result of the above test is below:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
add method works correctly and all tests passed.
10, 25
  -6
    -7
      null
      null


    0, 1, 8
      null
      null
      null
      null




  15
    12
      null
      null


    23
```



```
    null
    null



50, 63
  40
    null
    null


  56
    null
    null


  79, 85
    null
    null
    null
```

3 Project 9.5 in book

This part about Question3 in HW6

3.1 Problem Solution Approach

Code for new constructor:

```
public  AVLTree(BinaryTree<E> tree) throws Exception{
    boolean result = isAvlTree(tree);
    if (result) {
        ArrayList<E> arr = new ArrayList<>();
        addAllNodesToList(tree,arr);
        for (E item : arr)
            add(item);
    }
    else throw new Exception("Tree is not AVL tree.");
}
```

Pseudocode for new constructor:

```
public  AVLTree(BinaryTree<E> tree) throws Exception :
1-) decleare variable result in type of boolean.
2-) set result to value of called isAvlTree(tree) function.
3-) if (result == true) do //tree is avl tree.
        decleare an array list arr.
        create new array list.
        set arr to new created array list.
        call addAllNodesToList function parameters with tree,arr.
        add all elements in arr into avl tree.
4-) else throw new Exception("Tree is not AVL tree.").
```

Code for isAvlTree method:

```
private boolean isAvlTree(BinaryTree<E> tree) {
    if (maxDepth(tree) - minDepth(tree) <= AVLNode.RIGHT_HEAVY)
        return true;
    else return false;
}
```

Pseudo code for isAvlTree:

```
private boolean isAvlTree(BinaryTree<E> tree):
1-) if (return value of maxDepth(tree) - return value of minDepth(tree) <=
        AVLNode.RIGHT_HEAVY) do
        return true.
2-) else return false.
```

Code for maxDepth:

```java
private int maxDepth(BinaryTree<E> tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + Math.max(maxDepth(tree.getLeftSubtree()),
                                      maxDepth(tree.getRightSubtree()));
}
```

Pseudo code for maxDepth:

```
private int maxDepth(BinaryTree<E> tree):
1-) if (tree == null) do
        return 0
2-) return 1 + max height of (maxDepth(tree.leftTree) ,maxDepth(tree.rightTree))
```

Code for minDepth:

```java
private int minDepth(BinaryTree<E> tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + Math.min(minDepth(tree.getLeftSubtree()),
                 minDepth(tree.getRightSubtree()));
}
```

Pseudo code for minDepth:

```
private int minDepth(BinaryTree<E> tree) :
1-) if (tree == null) do
        return 0
2-) return 1 + min height of (minDepth(tree.leftTree),minDepth(tree.rightTree))
```

Code for addAllNodesToList :

```java
private void addAllNodesToList(BinaryTree<E> tree, ArrayList<E> list) {
    if (tree != null) {
        list.add(tree.getData());
        addAllNodesToList(tree.getLeftSubtree(), list);
        addAllNodesToList(tree.getRightSubtree(), list);
    }
}
```

Pseudo code for addAllNodesToList :

```
private void addAllNodesToList(BinaryTree<E> tree, ArrayList<E> list):
1-) if (tree != null) do
        add data of tree to list.
        call addAllNodesToList function parameters with left of tree and list.
        call addAllNodesToList function parameters with right of tree and list.
```

Code for removal method:

```
public E removal(E item) {
    decrease = false;
    root = removalHelper( (AVLNode < E > ) root, item);
    return deleteReturn;
}
```

Pseudo code for removal method:

```
public E removal(E item) :
1-) set decrease to false.
2-) set root to return value of removalHelper( (AVLNode < E > ) root, item).
3-) return deleteReturn.
```

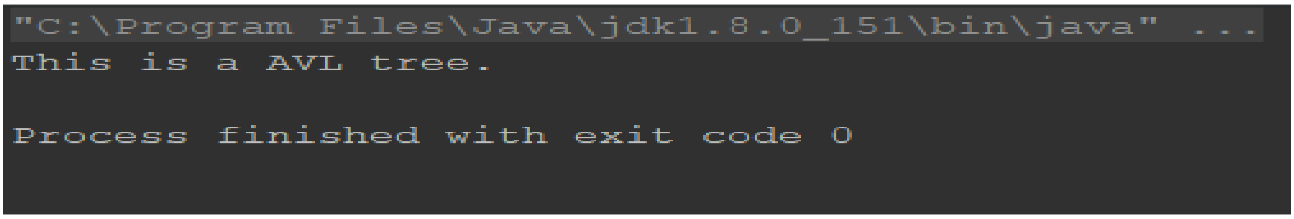
3.2 Test Cases

Result of test for new constructor:

--Balanced binary tree

```
public static void main(String[] args) {
    BinarySearchTree<Integer> tree = new BinarySearchTree<>();
    //BinarySearchTree<Integer> tree1 = new BinarySearchTree<>();
    /*This tree is balanced binar tree.*/
    tree.add(12);
    tree.add(5);
    tree.add(4);
    tree.add(11);
    tree.add(6);
    tree.add(14);
    tree.add(15);
    tree.add(13);
    try {
        AVLTree<Integer> avltree = new AVLTree<>(tree);
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(-1);
    }
}
```

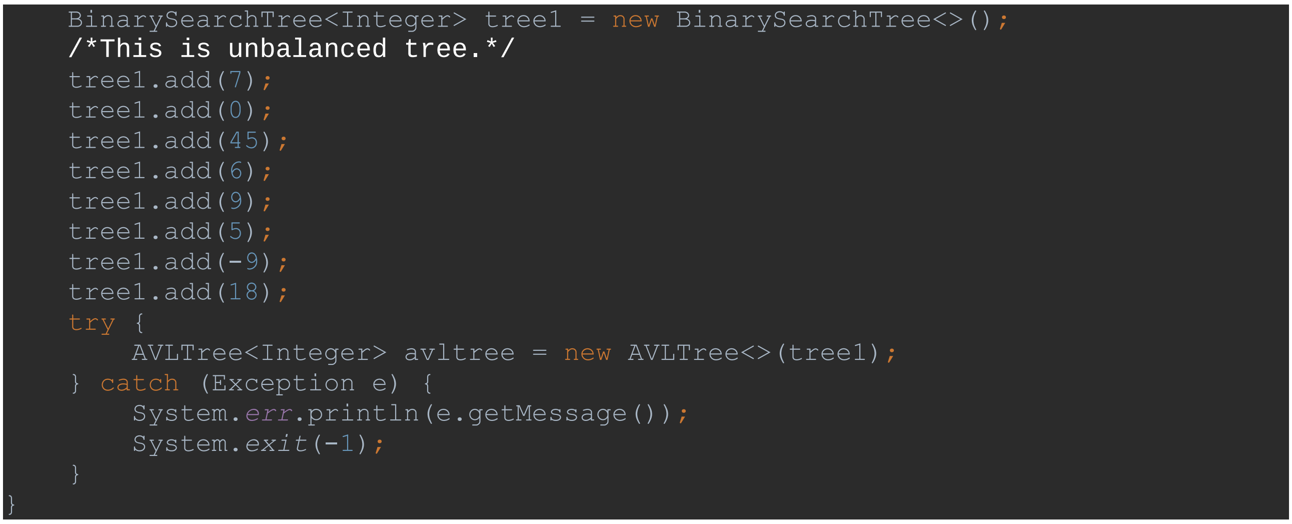
output :



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
This is a AVL tree.

Process finished with exit code 0
```

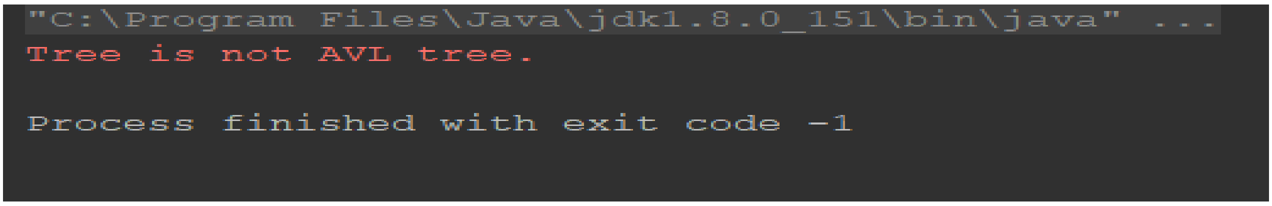
--Unbalanced binary tree

```
public static void main(String[] args) {
    //BinarySearchTree<Integer> tree = new BinarySearchTree<>();
```



```
    BinarySearchTree<Integer> tree1 = new BinarySearchTree<>();
    /*This is unbalanced tree.*/
    tree1.add(7);
    tree1.add(0);
    tree1.add(45);
    tree1.add(6);
    tree1.add(9);
    tree1.add(5);
    tree1.add(-9);
    tree1.add(18);
    try {
        AVLTree<Integer> avltree = new AVLTree<>(tree1);
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(-1);
    }
}
```

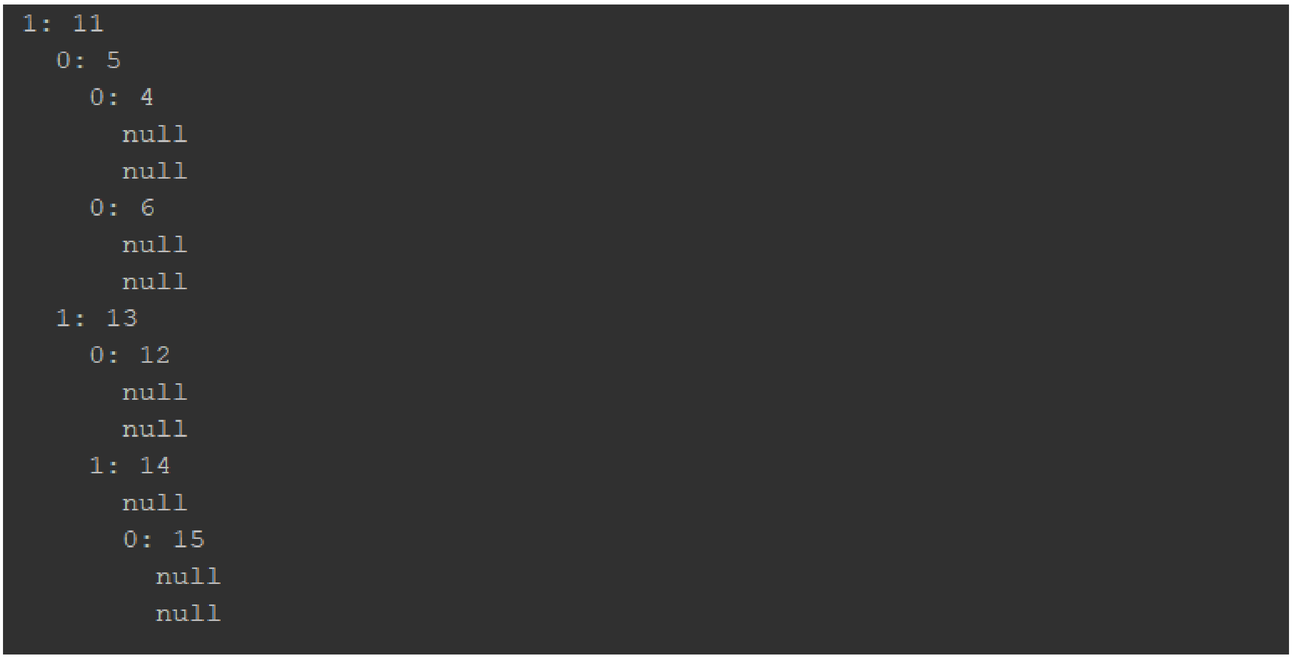
output:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
Tree is not AVL tree.

Process finished with exit code -1
```

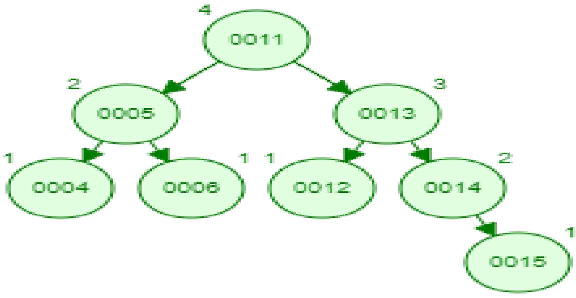
This outputs show that all functions new contructor used is working correctly.

Result of removal :
before remove 14 and 11 avl tree.

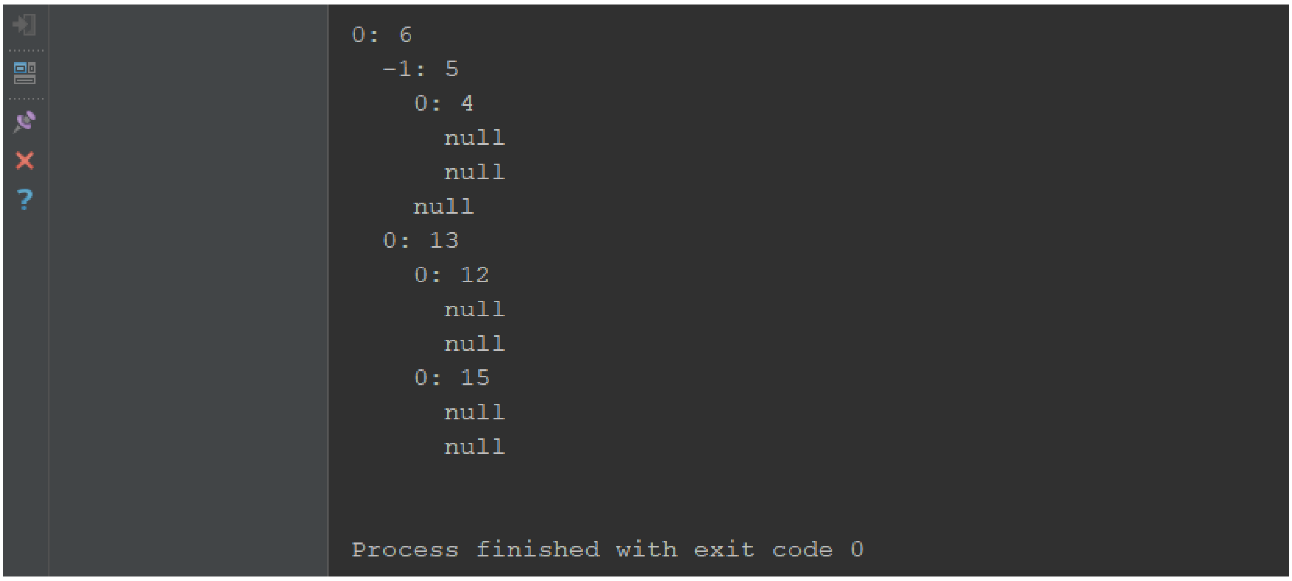


```
1: 11
  0: 5
    0: 4
      null
      null
    0: 6
      null
      null
  1: 13
    0: 12
      null
      null
    1: 14
      null
      0: 15
        null
        null
```

visualization of tree:

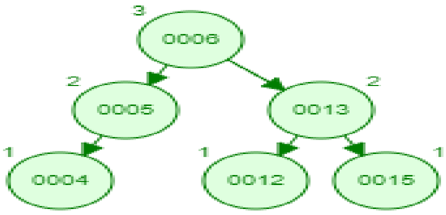


After remove 14 and 11 avl tree.



```
0: 6
  -1: 5
    0: 4
      null
      null
    null
  0: 13
    0: 12
      null
      null
    0: 15
      null
      null


Process finished with exit code 0
```

visualization of tree after removal:

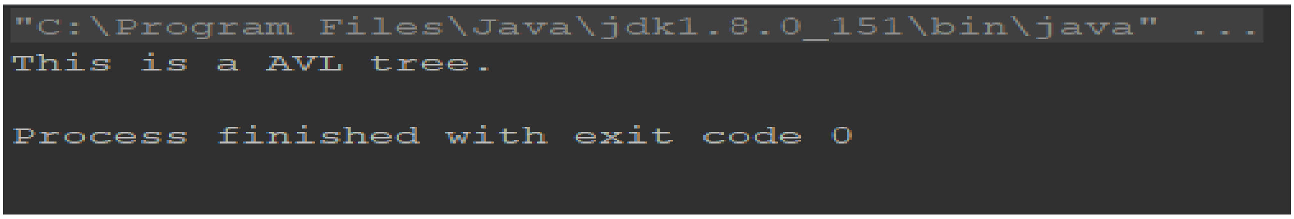


3.3 Running Commands and Results

İnput for new constructor:

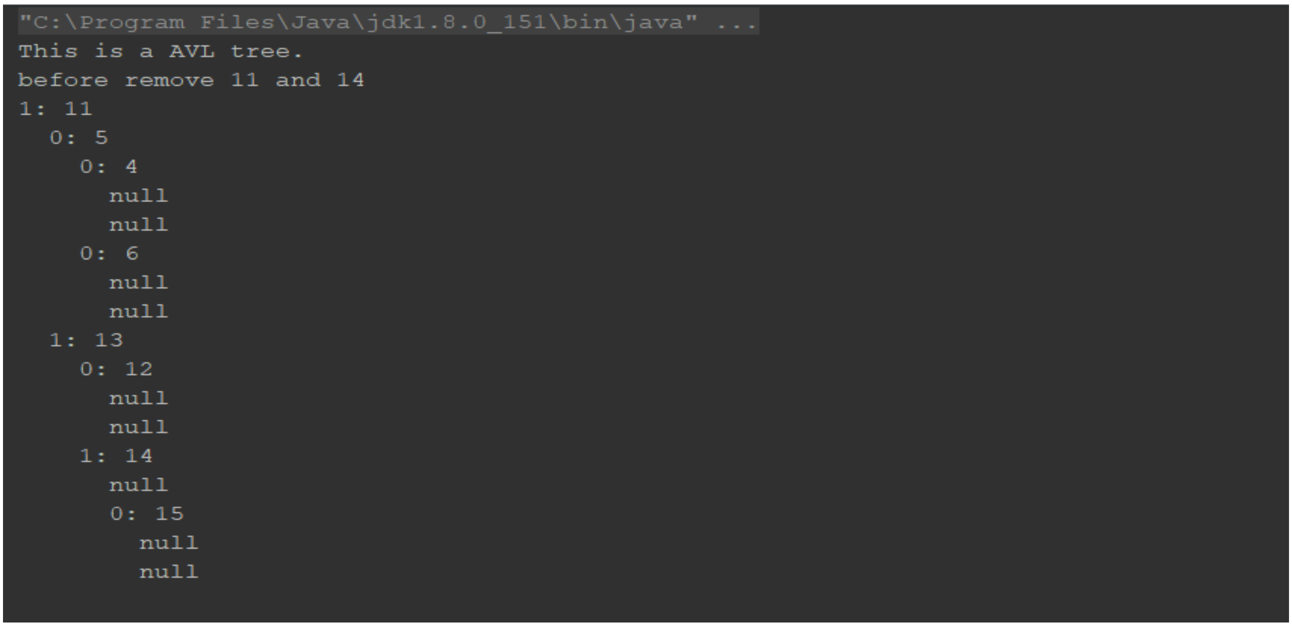
```
public static void main(String[] args) {
    BinarySearchTree<Integer> tree = new BinarySearchTree<>();
    /*This tree is balanced binar tree.*/
    tree.add(12);
    tree.add(5);
    tree.add(4);
    tree.add(11);
    tree.add(6);
    tree.add(14);
    tree.add(15);
    tree.add(13);
    try {
        AVLTree<Integer> avltree = new AVLTree<>(tree);
    } catch (Exception e) {
        System.err.println(e.getMessage());
        System.exit(-1);
    }
}
```

output for new constructor:

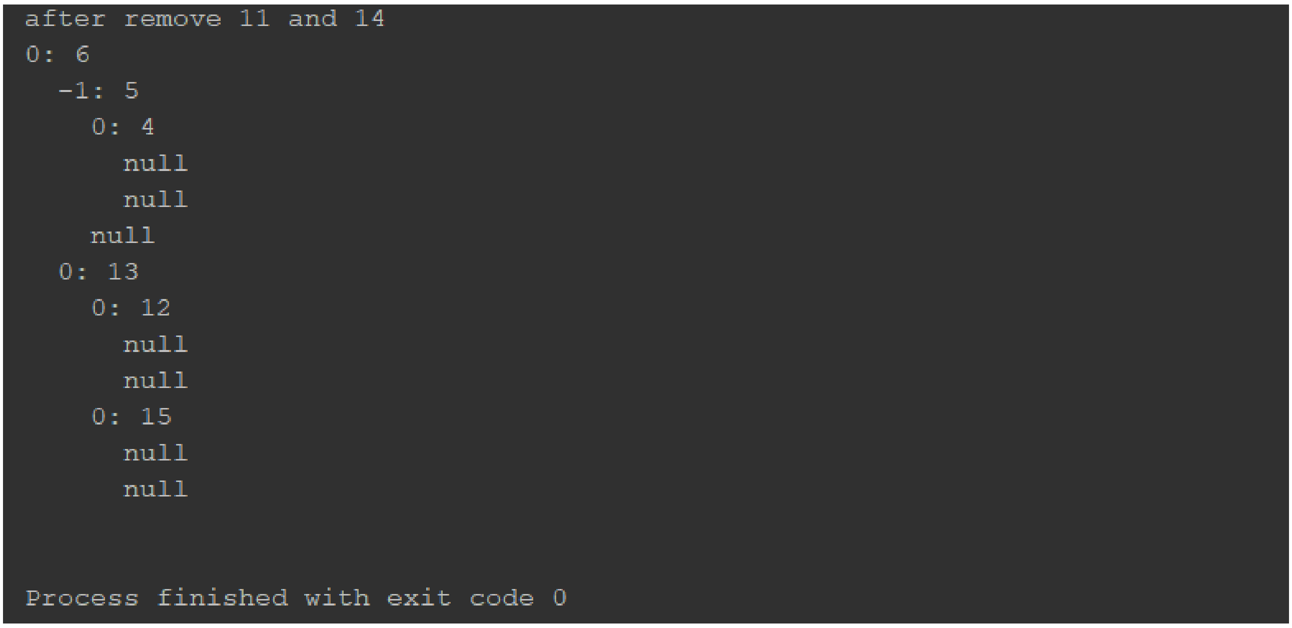


```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
This is a AVL tree.

Process finished with exit code 0
```

running result of removal method:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
This is a AVL tree.
before remove 11 and 14
1: 11
  0: 5
    0: 4
      null
      null
    0: 6
      null
      null
  1: 13
    0: 12
      null
      null
    1: 14
      null
      0: 15
        null
        null
```



```
after remove 11 and 14
0: 6
  -1: 5
    0: 4
      null
      null
    null
  0: 13
    0: 12
      null
      null
    0: 15
      null
      null


Process finished with exit code 0
```