Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 6 REPORT

SEVGİ BAYANSALDUZ
151044076

Course Assistant: Fatma Nur Esirci

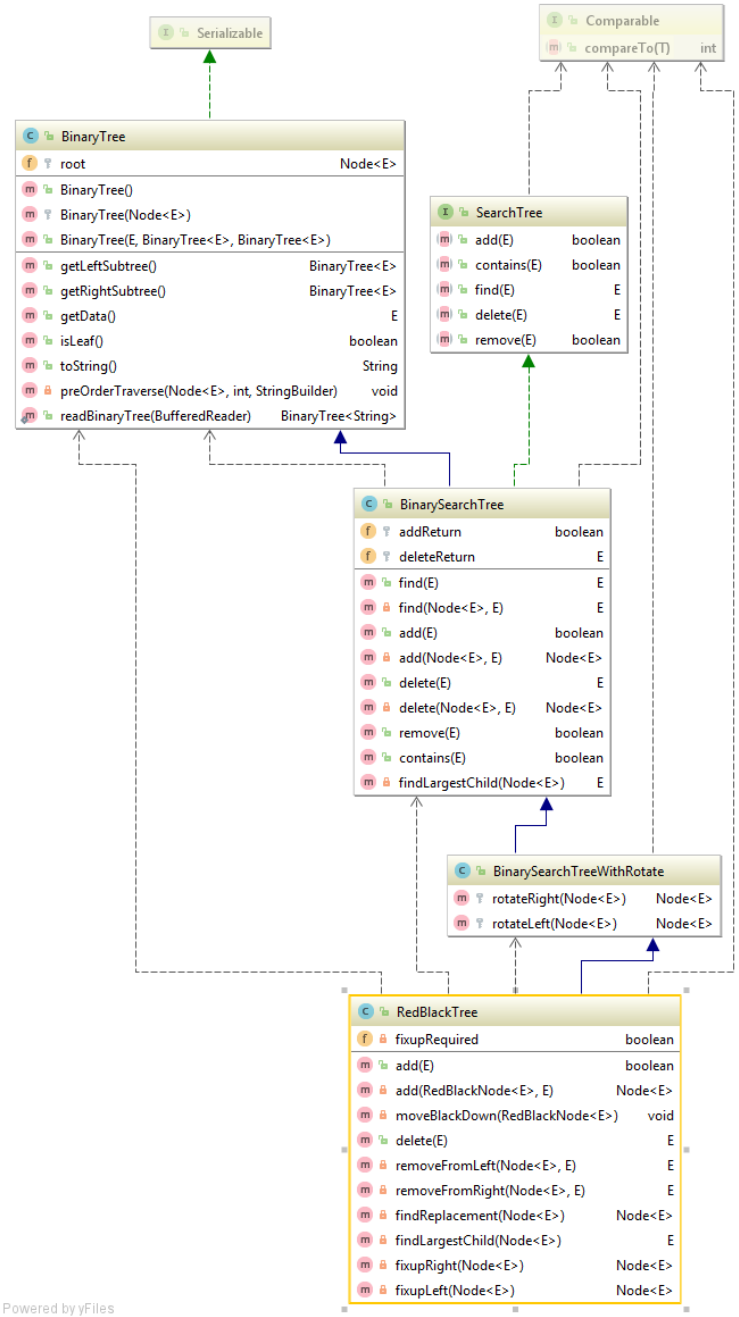
1 Worst RedBlack Tree

This part about Question1 in HW6

1.1 Problem Solution Approach

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.
In this part, I first completed the missing rotateLeft method in the class BinarySearchTreeWithRotate, the missing part of the add method in the RedBlackTree class, and the moveBlackDown method in the RedBlackTree class. (I got the methods for deletion from the our text book`s source code.)

1.1.1 Class Dİagram



The class RedBlackTree is deriving from the class BinarySearchWithRotate.

UML class diagram showing the ralationship between RedBlackTree and BinarySearchWithRotate. RedBlackTree overrides the add and delete methods.

RedBlackTree has an inner class that name is RedBlackNode. RedBlackNode class extends the nested class BinaryTree.Node. The RedBlackNode class has the additional data field isRed to indicate red nodes.

1.1.2 Pseudocodes

- The add method

Add(E item)

1. İf root is null
2. Insert a new RedBlackNode (color it black)
3. Return true
4. else if item == root.data
5. Return false
6. else if item < root.data
7. **if root.getLeftSubtree is null**
8. Insert a new RedBlackNode (color it red)
9. Return true
10. **else**
11. **if left.child and right child are red**
12. set color of localroot is red and set color of the children black
13. Recursively add item into the left subtree
14. **if left child is red**
15. **if left grandchild is red**
16. set the color of left child to black and the localroot to red
17. rotate local root right
18. **else if right grandchild is red**
19. Rotate the left child left
20. set the color of the leftchild black and local root to red.
21. rotate local root right.
22.else //item>root.data
23. processs is symmetric from step 6 to step 21
24. if localRoot is root
25. make color of the localRoot black

```
} else { // item > localRoot.data
    /* ********************************************************************************** */
t solution to programming exercise 1, section 3, chapter 9 here
    if (localRoot.right == null) {// create new right child
        localRoot.right = new RedBlackNode < E > ( (E) item);
        addReturn = true;
        return localRoot;
    }
    else {
        moveBlackDown(localRoot);
        localRoot.right = add( (RedBlackNode < E > ) localRoot.right, item);// recursively insert on
        if ( ( (RedBlackNode) localRoot.right).isRed) {
            if (localRoot.right.right != null
                    && ( (RedBlackNode) localRoot.right.right).isRed) {
                // right-right grandchild is also red-single rotate is necessary
                ( (RedBlackNode) localRoot.right).isRed = false;
                localRoot.isRed = true;
                return rotateLeft(localRoot);
            }
            else if (localRoot.right.left != null && ( (RedBlackNode) localRoot.right.left).isRed) {
                // left-right grandchild is also red- double rotate is necessary
                localRoot.right = rotateRight(localRoot.right);
                ( (RedBlackNode) localRoot.right).isRed = false;
                localRoot.isRed = true;
                return rotateLeft(localRoot);
            }
        }
        return localRoot;
        /* ********************************************************************************** */
    }
}
```

- The rotateLeft method

rotateLeft(Node root)

1. Remember the value of root.right
2. Set root.left to temp.right
3. Set temp.right to root
4. Set root to temp

```
// Insert solution to programming exercise 1, section 1, chapter 9 here
    protected Node < E > rotateLeft(Node < E > root) {
        Node < E > temp = root.right;
        root.right = temp.left;
        temp.left = root;
        return temp;
    }
```

- The moveBackDown method

moveBackDown(RedBlackTree root)

1. Root.right and root.right are not null Also they are red
2. Make them black
3. Make root red

```
private void moveBlackDown(RedBlackNode < E > root) {
    //  if both children are red
    if (root.left != null && root.right != null
            && ( (RedBlackNode) root.left).isRed
            && ( (RedBlackNode) root.right).isRed) {
        //make the children black and localRoot red
        ( (RedBlackNode) root.left).isRed = false;
        ( (RedBlackNode) root.right).isRed = false;
        root.isRed = true;
    }
}
```

1.1.3 Worst-Case Tree

The worst red black tree with height of 6 includes 22 elements.Elements is inserted in ascending or descending order.

1.2 Test Cases

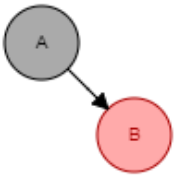
1.2.1 TEST1

```
/**To worst-case*/
public class MainTest {
    public static void main(String[] args) {
        RedBlackTree<String> test=new RedBlackTree<>();
        for(char i='A';i<'V';++i)
            test.add(String.valueOf(i));
        System.out.println(test.toString());
    }
}
```

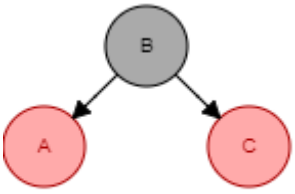
In the main test,main method inserts the alphabet letters from A to V in ascending order.

```
Black: H
  Black: D
    Black: B
      Black: A
        null
        null
      Black: C
        null
        null
    Black: F
      Black: E
        null
        null
      Black: G
        null
        null
  Black: L
    Black: J
      Black: I
        null
        null
      Black: K
        null
        null
    Red  : P
      Black: N
        Black: M
          null
          null
        Black: O
          null
          null
      Black: R
        Black: Q
          null
          null
        Black: T
          Red  : S
            null
            null
          Red  : U
            null
            null
```

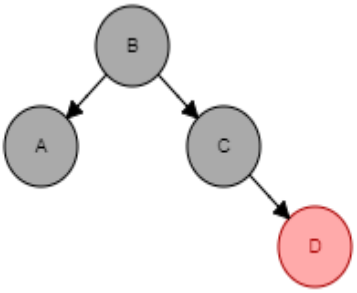
Inserting Steps



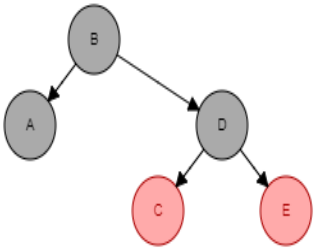
A and B are inserted simply



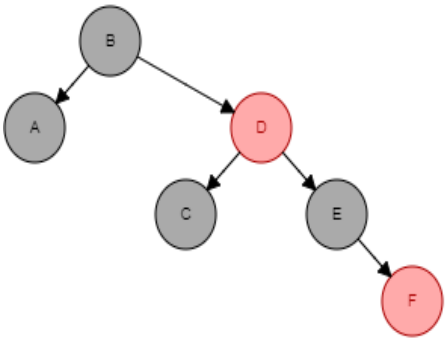
C is inserted to the right of the B.Then B is rotated to left



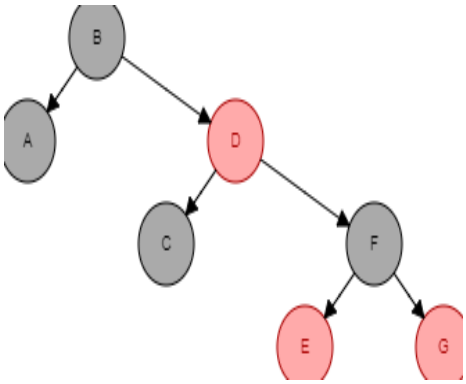
D is inserted to right of the C.Then color of A and C is changed to black.



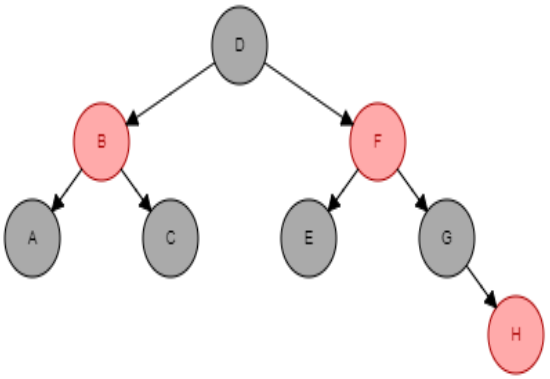
E inserted the right of the D.Then D is rotated to left.



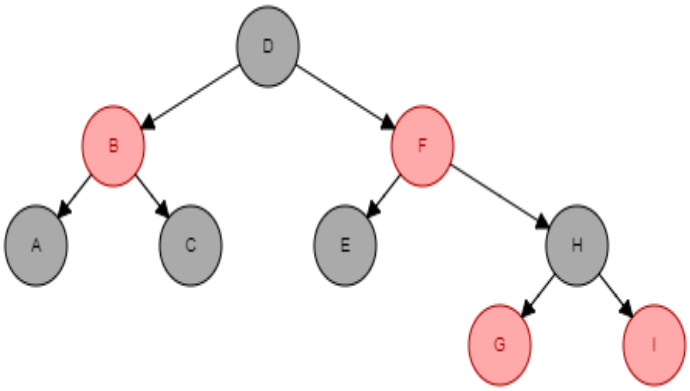
D is inserted to right of the C.Then color of A and C is changed to black,color of D is changet to red.



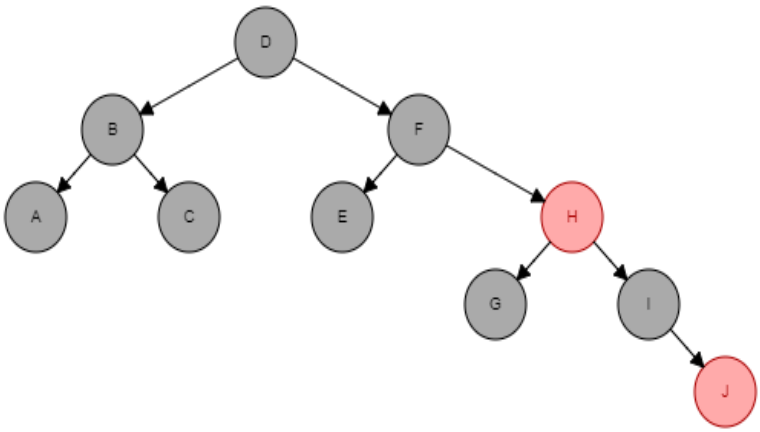
G inserted the right of the F. Then F is rotated to left.



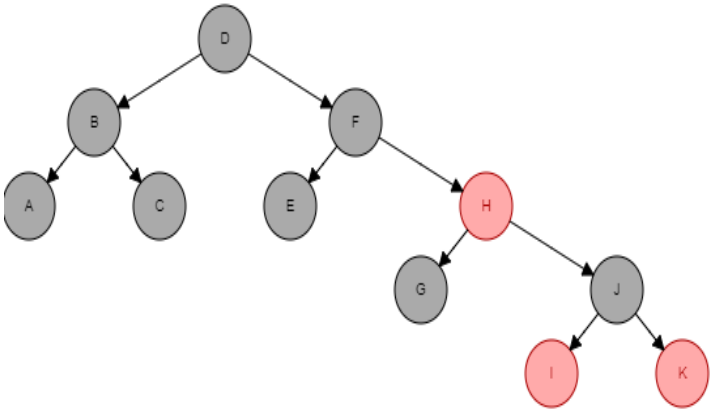
H is inserted to right of the G. Then D is rota left.After rotatig color of D ,E and G is chang black, color of F changed to the red.



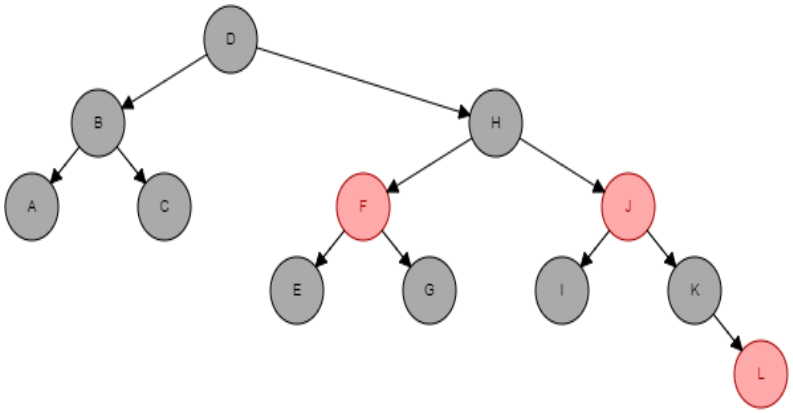
Single rotate left



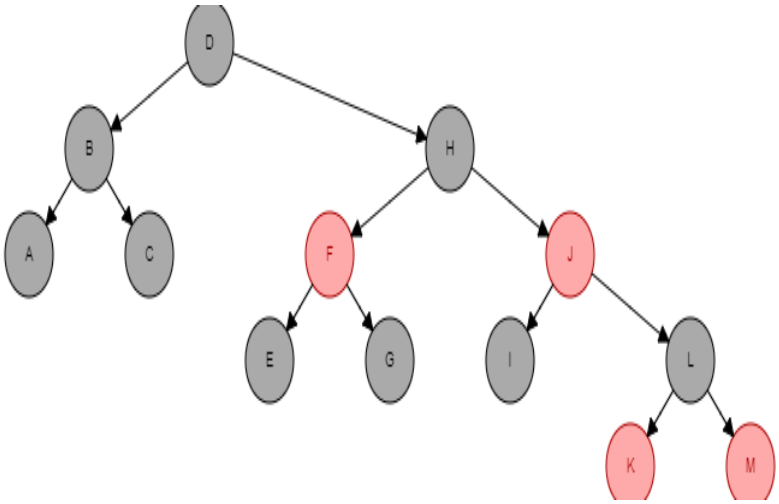
Colors are cahnged



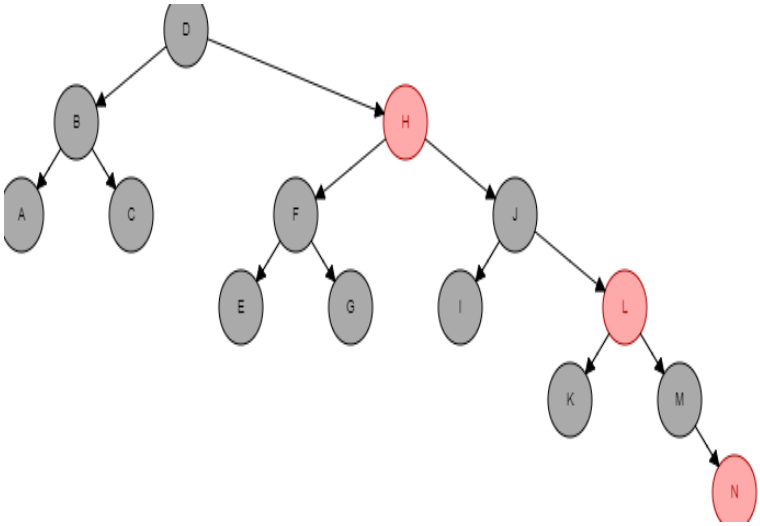
Single rotte left



Colors are changed



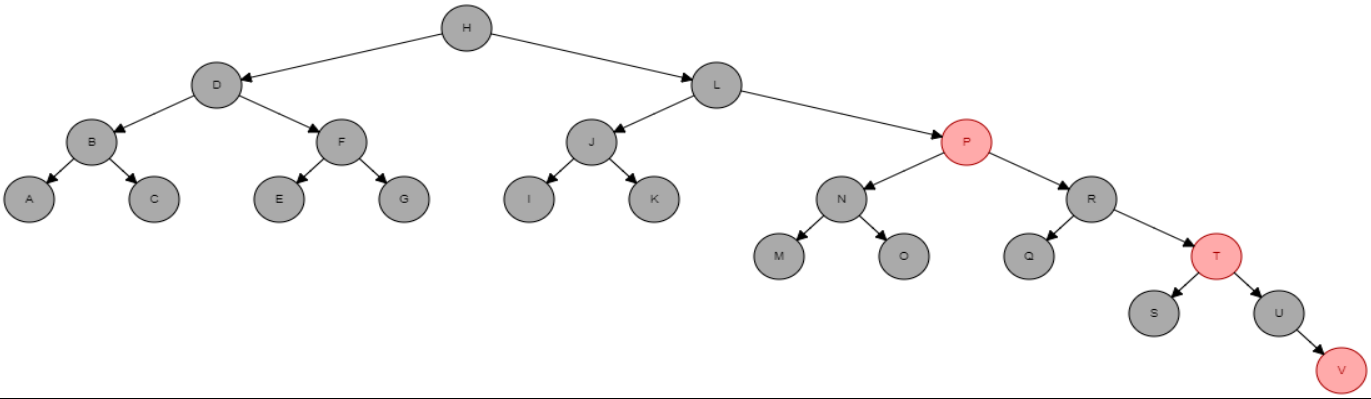
Single rotate left



Colors are changed

- The other steps are similar to the steps above.(Other steps in the SSQ1//MainTest folder.)

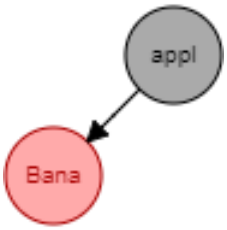
Final step

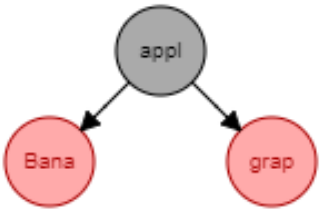


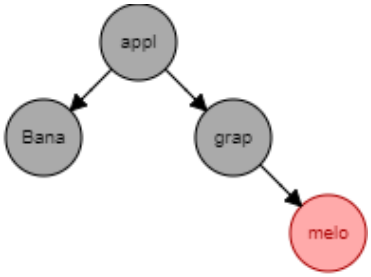
1.2.2 TEST2

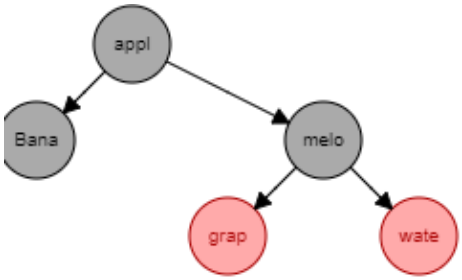
```
@Test
void add() {

    RedBlackTree<String> test=new RedBlackTree<>();
    test.add("apple");
    test.add("Banana");
    test.add("grape");
    test.add("melon");
    test.add("watermelon");
    test.add("orange");
    test.add("mandarin");
    test.add("pineapple");
    test.add("apricot");
    System.out.println(test.toString());
}
```

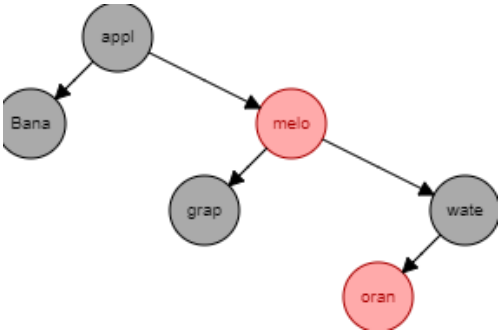
```
Black: apple
  Black: Banana
    null
    null
  Red  : melon
    Black: grape
      Red  : apricot
        null
        null
      Red  : mandarin
        null
        null
    Black: pineapple
      Red  : orange
        null
        null
      Red  : watermelon
        null
        null


Process finished with exit code 0
```

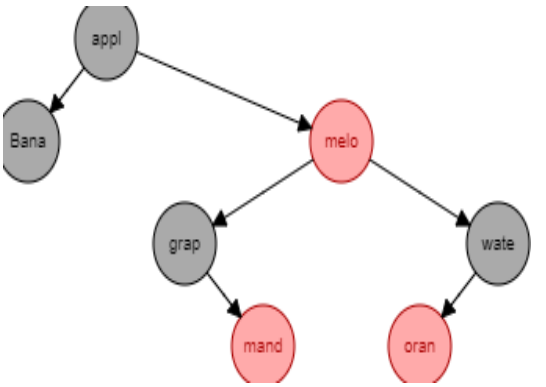


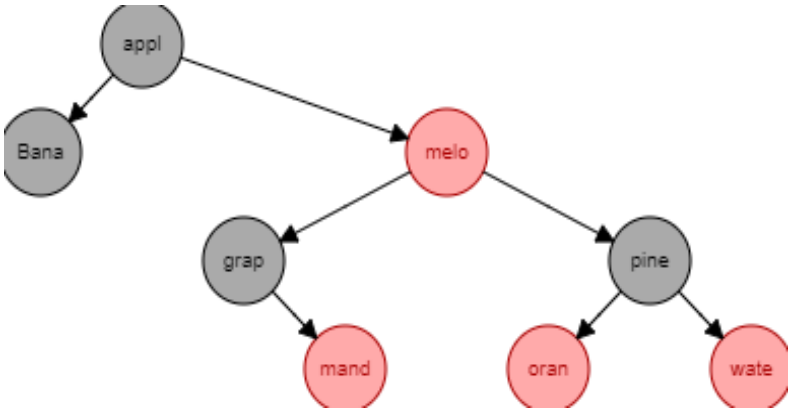


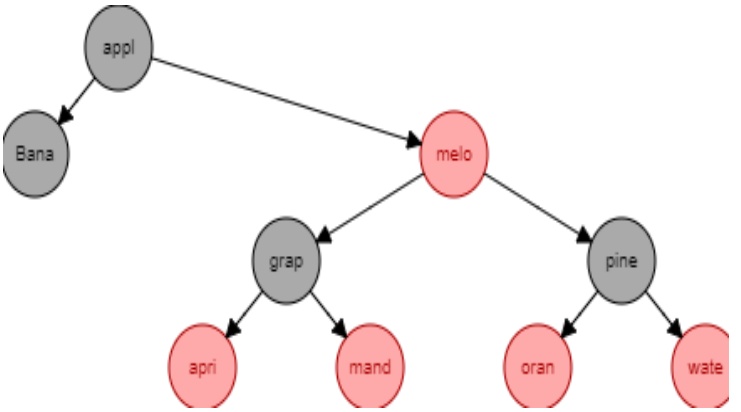












1.3 Running Commands and Results

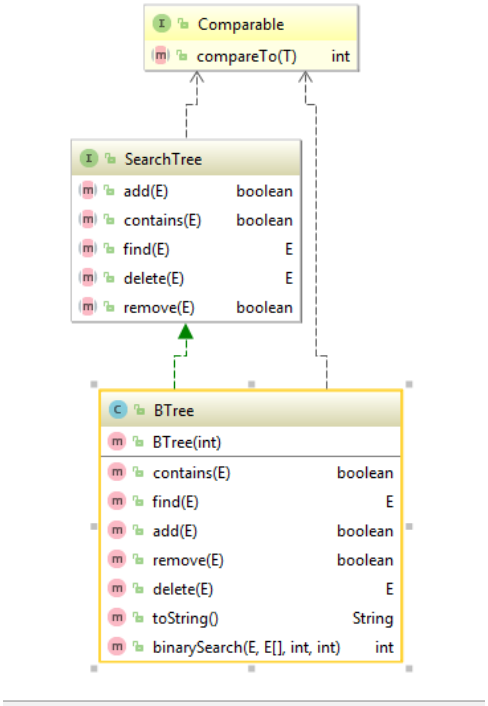
Show that test case results using screenshots.
Running results are also below 1.2 TestCase heading.

2 binarySearch method

This part about Question2 in HW6

2.1 Problem Solution Approach

Write pseudocode and explanation about code design. Indicate what you are using that interfaces, classes, structures, etc.



The BTree class implements the SearcTree interface. BTree strucu

I complete missing part int the BTree class.The binarySearch method does not exits in the source code of the text book.I create this method according to the below algorithm.

BinarySearch method

binarySeach(E target,E[] data, in first, int last)

1. **İf first is equal last**
2. Return first or last
3. **İf last – 1 is 1**
4. **İf target is less than data[first]**
5. Return first /*Add left*/
6. **Else**
7. Return last /*Add right*/
8. Set middle to average of the first and last
9. **if target is equal to data[middle]**
10. Return middle
11. **Else if target is less than data[middle]**
12. Return binarySearch(target,data,first,middle)
13. **Else**
14. Return binarySearch(target,data,middle+1,last)

```
/**Binary search method*/
public int binarySearch(   E target,E [] data, int first, int last) {

    if(first==last)
        return last;
    if (last-first==1)
    {
        if (target.compareTo(data[first])<0)
            return first;
        else
            return last;
    }
    int middle = (first+last)/ 2;
    if (target.compareTo(data[middle])==0)
        return middle;
     else if (target.compareTo(data[middle])<0)
        return binarySearch(target, data, first, middle);
    else
        return binarySearch(target, data,  first: middle + 1, last);
}
```

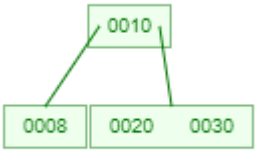
2.2 Test Cases

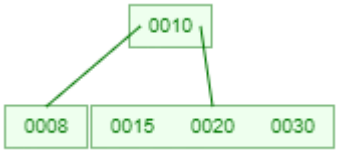
2.2.1 Tree

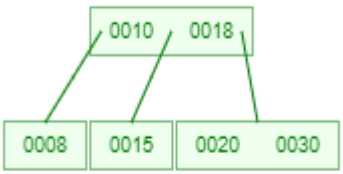
```java
@Test
void add() {
    BTree<Integer> test = new BTree<>( order: 4);
    test.add(20);test.add(30);test.add(8);test.add(10);
    test.add(15);test.add(18);test.add(44);test.add(26);
    test.add(28);test.add(23);test.add(25);test.add(43);
    System.out.print(test.toString());
}
```

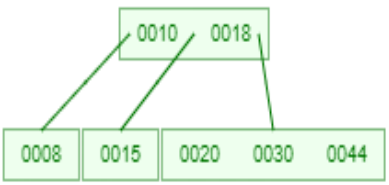
```
C:\Users\sevgi\Do
18
   10
      8
         null
         null

      15
         null
         null


   26, 30
      20, 23, 25
         null
         null
         null
         null

      28
         null
         null

      43, 44
         null
         null
         null
```

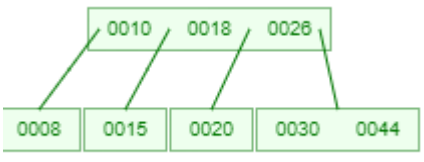
2.3 Running Commands and Results

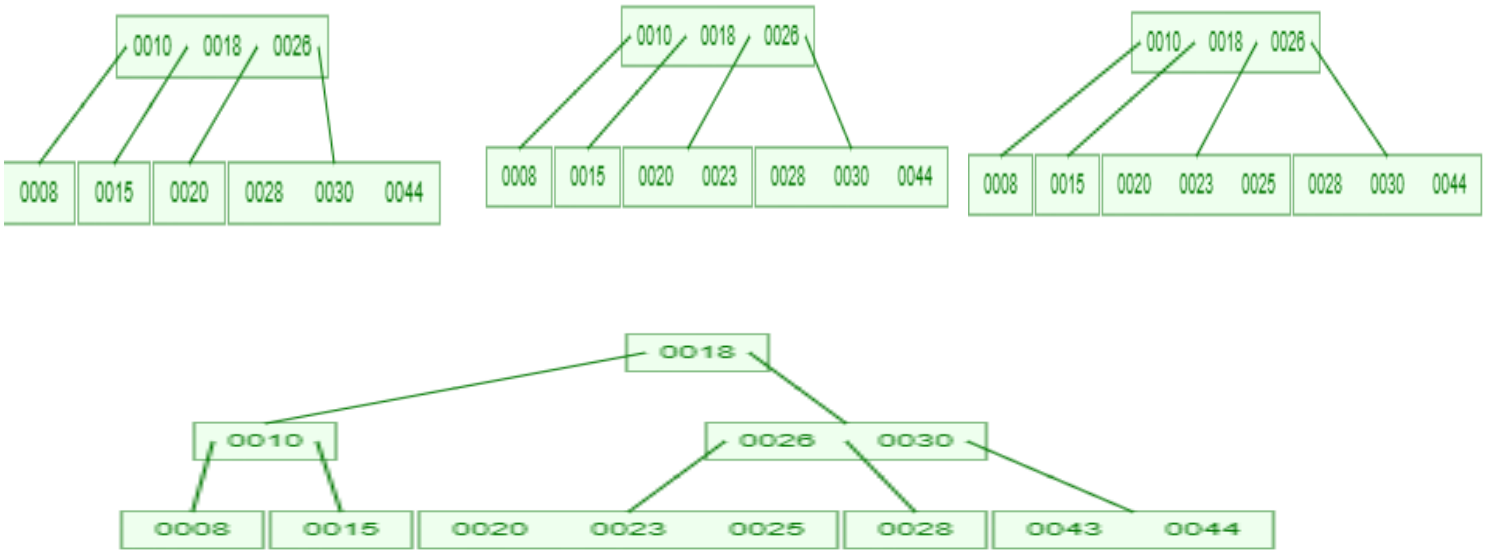












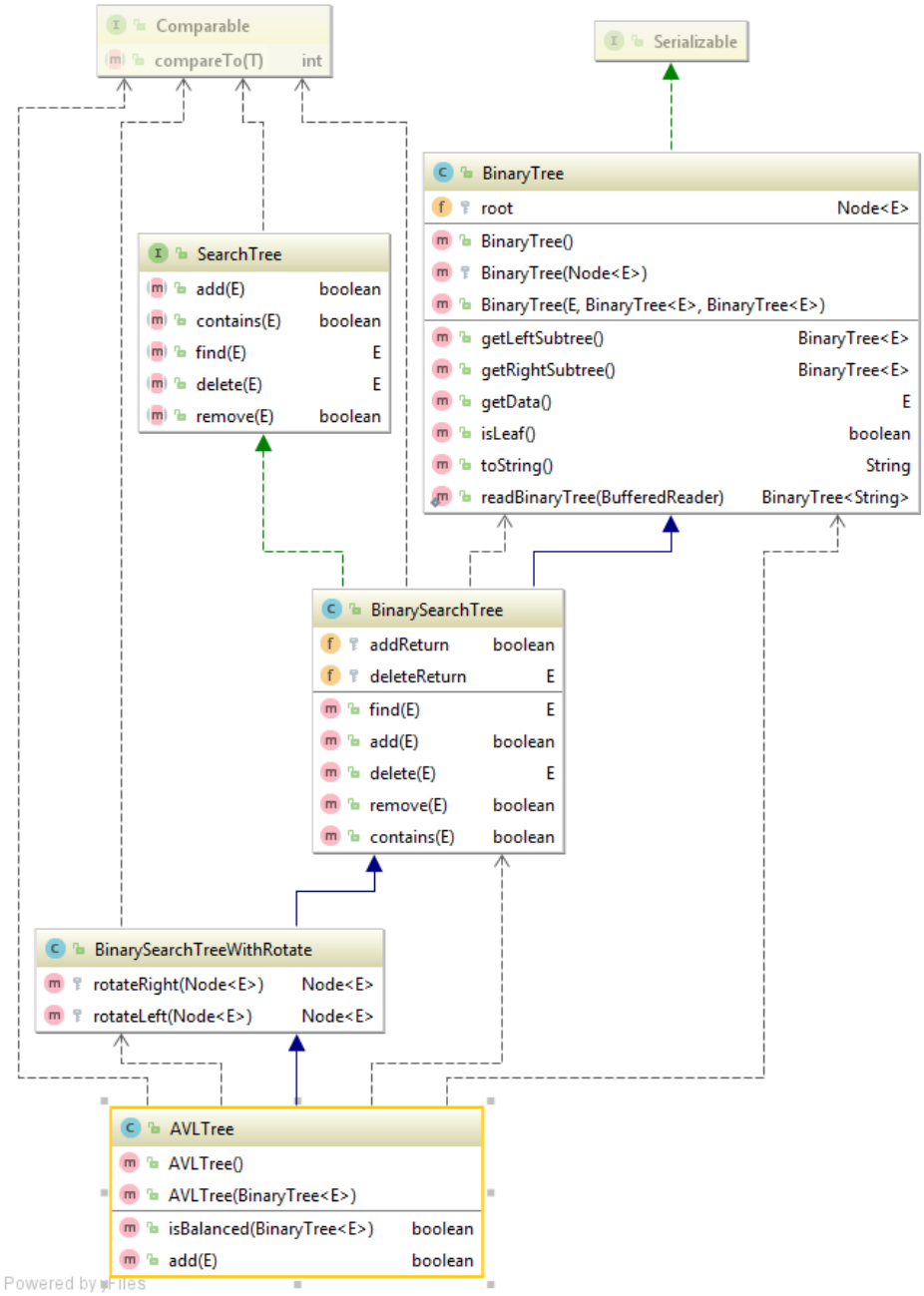
3 Project 9.5 in book

This part about Question3 in HW6

3.1 Problem Solution Approach

In this part I completed the missing part of the add method ; wrote incrementBalance ,rebalanceRight method and constructor for BinaryTree object.

(PS: I did not write the methods for removing!!)



- Constructor

1. İf node is null
2. Return 0
3. Set leftBalance to chechkBalance(left subtrre of node)
4. Set rightBalance to chechkBalance(right subtrre of node)
5. İf right or left balance are minus 1 or subtaction of these two variable is not beween 1 and minus 1
6. Return -1
7. Else
8. Return the result of the gather 1 with the greater of X and Y.

```
public boolean isBalanced(BinaryTree<E> root){
    return checkBalance(root) != -1;
}

private int checkBalance(BinaryTree<E> node){
    if(node == null)
        return 0;
    int lbalance = checkBalance(node.getLeftSubtree());
    int rbalance = checkBalance(node.getRightSubtree());

    int balance=rbalance-lbalance;
    if(rbalance==-1 || lbalance==-1 || balance > 1 || balance<-1)
        return -1;
    else
        return (lbalance > rbalance) ? lbalance+1 : rbalance+1;
}
```

- Add Method

I completed the missing part of the method according to completed part of this method. The add method was incomplete in case item is greater than data.I completed the this part according to case that item is less than data.

- reBalanceRight

1. Set the rightChild to rigthsubtree
2. if the right subtree has a negative balance
3. Set the rightLeftChild to rightChild.left
4. if the rightLeftChild has negative balance
5. Set right subtree balance to +1
6. Set rightLeftChild balance to 0
7. Set the localRoot balance to 0
8. Else if rightLeftChild has positive balance
9. Set right subtree balance to 0
10. Set rightLeftChild balance to 0
11. Set the localRoot balance to -1
12. Else
13. Set right subtree balance to 0
14. Set the localRoot balance to 0
15. Set localRoot.rigt to rotateRight(rightChild)
16. Else
17. Set right subtree balance to 0
18. Set the localRoot balance to 0
19. Return rotateLeft(localRoot)

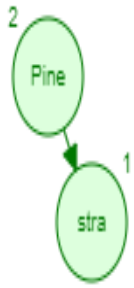
```
private AVLNode<E> rebalanceRight(AVLNode<E> localRoot) {
    // Obtain reference to right child
    AVLNode < E > rightChild = (AVLNode < E > ) localRoot.right;
    // See if right-left heavy
    if (rightChild.balance < AVLNode.BALANCED) {
        // Obtain reference to right-left child
        AVLNode < E > rightLeftChild = (AVLNode < E > ) rightChild.left;
// Adjust the balances to be their new values after
    // the rotations are performed.
        if (rightLeftChild.balance < AVLNode.BALANCED) {
            rightChild.balance = AVLNode.RIGHT_HEAVY;
            rightLeftChild.balance = AVLNode.BALANCED;
            localRoot.balance = AVLNode.BALANCED;
        }else if (rightLeftChild.balance > AVLNode.BALANCED) {
            rightChild.balance = AVLNode.BALANCED;
            rightLeftChild.balance = AVLNode.BALANCED;
            localRoot.balance = AVLNode.LEFT_HEAVY;
        }else {
            rightChild.balance = AVLNode.BALANCED;
            localRoot.balance = AVLNode.BALANCED;
        }
        // Perform right rotation
        localRoot.right = rotateRight(rightChild);
    }
    else {
        rightChild.balance = AVLNode.BALANCED;
        localRoot.balance = AVLNode.BALANCED;
    }
    return (AVLNode < E > ) rotateLeft(localRoot);
```

3.2 Test Cases

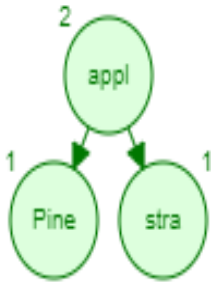
```
@Test
void add() {
    AVLTree<String> tree=new AVLTree<>();
    tree.add("Pineapple");
    tree.add("strawberry");
    tree.add("apple");
    tree.add("banana");
    tree.add("grape");
    tree.add("orange");
    tree.add("pineapple");
    tree.add("mandarin");
    tree.add("cherry");
    System.out.println(tree.toString());
}
```

```
0: grape
  1: apple
    0: Pineapple
      null
      null
    1: banana
      null
      0: cherry
        null
        null
  -1: pineapple
    -1: orange
      0: mandarin
        null
        null
      null
    0: strawberry
      null
      null
```

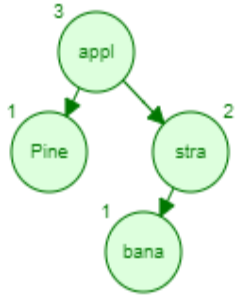
1.



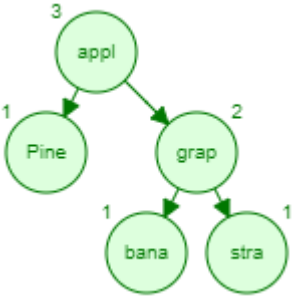
2.



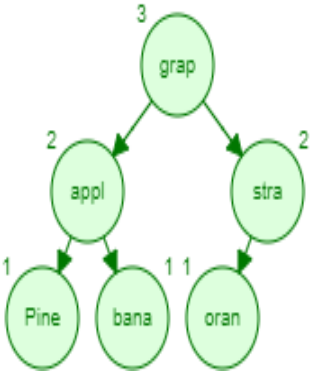
3.



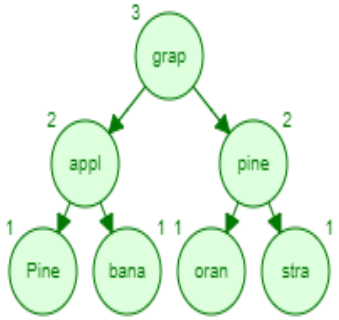
4.



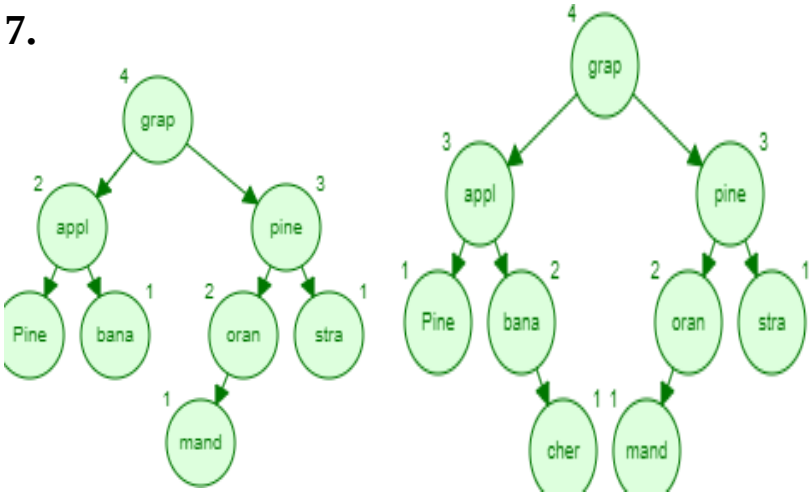
5.



6.



7.



3.3 Running Commands and Results

```
@Test
void is()
{
    BinarySearchTree<Integer> tree=new BinarySearchTree();
    tree.add(10);
    tree.add(15);
    tree.add(25);
    tree.add(20);
    System.out.println(tree);
    AVLTree<Integer> test=new AVLTree<>(tree);

   BinarySearchTree<Integer> tree2=new BinarySearchTree();
   tree2.add(10);
   tree2.add(8);
   tree2.add(25);
   tree2.add(20);
   System.out.println(tree);
   AVLTree<Integer> test2=new AVLTree<>(tree2);
```

```
C:\Users\sevgi\Documents\jdk1.8.0
10
  null
  15
    null
    25
      20
        null
        null
      null

It is not AVL Tree.
10
  null
  15
    null
    25
      20
        null
        null
      null

It is AVL Tree.

Process finished with exit code 0
```