# GTU Department of Computer Engineering
# CSE 222/505 - SPRING 2022
# HOMEWORK 7 REPORT


## MUHAMMET ÇAĞRI YILMAZ

## 1901042694

# SYSTEM REQUIREMENTS

The first question, we need to 5 methods.

We count the all nodes

```java
@SuppressWarnings("unchecked")
int countTotalNodes(BinaryTree.Node<E> root) {
    if (root == null)
        return 0;
    return countTotalNodes(root.leftTree) + countTotalNodes(root.rightTree) + 1;
}
```

Store the nodes

```java
@SuppressWarnings("unchecked")
void storeIndorderTo(BinaryTree.Node<E> node, E[] inorder) {

    if (node == null)
        return;

    storeIndorderTo(node.leftTree, inorder);

    inorder[index] = (E) node.data;
    index++;

    storeIndorderTo(node.rightTree, inorder);
}
```

Array converts to Binary Search Tree

```
@SuppressWarnings("unchecked")
void arrayConvertToBST(E[] arr, BinaryTree.Node<E> root) {
    if (root == null)
        return;

    arrayConvertToBST(arr, root.leftTree);
    root.data = arr[index];
    index++;

    arrayConvertToBST(arr, root.rightTree);
}
```

Print the binary search tree

```
@SuppressWarnings("unchecked")
 void printInorderTraverse(BinaryTree.Node<E> node) {
    if (node == null)
        return;

    printInorderTraverse(node.leftTree);
    System.out.print(" - "+node.data + " ");
    printInorderTraverse(node.rightTree);
}
```

Index holds the index of Array

```
static int index;
BinarySearchTree<E> datas=new BinarySearchTree<>();
```

Question 2

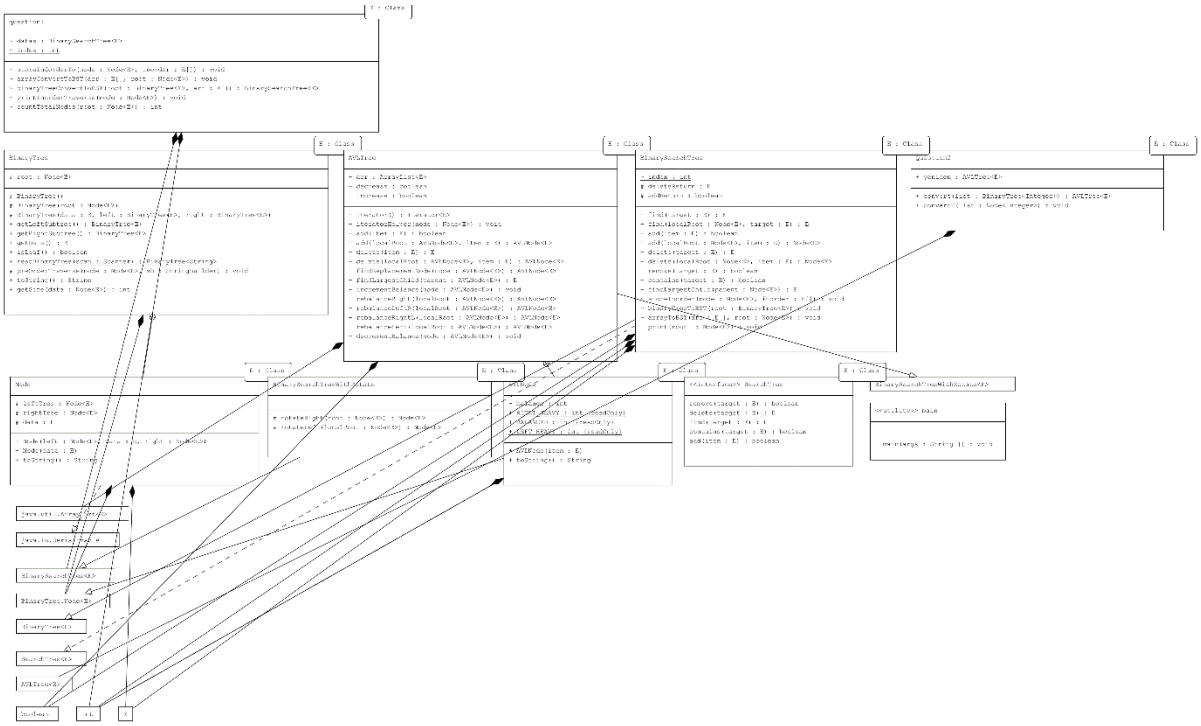
```
public AVLTree<E> avlTree =new AVLTree<>();
/unchecked/
public AVLTree<E> convert(BinaryTree<Integer> list){
  convert2(list.root);
    return avlTree;
}
/unchecked/
public void convert2(BinaryTree.Node<Integer> list){
    if(list==null){
        return;
    }else{
        avlTree.add((E) list.data);
        convert2(list.leftTree);
        convert2(list.rightTree);
    }
}
```

## PROBLEM SOLUTION APPROACH

In the first question first of all, We find total count of tree after that we create an array that size is total count of nodes. Finally we convert to Binary search tree.

Second Question, we add directly to avlTree because in add method we have balanced methods to make sure that whether the tree is balanced or not.

## CLASS DIAGRAM



# TEST CASE AND RESULT

```java
// QUESTION 1
question1 first=new question1();

BinaryTree.Node<Integer> datas2=new BinaryTree.Node<Integer>( data: 21);
BinaryTree.Node<Integer> datas4=new BinaryTree.Node<Integer>( data: 19);
BinaryTree.Node<Integer> datas3=new BinaryTree.Node<Integer>( data: 24);
datas2.rightTree=datas3;
datas2.leftTree=datas4;
BinaryTree.Node<Integer> datas1=new BinaryTree.Node<Integer>( data: 27);
datas1.rightTree= new BinaryTree.Node<Integer>( data: 28);
datas1.leftTree= new BinaryTree.Node<Integer>( data: 26);
BinaryTree firstBinary=new BinaryTree<>(datas2);
BinaryTree secondBinary=new BinaryTree(datas1);
BinaryTree<Integer> bt=new BinaryTree( data: 25,firstBinary,secondBinary);
final int count=first.countTotalNodes(bt.root);
StringBuilder str1=new StringBuilder();
bt.preOrderTraverse(bt.root,str1);
System.out.println(str1.toString());
Integer[] dataArray=new Integer[count];
BinarySearchTree<Integer> bst=first.binaryTreeConvertToBST(bt,dataArray);
first.printInorderTraverse(bst.root);
```

```java
//Question 2
Question2 second=new Question2();
BinaryTree.Node<Integer> node=new BinaryTree.Node<Integer>( data: 35);
node.rightTree=new BinaryTree.Node<Integer>( data: 45);
node.leftTree=new BinaryTree.Node<Integer>( data: 25);
node.rightTree.rightTree=new BinaryTree.Node<Integer>( data: 55);
node.leftTree.leftTree=new BinaryTree.Node<Integer>( data: 20);
node.rightTree.rightTree.rightTree=new BinaryTree.Node<Integer>( data: 65);
node.rightTree.rightTree.rightTree.rightTree=new BinaryTree.Node<Integer>( data: 75);
node.rightTree.rightTree.rightTree.rightTree.rightTree=new BinaryTree.Node<Integer>( data: 85);
BinaryTree<Integer> binaryTreeLeft=new BinaryTree<>(node.leftTree);
BinaryTree<Integer> binaryTreeRight=new BinaryTree<>(node.rightTree);
BinaryTree<Integer> binaryTree=new BinaryTree<Integer>(node.data,binaryTreeLeft,binaryTreeRight);
System.out.println();
AVLTree<Integer> avl=new AVLTree<Integer>();
avl=second.convert(binaryTree);
System.out.println("Question 2 ");
StringBuilder str2=new StringBuilder();
avl.preOrderTraverse(binaryTree.root,str2);
System.out.println(str2.toString());
```

```
25 21 19 24 27 26 28
19 21 24 25 26 27 28
Question 2
35 25 20 45 55 65 75 85
```

It prints the Inorder traversal and there is no data loss. Furthermore, I keep the structure of tree.