GIT Department of Computer Engineering CSE222/505-Spring 2020 Homework 7 Report

Mustafa Gurler

171044034

System Requirements:

Q1:

NavigableAVLTree class:

public boolean add(Object item) { // It takes a item and insert the item inside of the AVL tree

public NavigableSet headSet(Object toElement, boolean inclusive) { // It takes an bound an create an Navigable Set and adds the smaller elements with that bound inside of the instance.

private void preOrderTraverse(AVLTree.Node node){// It adds the elements from left subtree of node for headSet method

public NavigableSet tailSet(Object fromElement, boolean inclusive) { // It takes an bound an create an Navigable Set and adds the bigger elements with that bound inside of the instance.

private void postOrderTraverse(AVLTree.Node node){// It adds the elements from right subtree of node for tailSet method

public Iterator iterator() { // It returns a iterator for AVL Tree to reach to elements with iterating

NavigableSkipList class:

public boolean add(Object item) { // It takes a item and insert the item inside of the skip list

public boolean remove(Object target) { // Removes element from skip list

MainTest:

public static boolean isBalanced(BinaryTree.Node root){ // *checks if the binary search tree is balanced*

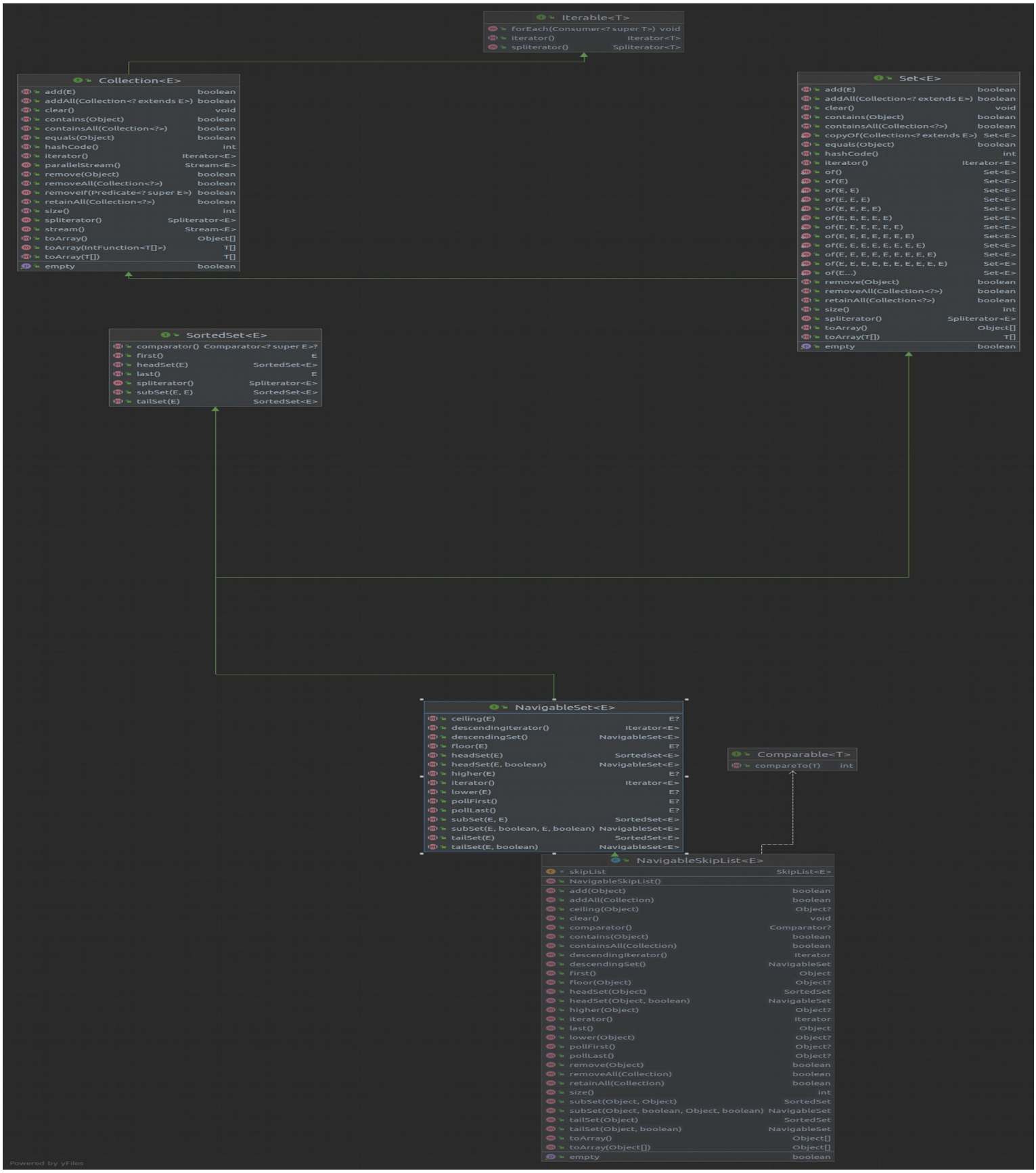
private static int height(BinaryTree.Node root){ // gives the height of current node

Part3 is only driver class for testing running time. It does not have a method that has been used for helping purposes.

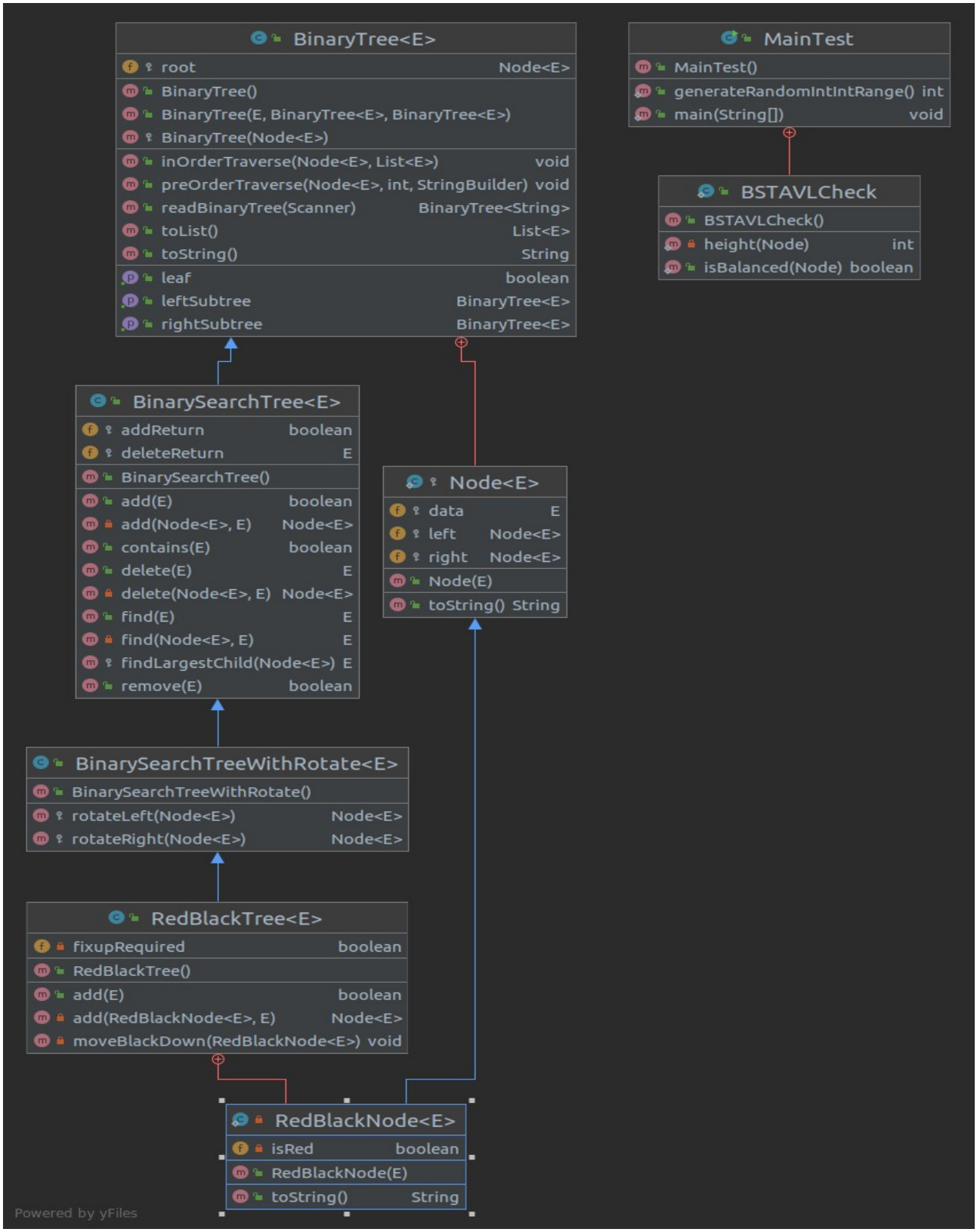
openjdk 11.0.11 2021-04-20
OpenJDK Runtime Environment (build 11.0.11+9-Ubuntu-0ubuntu2.18.04)
OpenJDK 64-Bit Server VM (build 11.0.11+9-Ubuntu-0ubuntu2.18.04, mixed mode, sharing)

Class Diagrams:

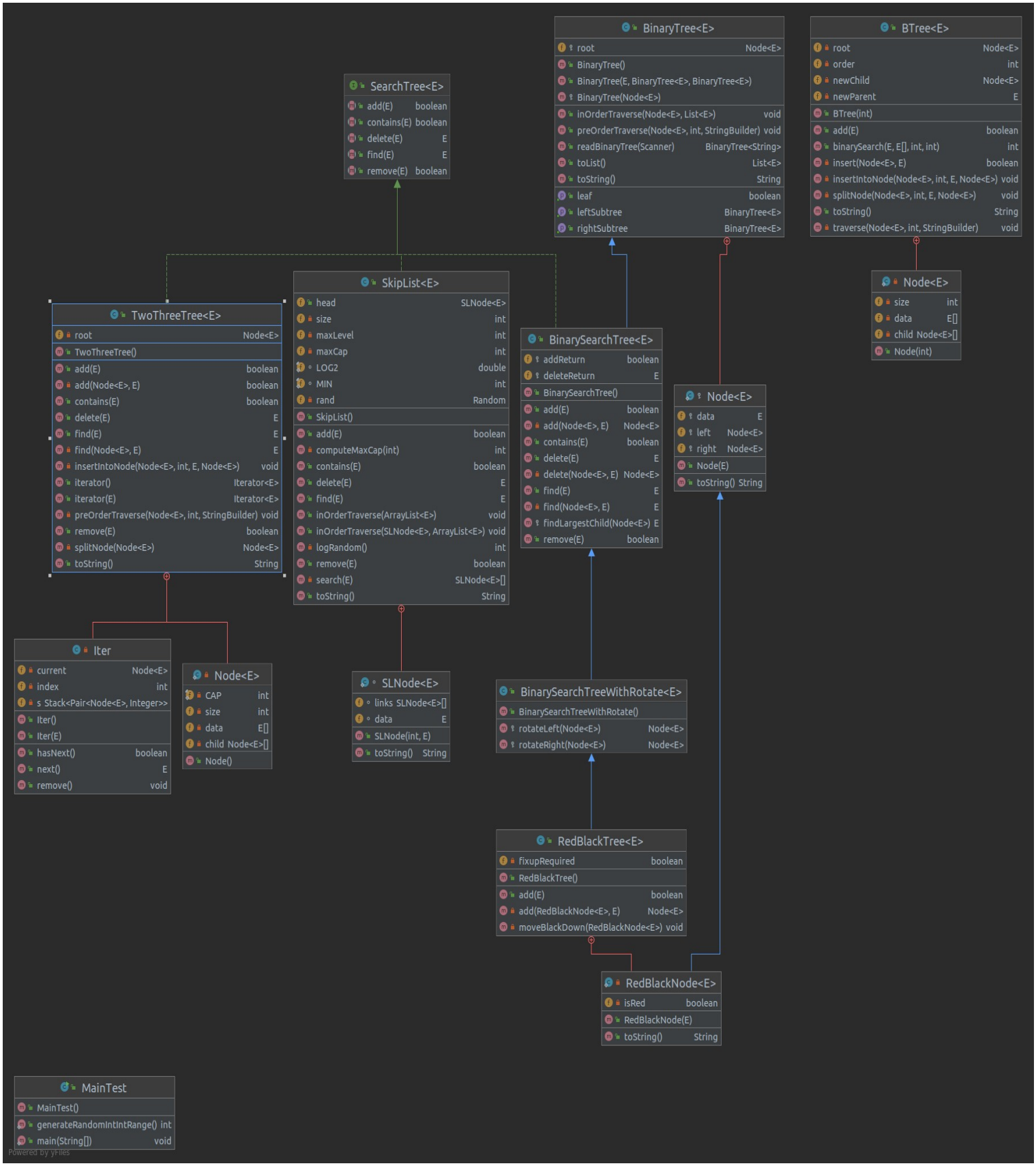
Q1:



Q2:


BinaryTree<E>
root Node<E>
BinaryTree()
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
BinaryTree(Node<E>)
inOrderTraverse(Node<E>, List<E>) void
preOrderTraverse(Node<E>, int, StringBuilder) void
readBinaryTree(Scanner) BinaryTree<String>
toList() List<E>
toString() String
leaf boolean
leftSubtree BinaryTree<E>
rightSubtree BinaryTree<E>
MainTest
MainTest()
generateRandomIntIntRange() int
main(String[]) void
BSTAVLCheck
BSTAVLCheck()
height(Node) int
isBalanced(Node) boolean
BinarySearchTree<E>
addReturn boolean
deleteReturn E
BinarySearchTree()
add(E) boolean
add(Node<E>, E) Node<E>
contains(E) boolean
delete(E) E
delete(Node<E>, E) Node<E>
find(E) E
find(Node<E>, E) E
findLargestChild(Node<E>) E
remove(E) boolean
Node<E>
data E
left Node<E>
right Node<E>
Node(E)
toString() String
BinarySearchTreeWithRotate<E>
BinarySearchTreeWithRotate()
rotateLeft(Node<E>) Node<E>
rotateRight(Node<E>) Node<E>
RedBlackTree<E>
fixupRequired boolean
RedBlackTree()
add(E) boolean
add(RedBlackNode<E>, E) Node<E>
moveBlackDown(RedBlackNode<E>) void
RedBlackNode<E>
isRed boolean
RedBlackNode(E)
toString() String
Powered by yFiles

Q3:


SearchTree<E>
add(E) boolean
contains(E) boolean
delete(E) E
find(E) E
remove(E) boolean
BinaryTree<E>
root Node<E>
BinaryTree()
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
BinaryTree(Node<E>)
inOrderTraverse(Node<E>, List<E>) void
preOrderTraverse(Node<E>, int, StringBuilder) void
readBinaryTree(Scanner) BinaryTree<String>
toList() List<E>
toString() String
leaf boolean
leftSubtree BinaryTree<E>
rightSubtree BinaryTree<E>
BTree<E>
root Node<E>
order int
newChild Node<E>
newParent E
BTree(int)
add(E) boolean
binarySearch(E, E[], int, int) int
insert(Node<E>, E) boolean
insertIntoNode(Node<E>, int, E, Node<E>) void
splitNode(Node<E>, int, E, Node<E>) void
toString() String
traverse(Node<E>, int, StringBuilder) void
Node<E>
size int
data E[]
child Node<E>[]
Node(int)
SkipList<E>
head SLNode<E>
size int
maxLevel int
maxCap int
LOG2 double
MIN int
rand Random
SkipList()
add(E) boolean
computeMaxCap(int) int
contains(E) boolean
delete(E) E
find(E) E
inOrderTraverse(ArrayList<E>) void
inOrderTraverse(SLNode<E>, ArrayList<E>) void
logRandom() int
remove(E) boolean
search(E) SLNode<E>[]
toString() String
TwoThreeTree<E>
root Node<E>
TwoThreeTree()
add(E) boolean
add(Node<E>, E) boolean
contains(E) boolean
delete(E) E
find(E) E
find(Node<E>, E) E
insertIntoNode(Node<E>, int, E, Node<E>) void
iterator() Iterator<E>
iterator(E) Iterator<E>
preOrderTraverse(Node<E>, int, StringBuilder) void
remove(E) boolean
splitNode(Node<E>) Node<E>
toString() String
BinarySearchTree<E>
addReturn boolean
deleteReturn E
BinarySearchTree()
add(E) boolean
add(Node<E>, E) Node<E>
contains(E) boolean
delete(E) E
delete(Node<E>, E) Node<E>
find(E) E
find(Node<E>, E) E
findLargestChild(Node<E>) E
remove(E) boolean
Node<E>
data E
left Node<E>
right Node<E>
Node(E)
toString() String
Iter
current Node<E>
index int
s Stack<Pair<Node<E>, Integer>>
Iter()
Iter(E)
hasNext() boolean
next() E
remove() void
Node<E>
CAP int
size int
data E[]
child Node<E>[]
Node()
SLNode<E>
links SLNode<E>[]
data E
SLNode(int, E)
toString() String
BinarySearchTreeWithRotate<E>
BinarySearchTreeWithRotate()
rotateLeft(Node<E>) Node<E>
rotateRight(Node<E>) Node<E>
RedBlackTree<E>
fixupRequired boolean
RedBlackTree()
add(E) boolean
add(RedBlackNode<E>, E) Node<E>
moveBlackDown(RedBlackNode<E>) void
RedBlackNode<E>
isRed boolean
RedBlackNode(E)
toString() String
MainTest
MainTest()
generateRandomIntIntRange() int
main(String[]) void
Powered by yFiles

Problem Solution Approach:

Q1:

I created an AVL Tree inside of my Navigable AVLTree. Inside of add method I add the elements inside of avl tree. That way I filled the AVL Tree with 100 different number.

For Iterator method I created a class which implements Iterator class. Methods have been overridden. I kept the elements inside of a stack. Stack is efficient like recursive function for trees. It returns the elements one by one to user. I used HasNext method to check if stack is empty.

HeadSet and TailSet are opposite of each other. I need to insert to elements for these any other navigable AVL Tree. All the left elements from given element inserted that navigable AVL Tree. I returned new navigable AVLTree. With iterator method I controlled all the smaller elements inside of nev navigable AVLTree . I tried to approach to tailSet with same algorithm but opposite way. It gives me the correct result.

Also I created an Skip List inside of my NavigableSkipList. I used to same approach with AVL Tree for add and remove method inside of skip list. I checked with 100 different number and showed them with 2 different numbers.

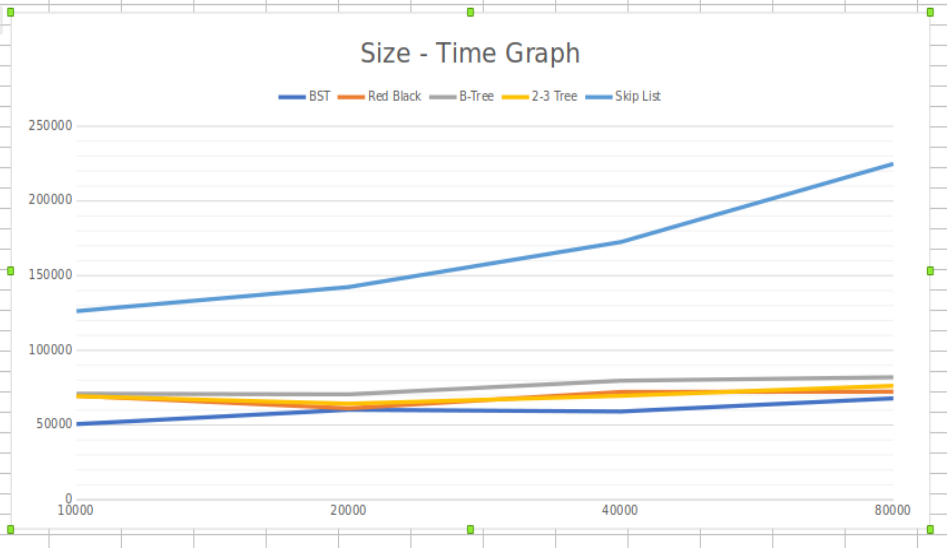
Q2:

I only did AVLTree part. AVLTree is a balanced tree , so I need to check the binary search tree is a balanced tree or not. We know binary search tree could be imbalanced with a bigger size. So I added 100 elements for testing. It gives me a non AVL Tree but when I add 5-6 element like balanced AVL Tree, my method shows the binary search tree is an AVL Tree.

Q3:

I created 10 instances for every test structure. I added expected elements to all the data structures. (10000, 20000, 40000, 80000). Then I added extrea 100 elements all the 200 instances. Every test 10 times have been tested and we got the average result. They were very close as expected.

	10000	20000	40000	80000
BST	50633	60240	59077	67883
Red Black	69618	61206	72186	72344
B-Tree	70952	70601	79711	82005
2-3 Tree	69340	64308	69596	76295
Skip List	126267	142400	172557	224850



BST was very fast for inserting but it was going worse because after 40000 elements , the balance of BST goes to an unbalanced tree. But Balanced trees run as logn every time in a perfectly shape. Worst case and base case was same. If we draw a line for skip list, from 0 to end. We can see line is always going to be above of the skip list.

Test Cases:

Q1:

Test Case #	Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Adding elements to skip list	NavigableSkipList.add(1000)	Added element succesful	Added element succesful	Pass
2	Removing elements from skip list	NavigableSkipList.remove(1000)	Removed element succesful	Removed element succesful	Pass
3	Added 100 elements to navigable AVL tree and iterated	NavigableAVLTree.iterator();	100 number found inside of iterator	100 number found inside of iterator	Pass
4	Added 100 elements to navigable AVL tree and finds the numbers smaller than 2500 (headSet)	NavigableAVLTree.headSet(2500, true);	All the smaller number has been found	All the smaller number has been found	Pass
5	Added 100 elements to navigable AVL tree and finds the numbers bigger than 2500 (tailSet)	NavigableAVLTree.tailSet(2500, true);	All the bigger numbers have been found	All the bigger numbers have been found	Pass
6	Added 2 different number have been added to AVL tree and checked if it is succesful	NavigableAVLTree.add(1000);	Added element succesful	Added element succesful	Pass

Q2:

Test Case #	Description	Test Data	Expected Result	Actual Resut	Pass/Fail
1	Fill the binary search tree with random numbers and check if it is a AVL Tree	Test.add(generateRandomNumber);	It is not an AVL Tree	It is not an AVL Tree	Pass
2	Fill the binary search tree with numbers that is going to be balanced binary search tree and check if it is a AVL Tree	Test2.add(numbers);	It is an AVL Tree	It is an AVL Tree	Pass

Q3:

Test Case #	Description	Test Data	Expected Result	Actual Result	Pass/Fail
1	Added 100 elements to 10 different BinarySearchTree instance which was 10000-20000-40000-80000 elements added before and Running time has been measured	binarySearchTree1[i].add(generateRandom());	Numbers are added and time has been showed	Numbers are added and time has been showed	Pass
2	Added 100 elements to 10 different RedBlackTree instance which was 10000-20000-40000-8000 elements added before and Running time has been measured	redBlackTree[i].add(generateRandom());	Numbers are added and time has been showed	Numbers are added and time has been showed	Pass
3	Added 100 elements to 10 different BinaryTree instance which was 10000-20000-40000-80000 element added before and Running time has been measured	binaryTree[i].add(generateRandom());	Numbers are added and time has been showed	Numbers are added and time has been showed	Pass
4	Added 100 elements to 10 different TwoThreeTree instance which was 10000-20000-40000-80000 element added before and Running time has been measured	twoThreeTree[i].add(generateRandom());	Numbers are added and time has been showed	Numbers are added and time has been showed	Pass
5	Added 100 elements to 10 different SkipList instance which was 10000-20000-40000-80000 element added before and Running time has been measured	skipList[i].add(generateRandom());	Numbers are added and time has been showed	Numbers are added and time has been showed	Pass

Running and Results

Running:

```
System.out.println("Adding elements to Navigable Skip list");
if(navigableSkipList.add(1000)){
	System.out.println("Added element successful");
}else{
	System.out.println("Added element unsuccessful");
}
if(navigableSkipList.add(2000)){
	System.out.println("Added element successful");
}else{
	System.out.println("Added element unsuccessful");
}
```

Result:

****

Adding elements to Navigable Skip list
Added element successful
Added element successful

Running:

```
System.out.println("**********");
System.out.println("Removing elements to Navigable Skip list");
if(navigableSkipList.remove(1000)){
	System.out.println("Remove element successful");
}else{
	System.out.println("Remove element unsuccessful");
}
if(navigableSkipList.remove(2000)){
	System.out.println("Remove element successful");
}else{
	System.out.println("Remove element unsuccessful");
}
```

Results:

****

Removing elements to Navigable Skip list
Remove element successful
Remove element successful

Running:

```
System.out.println("**********");
System.out.println("Iterator for Navigable AVLTree");
Iterator it = navigableAVLTree.iterator();
int data;
int counter = 0;
while(it.hasNext()){
	data = (Integer)it.next();
	for(int i=0 ; i<100 ; i++){
		if(data == number[i]){
			counter++;
```

```
            }
      }
}
if(counter != 100){
      System.out.println("Error =>ALl the number could not iterated!!!!");
      throw new Exception();
}
```

Result:

Iterator for Navigable AVLTree

100 number found

Running

```
System.out.println("**********************");
System.out.println("Navigable set headset");
NavigableAVLTree<Integer> avlTreeHeadSet = (NavigableAVLTree)
navigableAVLTree.headSet(2500, true);
Iterator it1 = avlTreeHeadSet.iterator();
while(it1.hasNext()){
      if((Integer)it1.next() > 2500){
            System.out.println("HeadSet does not run correctly");
            throw new Exception();
      }
}
System.out.println("Navigable Set HeadSet did not give any error");
```

Result:

Navigable set headset

Navigable Set HeadSet did not give any error

```
System.out.println("**********************");
System.out.println("Navigable set tailSet");
NavigableAVLTree<Integer> avlTreeTailSet = (NavigableAVLTree<Integer>)
navigableAVLTree.tailSet(2500, true);
Iterator it2 = avlTreeTailSet.iterator();
while(it2.hasNext()){
      if((Integer)it2.next() < 2500){
            System.out.println("TailSet does not run correctly");
            throw new Exception();
      }
}
System.out.println("Navigable Set TailSet did not give any error");
```

Result:

Navigable set tailSet

Navigable Set TailSet did not give any error

Running:

```
System.out.println("**********************************");
System.out.println("Adding elements to Navigable AVL Tree");
if(navigableAVLTree.add(1000)){
	System.out.println("Added element successful");
}else{
	System.out.println("Added element unsuccessful");
}
if(navigableAVLTree.add(1000)){
	System.out.println("Added element successful");
}else{
	System.out.println("Added element unsuccessful");
}
if(navigableAVLTree.add(2000)){
	System.out.println("Added element successful");
}else{
	System.out.println("Added element unsuccessful");
}
```

Result:

Adding elements to Navigable AVL Tree

Added element successful

Added element unsuccessful

Added element successful

Q2: AVLTree

Running:

```
BinarySearchTree<Integer> test = new BinarySearchTree<>();
for(int i=0 ; i<100 ; i++){
	test.add(generateRandomIntIntRange());
}
int max = 75;
int min = 25;
int minus = 1;
int maximus = 1;
BinarySearchTree<Integer> test1 = new BinarySearchTree<>();
test1.add(50);
test1.add(min);
test1.add(max);
for(int i=0 ; i<25 ; i++){
		test1.add(min-minus);
		test1.add(max-minus);
		minus++;
		test1.add(min+maximus);
		test1.add(max+maximus);
		maximus++;
}
BinarySearchTree<Integer> t2 = new BinarySearchTree<>();
t2.add(15);
t2.add(5);
t2.add(20);
t2.add(18);
t2.add(23);
t2.add(4);
t2.add(6);
if(BSTAVLCheck.isBalanced(test.root)){
	System.out.println("It is an AVL tree");
}else{
	System.out.println("It is not an AVL tree");
}
if(BSTAVLCheck.isBalanced(test1.root)){
	System.out.println("It is an AVL tree");
}else{
	System.out.println("It is not an AVL tree");
}
if(BSTAVLCheck.isBalanced(t2.root)){
	System.out.println("It is an AVL tree");
}else{
	System.out.println("It is not an AVL tree");
}
```

Results:

It is not an AVL tree

It is not an AVL tree

It is an AVL tree

Q3:

****

Adding 100 elements to 10000 added structures

Binary Search Tree => 50633

Red Black Tree => 69618

TwoThree Tree => 69340

Binary Tree => 70952

Skip List => 126267

****

Adding 100 elements to 20000 added structures

Binary Search Tree => 60240

Red Black Tree => 61206

TwoThree Tree => 64308

Binary Tree => 70601

Skip List => 142400

****

Adding 100 elements to 40000 added structures

Binary Search Tree => 59077

Red Black Tree => 72186

TwoThree Tree => 69596

Binary Tree => 79711

Skip List => 172557

****

Adding 100 elements to 80000 added structures

Binary Search Tree => 67883

Red Black Tree => 72344

TwoThree Tree => 76295

Binary Tree => 82005

Skip List => 224850