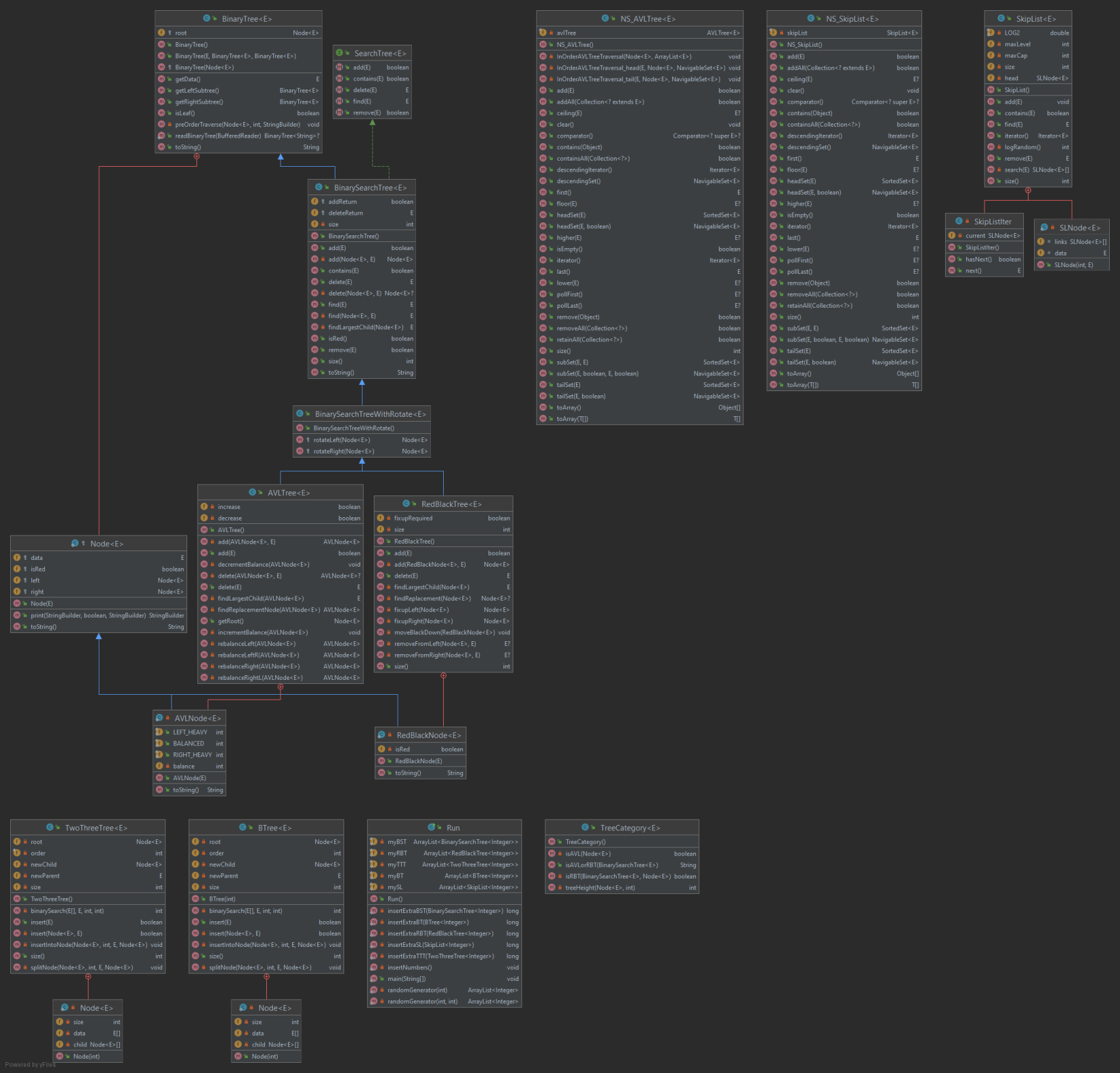
BinaryTree<E>
root Node<E>
BinaryTree()
BinaryTree(E, BinaryTree<E>, BinaryTree<E>)
BinaryTree(Node<E>)
getData() E
getLeftSubtree() BinaryTree<E>
getRightSubtree() BinaryTree<E>
isLeaf() boolean
preOrderTraverse(Node<E>, int, StringBuilder) void
readBinaryTree(BufferedReader) BinaryTree<String>?
toString() String
SearchTree<E>
add(E) boolean
contains(E) boolean
delete(E) E
find(E) E
remove(E) boolean
BinarySearchTree<E>
addReturn boolean
deleteReturn E
size int
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
isRed() boolean
remove(E) boolean
size() int
toString() String
BinarySearchTreeWithRotate<E>
BinarySearchTreeWithRotate()
rotateLeft(Node<E>) Node<E>
rotateRight(Node<E>) Node<E>
AVLTree<E>
increase boolean
decrease boolean
AVLTree()
add(AVLNode<E>, E) AVLNode<E>
add(E) boolean
decrementBalance(AVLNode<E>) void
delete(AVLNode<E>, E) AVLNode<E>?
delete(E) E
findLargestChild(AVLNode<E>) E
findReplacementNode(AVLNode<E>) AVLNode<E>
getRoot() Node<E>
incrementBalance(AVLNode<E>) void
rebalanceLeft(AVLNode<E>) AVLNode<E>
rebalanceLeftR(AVLNode<E>) AVLNode<E>
rebalanceRight(AVLNode<E>) AVLNode<E>
rebalanceRightL(AVLNode<E>) AVLNode<E>
RedBlackTree<E>
fixupRequired boolean
size int
RedBlackTree()
add(E) boolean
add(RedBlackNode<E>, E) Node<E>
delete(E) E
findLargestChild(Node<E>) E
findReplacement(Node<E>) Node<E>?
fixupLeft(Node<E>) Node<E>
fixupRight(Node<E>) Node<E>
moveBlackDown(RedBlackNode<E>) void
removeFromLeft(Node<E>, E) E?
removeFromRight(Node<E>, E) E?
size() int
Node<E>
data E
isRed boolean
left Node<E>
right Node<E>
Node(E)
print(StringBuilder, boolean, StringBuilder) StringBuilder
toString() String
AVLNode<E>
LEFT_HEAVY int
BALANCED int
RIGHT_HEAVY int
balance int
AVLNode(E)
toString() String
RedBlackNode<E>
isRed boolean
RedBlackNode(E)
toString() String
NS_AVLTree<E>
avlTree AVLTree<E>
NS_AVLTree()
inOrderAVLTreeTraversal(Node<E>, ArrayList<E>) void
inOrderAVLTreeTraversal_head(E, Node<E>, NavigableSet<E>) void
inOrderAVLTreeTraversal_tail(E, Node<E>, NavigableSet<E>) void
add(E) boolean
addAll(Collection<? extends E>) boolean
ceiling(E) E?
clear() void
comparator() Comparator<? super E>?
contains(Object) boolean
containsAll(Collection<?>) boolean
descendingIterator() Iterator<E>
descendingSet() NavigableSet<E>
first() E
floor(E) E?
headSet(E) SortedSet<E>
headSet(E, boolean) NavigableSet<E>
higher(E) E?
isEmpty() boolean
iterator() Iterator<E>
last() E
lower(E) E?
pollFirst() E?
pollLast() E?
remove(Object) boolean
removeAll(Collection<?>) boolean
retainAll(Collection<?>) boolean
size() int
subSet(E, E) SortedSet<E>
subSet(E, boolean, E, boolean) NavigableSet<E>
tailSet(E) SortedSet<E>
tailSet(E, boolean) NavigableSet<E>
toArray() Object[]
toArray(T[]) T[]
NS_SkipList<E>
skipList SkipList<E>
NS_SkipList()
add(E) boolean
addAll(Collection<? extends E>) boolean
ceiling(E) E?
clear() void
comparator() Comparator<? super E>?
contains(Object) boolean
containsAll(Collection<?>) boolean
descendingIterator() Iterator<E>
descendingSet() NavigableSet<E>
first() E
floor(E) E?
headSet(E) SortedSet<E>
headSet(E, boolean) NavigableSet<E>
higher(E) E?
isEmpty() boolean
iterator() Iterator<E>
last() E
lower(E) E?
pollFirst() E?
pollLast() E?
remove(Object) boolean
removeAll(Collection<?>) boolean
retainAll(Collection<?>) boolean
size() int
subSet(E, E) SortedSet<E>
subSet(E, boolean, E, boolean) NavigableSet<E>
tailSet(E) SortedSet<E>
tailSet(E, boolean) NavigableSet<E>
toArray() Object[]
toArray(T[]) T[]
SkipList<E>
LOG2 double
maxLevel int
maxCap int
size int
head SLNode<E>
SkipList()
add(E) void
contains(E) boolean
find(E) E
iterator() Iterator<E>
logRandom() int
remove(E) E
search(E) SLNode<E>[]
size() int
SkipListIter
current SLNode<E>
SkipListIter()
hasNext() boolean
next() E
SLNode<E>
links SLNode<E>[]
data E
SLNode(int, E)
TwoThreeTree<E>
root Node<E>
order int
newChild Node<E>
newParent E
size int
TwoThreeTree()
binarySearch(E[], E, int, int) int
insert(E) boolean
insert(Node<E>, E) boolean
insertIntoNode(Node<E>, int, E, Node<E>) void
size() int
splitNode(Node<E>, int, E, Node<E>) void
BTree<E>
root Node<E>
order int
newChild Node<E>
newParent E
size int
BTree(int)
binarySearch(E[], E, int, int) int
insert(E) boolean
insert(Node<E>, E) boolean
insertIntoNode(Node<E>, int, E, Node<E>) void
size() int
splitNode(Node<E>, int, E, Node<E>) void
Run
myBST ArrayList<BinarySearchTree<Integer>>
myRBT ArrayList<RedBlackTree<Integer>>
myTTT ArrayList<TwoThreeTree<Integer>>
myBT ArrayList<BTree<Integer>>
mySL ArrayList<SkipList<Integer>>
Run()
insertExtraBST(BinarySearchTree<Integer>) long
insertExtraBT(BTree<Integer>) long
insertExtraRBT(RedBlackTree<Integer>) long
insertExtraSL(SkipList<Integer>) long
insertExtraTTT(TwoThreeTree<Integer>) long
insertNumbers() void
main(String[]) void
randomGenerator(int) ArrayList<Integer>
randomGenerator(int, int) ArrayList<Integer>
TreeCategory<E>
TreeCategory()
isAVL(Node<E>) boolean
isAVLorRBT(BinarySearchTree<E>) String
isRBT(BinarySearchTree<E>, Node<E>) boolean
treeHeight(Node<E>, int) int
Node<E>
size int
data E[]
child Node<E>[]
Node(int)
Node<E>
size int
data E[]
child Node<E>[]
Node(int)