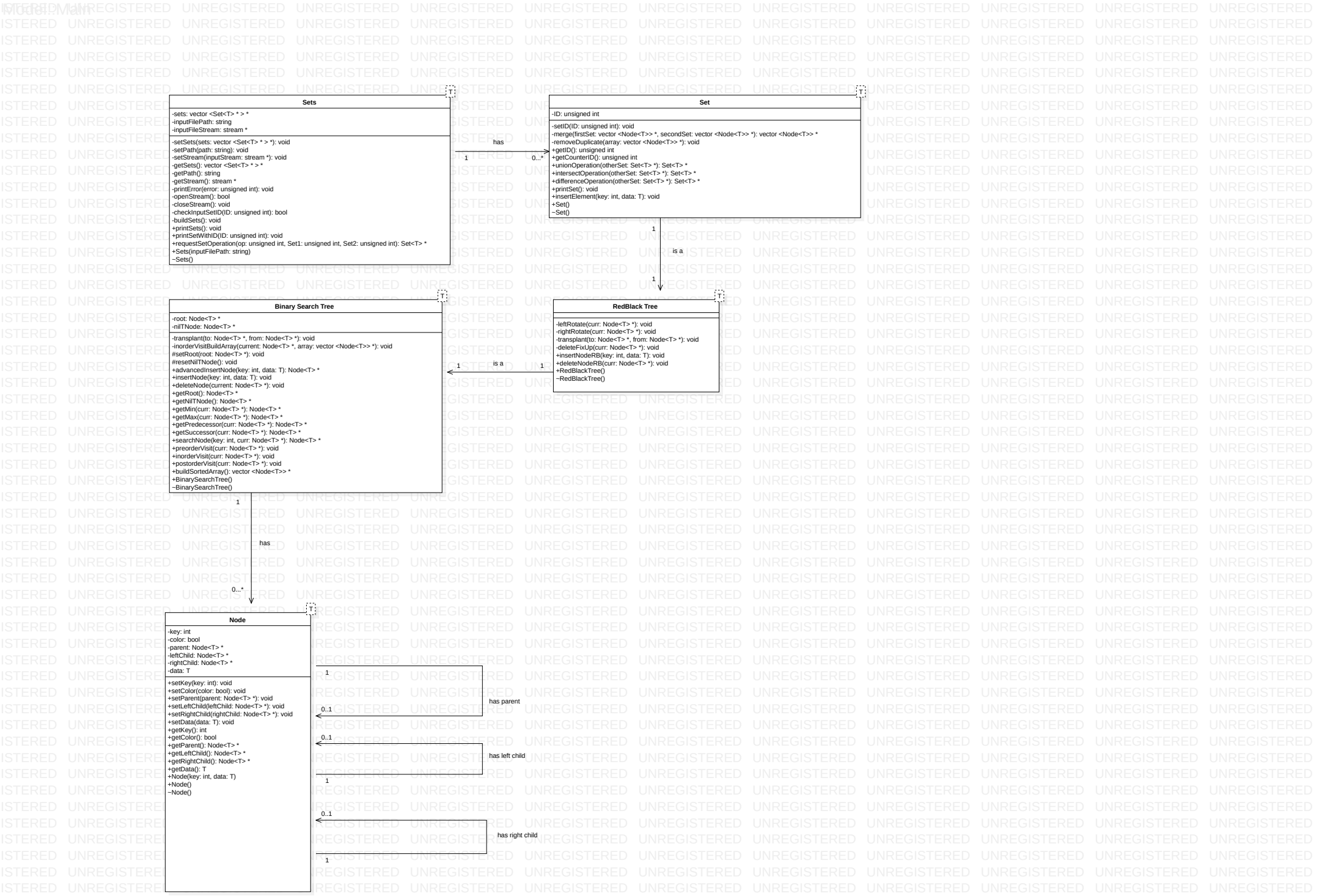
Sets
-sets: vector <Set<T> * > *
-inputFilePath: string
-inputFileStream: stream *
-setSets(sets: vector <Set<T> * > *): void
-setPath(path: string): void
-setStream(inputStream: stream *): void
-getSets(): vector <Set<T> * > *
-getPath(): string
-getStream(): stream *
-printError(error: unsigned int): void
-openStream(): bool
-closeStream(): void
-checkInputSetID(ID: unsigned int): bool
-buildSets(): void
+printSets(): void
+printSetWithID(ID: unsigned int): void
+requestSetOperation(op: unsigned int, Set1: unsigned int, Set2: unsigned int): Set<T> *
+Sets(inputFilePath: string)
~Sets()
T
has
1
0...*
Set
-ID: unsigned int
-setID(ID: unsigned int): void
-merge(firstSet: vector <Node<T>> *, secondSet: vector <Node<T>> *): vector <Node<T>> *
-removeDuplicate(array: vector <Node<T>> *): void
+getID(): unsigned int
+getCounterID(): unsigned int
+unionOperation(otherSet: Set<T> *): Set<T> *
+intersectOperation(otherSet: Set<T> *): Set<T> *
+differenceOperation(otherSet: Set<T> *): Set<T> *
+printSet(): void
+insertElement(key: int, data: T): void
+Set()
~Set()
T
1
is a
1
Binary Search Tree
-root: Node<T> *
-nilTNode: Node<T> *
-transplant(to: Node<T> *, from: Node<T> *): void
-inorderVisitBuildArray(current: Node<T> *, array: vector <Node<T>> *): void
#setRoot(root: Node<T> *): void
#resetNilTNode(): void
+advancedInsertNode(key: int, data: T): Node<T> *
+insertNode(key: int, data: T): void
+deleteNode(current: Node<T> *): void
+getRoot(): Node<T> *
+getNilTNode(): Node<T> *
+getMin(curr: Node<T> *): Node<T> *
+getMax(curr: Node<T> *): Node<T> *
+getPredecessor(curr: Node<T> *): Node<T> *
+getSuccessor(curr: Node<T> *): Node<T> *
+searchNode(key: int, curr: Node<T> *): Node<T> *
+preorderVisit(curr: Node<T> *): void
+inorderVisit(curr: Node<T> *): void
+postorderVisit(curr: Node<T> *): void
+buildSortedArray(): vector <Node<T>> *
+BinarySearchTree()
~BinarySearchTree()
T
1
is a
1
RedBlack Tree
-leftRotate(curr: Node<T> *): void
-rightRotate(curr: Node<T> *): void
-transplant(to: Node<T> *, from: Node<T> *): void
-deleteFixUp(curr: Node<T> *): void
+insertNodeRB(key: int, data: T): void
+deleteNodeRB(curr: Node<T> *): void
+RedBlackTree()
~RedBlackTree()
T
1
has
0...*
Node
-key: int
-color: bool
-parent: Node<T> *
-leftChild: Node<T> *
-rightChild: Node<T> *
-data: T
+setKey(key: int): void
+setColor(color: bool): void
+setParent(parent: Node<T> *): void
+setLeftChild(leftChild: Node<T> *): void
+setRightChild(rightChild: Node<T> *): void
+setData(data: T): void
+getKey(): int
+getColor(): bool
+getParent(): Node<T> *
+getLeftChild(): Node<T> *
+getRightChild(): Node<T> *
+getData(): T
+Node(key: int, data: T)
+Node()
~Node()
T
1
has parent
0..1
0..1
has left child
1
0..1
has right child
1