Contents

1 A java program that generates a BST

```
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.NoSuchElementException;
import java.util.Stack;

public class BST<E extends Comparable<E>> implements Iterable<E> {
// ^ Cannot just use E because some objects cannot be compared

public static void main(String[] args) {
java.util.Random rand = new java.util.Random();
BST<Integer> tree = new BST<Integer>();
System.out.print("Adding: ");
for (int i = 0; i < 15; i++) {
int r = rand.nextInt(100);
System.out.print(r + ", ");
tree.add(r);
}
System.out.println();
System.out.println("Printing tree with .toArray() : " + Arrays.toString(tree.toArray()
System.out.println("Printing tree with .print() : ");
tree.print();
System.out.println(tree);
System.out.println("Printing with for loop: ");
for (int x : tree)
System.out.print(x + ", ");
System.out.println("");
System.out.println("Printing with .printpreorder()  ");
tree.printpreorder();

BST<String> sTree = new BST<String>();
sTree.add("Frodo");
sTree.add("Pippin");
sTree.add("Sam");
sTree.add("Merry");
sTree.add("Fatty");
System.out.println("Printing tree with .print() : ");
sTree.print();
System.out.println("Printing with toString : " + sTree);

System.out.println("Printing with a for loop: ");
for (String s : sTree)
System.out.print(s + ", ");
System.out.println();
System.out.println("Printing with .toArray() : " + Arrays.toString(sTree.toArray()));
}

private class Node {
E element;
Node left;
Node right;
}

private Node root;
private int size;

public BST() {
root = null;
size = 0;
}

@Override
public boolean equals(Object obj) {
if (obj instanceof BST) {
BST<E> t = (BST<E>) obj;
if (t.size != this.size)
return false;
else {
LinkedList<E> l1 = t.toList();
LinkedList<E> l2 = this.toList();
if (l1.equals(l2))
return true;
else
return false;
}

} else
return false;
}

public LinkedList<E> toList() {
LinkedList<E> l = toList(root);
return l;
}

public LinkedList<E> toList(Node n) {
if (n == null) {
// if null return Empty list
LinkedList<E> el = new LinkedList<E>();
return el;
} else {
LinkedList<E> a = toList(n.left);
a.add(n.element);
a.addAll(toList(n.right));
return a;
}
}

public boolean contains(E element) {
return contains(element, root);
}

public boolean contains(E element, Node n) {
if (n == null)
return false;
else if (n.element.equals(element))
return true;
else if (n.element.compareTo(element) < 0)
return contains(element, n.right);
else
return contains(element, n.left);
}
```

```
public int leafamount() {
return leafamount(root);
}

public int leafamount(Node n) {
if (n == null)
return 0;
else if (n.left == null && n.right == null)
return 1;
return leafamount(n.left) + leafamount(n.right);

}

public int height() {
return height(root);
}

public int height(Node n) {
if (n == null)
return -1;
else {
int lefth = height(n.left);
int righth = height(n.right);
if (lefth > righth)
return lefth + 1;
else
return righth + 1;
}
}

@Override
public String toString() {
String s = toString(root);
s = s.substring(0, s.length() - 1);
return "[" + s + "]";
}

public String toString(Node n) {
if (n == null) {
return "";
} else {
String s = toString(n.left);
s += n.element.toString() + ",";
s += toString(n.right);
return s;
}
}

public void print() {
print(root);
System.out.println();
}

private void print(Node n) {
if (n != null) {
print(n.left);
System.out.print(n.element + " ");
print(n.right);
}
}

public void printpreorder() {
printpreorder(root);
System.out.println();
}

private void printpreorder(Node n) {
if (n != null) {
System.out.print(n.element + ", ");
printpreorder(n.left);
printpreorder(n.right);
}
}

public int size() {
return size;
}

public void add(E element) {
Node n = new Node();
```

```
n.element = element;
if (root == null) {
root = n;
} else {
Node parent = null;
Node walker = root;
while (walker != null) {
parent = walker;
if (walker.element.compareTo(element) > 0) {
walker = walker.left;
} else {
walker = walker.right;
}
}
if (parent.element.compareTo(element) > 0) {
parent.left = n;
} else {
parent.right = n;
}
}
size++;
}

public void printwloop() {
System.out.print('[');
Stack<Node> s = new Stack<Node>();
Node n = root;
while (n != null || s.size() > 0) {
// get to the element all the way to the left
while (n != null) {
s.push(n);
n = n.left;
}
n = s.pop();

E e = n.element;
System.out.print(e + ", ");
n = n.right;
}
System.out.print("]\n");
```

```
}

private class TreeIterator implements Iterator<E> {

private Node w;
private Stack<Node> s = new Stack<Node>();

public TreeIterator(BST b) {
w = b.root;
s.push(w);
}

@Override
public boolean hasNext() {
return !s.isEmpty();
}

@Override
public E next() {
if (hasNext()) {
w = s.pop();
if (w.right != null)
s.push(w.right);
if (w.left != null)
s.push(w.left);
return w.element;
} else
throw new NoSuchElementException();

}
}

@Override
public Iterator<E> iterator() {
return new TreeIterator(this);
}

public Object[] toArray() {
Object[] a = new Object[size];
toArray(a, 0, root);
```

```
return a;
}

private int toArray(Object[] a, int i, Node n) {
if (i < size && n != null) {
i = toArray(a, i, n.left);
a[i++] = n.element;
i = toArray(a, i, n.right);
}
return i; // return current index after elements added
}

}
```

```
Adding: 21, 79, 23, 22, 5, 91, 24, 55, 60, 68, 38, 8, 24, 95, 5,
Printing tree with .toArray() : [5, 5, 8, 21, 22, 23, 24, 24, 38, 55, 60, 68, 79, 91, 9
Printing tree with .print() :
5 5 8 21 22 23 24 24 38 55 60 68 79 91 95
[5,5,8,21,22,23,24,24,38,55,60,68,79,91,95]
Printing with for loop:
21, 5, 8, 5, 79, 23, 22, 24, 55, 38, 24, 60, 68, 91, 95,
Printing with .printpreorder()
21, 5, 8, 5, 79, 23, 22, 24, 55, 38, 24, 60, 68, 91, 95,
Printing tree with .print() :
Fatty Frodo Merry Pippin Sam
Printing with toString : [Fatty,Frodo,Merry,Pippin,Sam]
Printing with a for loop:
Frodo, Fatty, Pippin, Merry, Sam,
Printing with .toArray() : [Fatty, Frodo, Merry, Pippin, Sam]
```