Homework 6

Source Code:

```java
// Homework6.java

package com;

// import com.BinaryTree.java
public class Homework6 {

        public static void main(String[] args) {

                // Create two new binary trees
                BinaryTree<Integer> tree1 = new BinaryTree<>();
                BinaryTree<Integer> tree2 = new BinaryTree<>();

                // Build Tree #1
                tree1.insertRoot(1);
                tree1.getRoot().insertLeft(2);
                tree1.getRoot().insertRight(3);
                tree1.getRoot().getLeft().insertLeft(4);
                tree1.getRoot().getLeft().getLeft().insertLeft(7);
                tree1.getRoot().getRight().insertLeft(5);
                tree1.getRoot().getRight().insertRight(6);
                tree1.getRoot().getRight().getRight().insertRight(8);
                tree1.getRoot().getRight().getRight().getRight().insertRight(9);

                // Build Tree #2
                tree2.insertRoot(6);
                tree2.getRoot().insertLeft(4);
                tree2.getRoot().insertRight(8);
                tree2.getRoot().getLeft().insertLeft(2);
                tree2.getRoot().getLeft().insertRight(5);
                tree2.getRoot().getRight().insertLeft(7);
                tree2.getRoot().getRight().insertRight(9);
                tree2.getRoot().getLeft().getLeft().insertLeft(1);
                tree2.getRoot().getLeft().getLeft().insertRight(3);

                // test counting leaves
                System.out.println("The number of leaves of tree#1 is " +
                                                countLeaves(tree1) + ".");
                System.out.println("The number of leaves of tree#2 is " +
                                countLeaves(tree2) + ".");

                // test counting non-leaves
                System.out.println("The number of non-leaves of tree#1 is " +
                                countNonLeaves(tree1) + ".");
                System.out.println("The number of non-leaves of tree#2 is " +
                                countNonLeaves(tree2) + ".");

                // test getting height
                System.out.println("The height of tree#1 is " +
                                getHeight(tree1) + ".");
                System.out.println("The height of tree#2 is " +
                                getHeight(tree2) + ".");

                // test printing tree using pre-order traversal
                System.out.println("Tree#1 in pre-order traversal: ");
                printPreOrder(tree1);
                System.out.println();

                System.out.println("Tree#2 in pre-order traversal: ");
                printPreOrder(tree2);
                System.out.println();

                // test printing tree using in-order traversal
                System.out.println("Tree#1 in in-order traversal: ");
                printInOrder(tree1);
                System.out.println();
```

```java
        System.out.println("Tree#2 in in-order traversal: ");
        printInOrder(tree2);
        System.out.println();

        // test printing tree using post-order traversal
        System.out.println("Tree#1 in post-order traversal: ");
        printPostOrder(tree1);
        System.out.println();

        System.out.println("Tree#2 in post-order traversal: ");
        printPostOrder(tree2);
        System.out.println();

        // test removing leaves with tree#1
        removeLeaves(tree1);
        System.out.println("Tree#1 after removing leaves (pre-order traversal): ");
        printInOrder(tree1);
        System.out.println();

        // test removing leaves with tree#2
        removeLeaves(tree2);
        System.out.println("Tree#2 after removing leaves (post-order traversal): ");
        printPostOrder(tree2);
        System.out.println();

    }

    // Helper method for counting leaves
    private static int countLeavesHelper(BinaryTree<Integer>.Node node) {

        // when tree is empty
        if (node == null) {
                return 0;
        }

        // when only has one root
        if (node.getLeft() == null && node.getRight() == null) {
                return 1;
        }

        // recursively counting the number of leaves
        else {
                return countLeavesHelper(node.getLeft()) +
                                countLeavesHelper(node.getRight());
        }

    }

    // Helper method for getting height
    private static int getHeightHelper(BinaryTree<Integer>.Node node) {

        // when tree is empty
        if(node == null) {
                return 0;
        }

        // recursively counting the height of tree
        return 1 + Math.max(getHeightHelper(node.getLeft()),
                        getHeightHelper(node.getRight()));
    }

    // Helper method for pre-order printing
    private static void preOrderHelper(BinaryTree<Integer>.Node node) {

        // when tree is empty
        if (node == null) {
                return;
        }

        // recursively traverse and print the tree
        System.out.print(node.getData() + " ");
        preOrderHelper(node.getLeft());
        preOrderHelper(node.getRight());
```

```java
        }

        // Helper method for in-order printing
        private static void inOrderHelper(BinaryTree<Integer>.Node node) {

                // when tree is empty
                if (node == null) {
                        return;
                }

                // recursively traverse and print the tree
                inOrderHelper(node.getLeft());
                System.out.print(node.getData() + " ");
                inOrderHelper(node.getRight());
        }

        // Helper method for post-order printing
        private static void postOrderHelper(BinaryTree<Integer>.Node node) {

                // when tree is empty
                if (node == null) {
                        return;
                }

                // recursively traverse and print the tree
                inOrderHelper(node.getLeft());
                inOrderHelper(node.getRight());
                System.out.print(node.getData() + " ");
        }

        // Helper method for removing leaves
        private static BinaryTree<Integer>.Node removeLeavesHelper(BinaryTree<Integer>.Node
node) {

                // when tree is empty
                if (node == null) {
                        return null;
                }

                // when tree only has the root
                if (node.getLeft() == null && node.getRight() == null) {
                        return null;
                }

                // recursively remove leaves
                node.left = removeLeavesHelper(node.left);
                node.right = removeLeavesHelper(node.right);
                return node;
        }

        // Counting leaves
        public static int countLeaves(BinaryTree<Integer> tree) {
                return countLeavesHelper(tree.getRoot());
        }

        // Counting non-leaves
        public static int countNonLeaves(BinaryTree<Integer> tree) {
                return tree.getSize() - countLeaves(tree);
        }

        // Getting height of tree
        public static int getHeight(BinaryTree<Integer> tree) {
                return getHeightHelper(tree.getRoot());
        }

        // Printing tree with pre-order tranversal
        public static void printPreOrder(BinaryTree<Integer> tree) {
                preOrderHelper(tree.getRoot());
        }

        // Printing tree with in-order traversal
        public static void printInOrder(BinaryTree<Integer> tree) {
                inOrderHelper(tree.getRoot());
        }
```

```java
        // Printing tree with post-order traversal
        public static void printPostOrder(BinaryTree<Integer> tree) {
                postOrderHelper(tree.getRoot());
        }

        // Removing all leaves
        public static void removeLeaves(BinaryTree<Integer> tree) {
                removeLeavesHelper(tree.getRoot());
        }

}
```

Output:

```
Problems  @ Javadoc  Declaration  Console ×
<terminated> Homework6 [Java Application] /Library/Java/JavaVirtualMachines
The number of leaves of tree#1 is 3.
The number of leaves of tree#2 is 5.
The number of non-leaves of tree#1 is 6.
The number of non-leaves of tree#2 is 4.
The height of tree#1 is 5.
The height of tree#2 is 4.
Tree#1 in pre-order traversal:
1 2 4 7 3 5 6 8 9
Tree#2 in pre-order traversal:
6 4 2 1 3 5 8 7 9
Tree#1 in in-order traversal:
7 4 2 1 5 3 6 8 9
Tree#2 in in-order traversal:
1 2 3 4 5 6 7 8 9
Tree#1 in post-order traversal:
7 4 2 5 3 6 8 9 1
Tree#2 in post-order traversal:
1 2 3 4 5 7 8 9 6
Tree#1 after removing leaves (pre-order traversal):
4 2 1 3 6 8
Tree#2 after removing leaves (post-order traversal):
2 4 8 6
```

a) Returns the number of leaf nodes in the tree.

```
The number of leaves of tree#1 is 3.
The number of leaves of tree#2 is 5.
```

b) Returns the number of non-leaf nodes in the tree.

```
The number of non-leaves of tree#1 is 6.
The number of non-leaves of tree#2 is 4.
```

c) Returns the height of the tree.

```
The height of tree#1 is 5.
The height of tree#2 is 4.
```

d) Prints the elements of the tree using a pre-order traversal.

```
Tree#1 in pre-order traversal:
1 2 4 7 3 5 6 8 9
Tree#2 in pre-order traversal:
6 4 2 1 3 5 8 7 9
```

e) Prints the elements of the tree using an in-order traversal.

```
Tree#1 in in-order traversal:
7 4 2 1 5 3 6 8 9
Tree#2 in in-order traversal:
1 2 3 4 5 6 7 8 9
```

f) Prints the elements of the tree using a post-order traversal.

```
Tree#1 in post-order traversal:
7 4 2 5 3 6 8 9 1
Tree#2 in post-order traversal:
1 2 3 4 5 7 8 9 6
```

g) Removes all leaf nodes from the tree. Use `printPreOrder`, `printInOrder`, or `printPostOrder` after calling `removeLeaves` to show that `removeLeaves` successfully removed all leaves.

```
Tree#1 after removing leaves (pre-order traversal):
4 2 1 3 6 8
Tree#2 after removing leaves (post-order traversal):
2 4 8 6
```