IF184301 Object Oriented Programming (E)

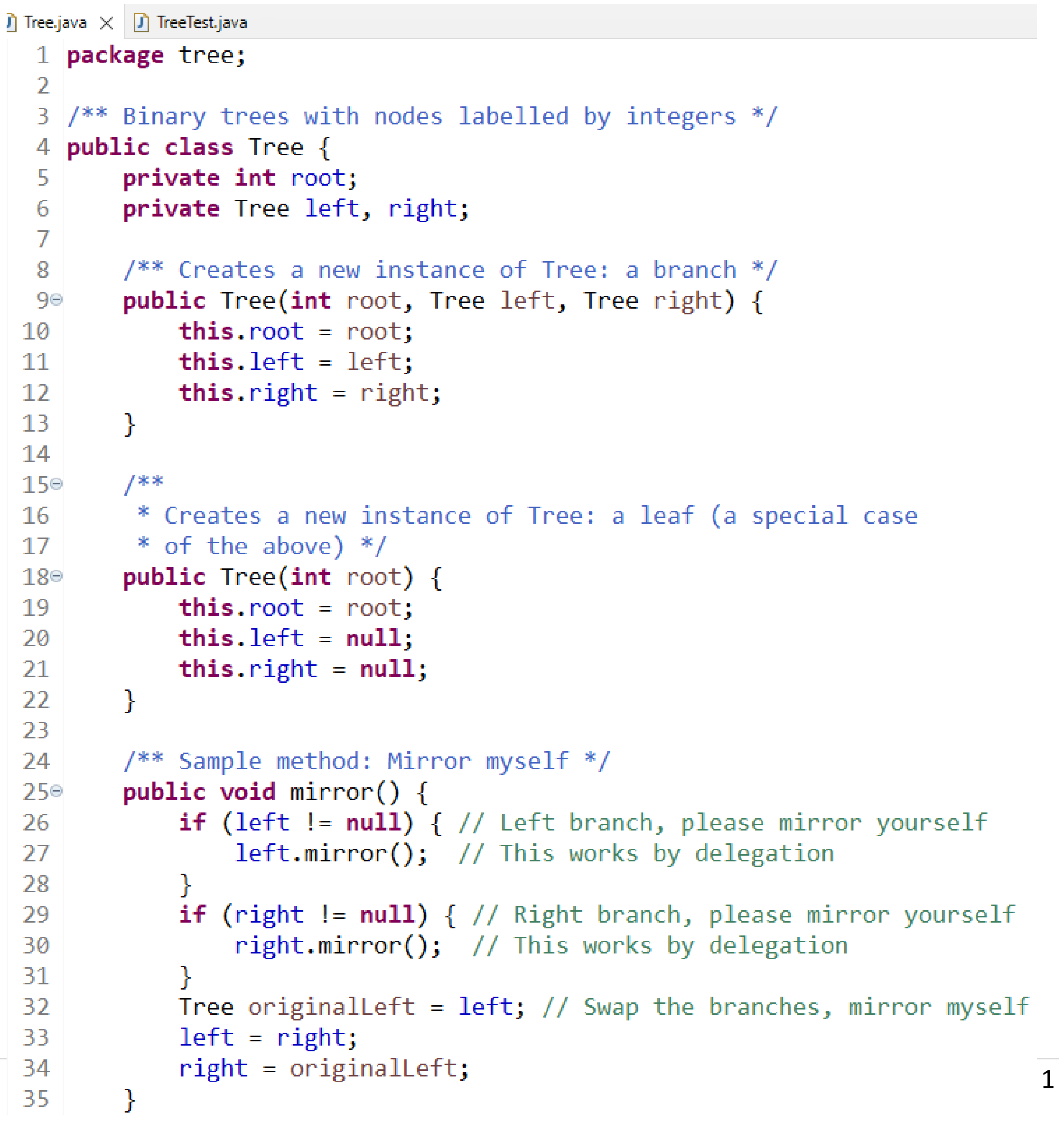
Midterm Exam

Starting date:	21 October 2021
Deadline:	28 October 2021, 23:59 WIB. **Penalty: 0.15% of grade/minute of tardiness.**
Exam type:	Open
Send to:	MM Irfan Subakti <yifana@gmail.com> CC to Dicksen Alfersius Novian <dicksenan@gmail.com> with the subject: **IF184301_OOP(E)_MID_StudentID_Name**
File type and format:	A **zip file** containing the *working file* & the *declaration*
Filename format:	IF184301_OOP(E)_MID_StudentID_Name.ZIP

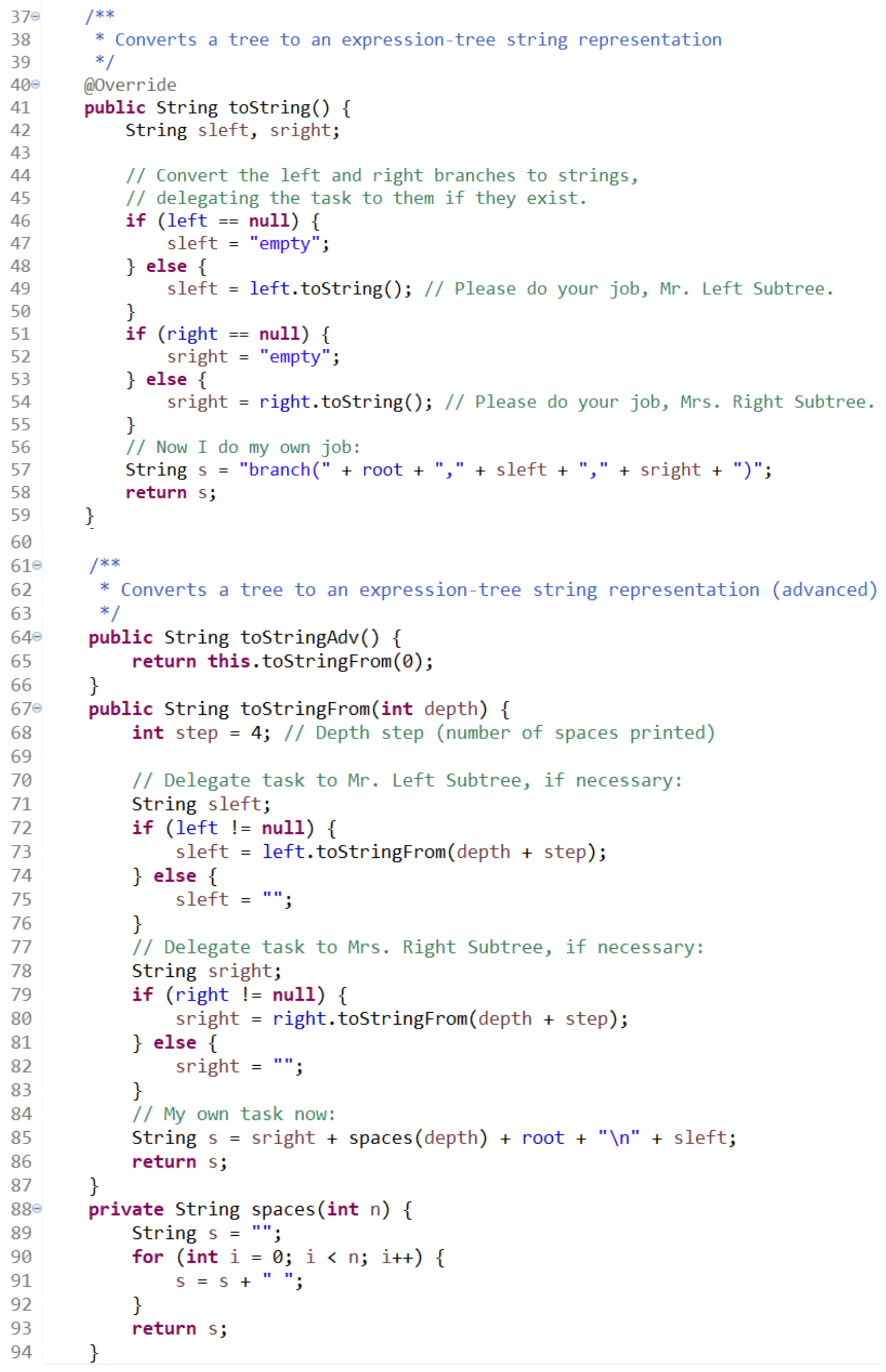
Instruction

Please do these steps as in the following.

0. Download the *working template file* – and use it for your work, i.e., `IF184301_OOP(E)_MID_StudentID_Name.zip`. There are two packages: `tree` and `list`. No. 1-7 utilise package `tree`, likewise, package `list` will be used for no. 8-15.



```java
package tree;

/** Binary trees with nodes labelled by integers */
public class Tree {
    private int root;
    private Tree left, right;

    /** Creates a new instance of Tree: a branch */
    public Tree(int root, Tree left, Tree right) {
        this.root = root;
        this.left = left;
        this.right = right;
    }

    /**
     * Creates a new instance of Tree: a leaf (a special case
     * of the above) */
    public Tree(int root) {
        this.root = root;
        this.left = null;
        this.right = null;
    }

    /** Sample method: Mirror myself */
    public void mirror() {
        if (left != null) { // Left branch, please mirror yourself
            left.mirror();  // This works by delegation
        }
        if (right != null) { // Right branch, please mirror yourself
            right.mirror();  // This works by delegation
        }
        Tree originalLeft = left; // Swap the branches, mirror myself
        left = right;
        right = originalLeft;
    }
```



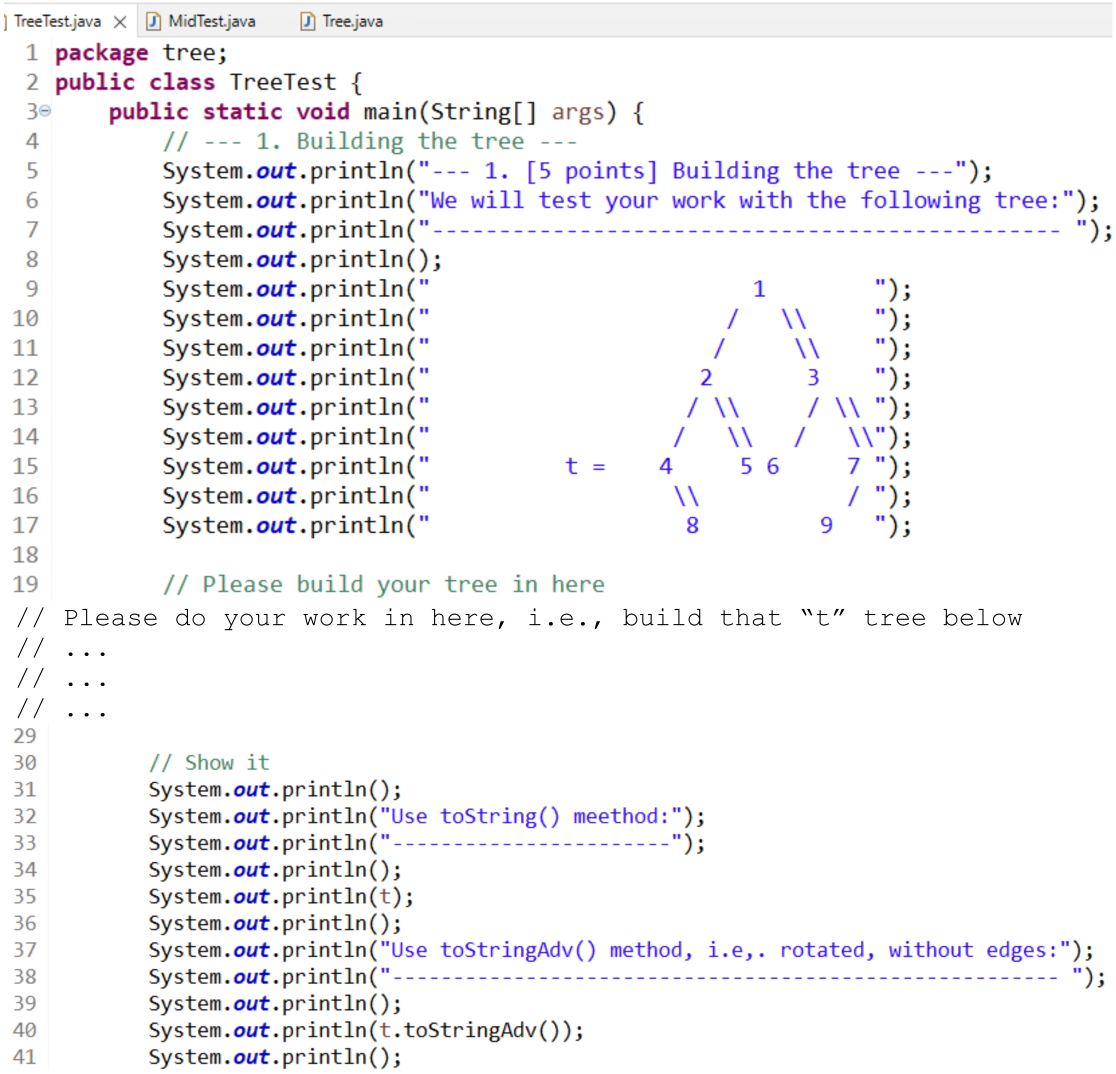
```java
    /**
     * Converts a tree to an expression-tree string representation
     */
    @Override
    public String toString() {
        String sleft, sright;

        // Convert the left and right branches to strings,
        // delegating the task to them if they exist.
        if (left == null) {
            sleft = "empty";
        } else {
            sleft = left.toString(); // Please do your job, Mr. Left Subtree.
        }
        if (right == null) {
            sright = "empty";
        } else {
            sright = right.toString(); // Please do your job, Mrs. Right Subtree.
        }
        // Now I do my own job:
        String s = "branch(" + root + "," + sleft + "," + sright + ")";
        return s;
    }

    /**
     * Converts a tree to an expression-tree string representation (advanced)
     */
    public String toStringAdv() {
        return this.toStringFrom(0);
    }
    public String toStringFrom(int depth) {
        int step = 4; // Depth step (number of spaces printed)

        // Delegate task to Mr. Left Subtree, if necessary:
        String sleft;
        if (left != null) {
            sleft = left.toStringFrom(depth + step);
        } else {
            sleft = "";
        }
        // Delegate task to Mrs. Right Subtree, if necessary:
        String sright;
        if (right != null) {
            sright = right.toStringFrom(depth + step);
        } else {
            sright = "";
        }
        // My own task now:
        String s = sright + spaces(depth) + root + "\n" + sleft;
        return s;
    }
    private String spaces(int n) {
        String s = "";
        for (int i = 0; i < n; i++) {
            s = s + " ";
        }
        return s;
    }
```

No. 1-7. In the package `tree`, we have the following Java class, i.e., `Tree.java`, and you must fill with more methods in the following questions. Use `TreeTest.java` from the package `tree` to test your works.

1. [**5 points**] Write Java code to create the following tree using `new Tree` statements:

```
          1
         / \
        /   \
       2     3
      / \   / \
     /   \ /   \
    4    5 6    7
     \         /
      8       9
```

Hint: You will need 9 such statements, i.e., 9 of `new Tree` statements right away at `TreeTest.java` as follows.



```java
package tree;
public class TreeTest {
    public static void main(String[] args) {
        // --- 1. Building the tree ---
        System.out.println("--- 1. [5 points] Building the tree ---");
        System.out.println("We will test your work with the following tree:");
        System.out.println("------------------------------------------------ ");
        System.out.println();
        System.out.println("                        1        ");
        System.out.println("                      /   \\     ");
        System.out.println("                     /     \\    ");
        System.out.println("                    2       3    ");
        System.out.println("                   / \\     / \\ ");
        System.out.println("                  /   \\   /   \\");
        System.out.println("          t =    4     5 6     7 ");
        System.out.println("                 \\            / ");
        System.out.println("                   8         9   ");

        // Please build your tree in here
// Please do your work in here, i.e., build that "t" tree below
// ...
// ...
// ...

       // Show it
       System.out.println();
       System.out.println("Use toString() meethod:");
       System.out.println("----------------------");
       System.out.println();
       System.out.println(t);
       System.out.println();
       System.out.println("Use toStringAdv() method, i.e,. rotated, without edges:");
       System.out.println("-------------------------------------------------------- ");
       System.out.println();
       System.out.println(t.toStringAdv());
       System.out.println();
```

2. [**5 points**] Include a *recursive* method in `Tree.java` that will multiply by three every node of the tree:

```java
public void triple() {
    // Your code is in here
}
```

For example, the tree

```
         3
        / \
       /   \
t =   4     5
     / \   / \
    /   \ /   \
   6    7 8    9
```

after doing

```
    t.triple();
```

should become

```
         9
        / \
       /   \
      12    15
     / \    / \
    /   \  /   \
   18   21 24   27
```

Hint: Ask your left and right subtrees to apply the procedure to themselves and then multiply by three your root node.

Don't forget to always use `TreeTest.java` to test the correctness of your works.

3. [**5 points**] Include a *recursive* method in `Tree.java` that will print the dept-first traversal of a tree, i.e., choosing left branches first.

```
public void printDepthFirst() {
   // Your code is in here
}
```

For example, for the tree

```
         3
        / \
       /   \
t =   4     5
     / \   / \
    /   \ /   \
   6    7 8    9
```

Should print

```
    3 4 6 7 5 8 9
```

Hint: Print your root, then ask your left subtree to print its depth-first traversal, and finally ask your right-subtree to print its depth-first traversal.

4. [**10 points**] Include a *recursive* method in `Tree.java` that generate a fresh copy of a tree:

```
public Tree createFreshCopy() {
   // Your code is in here
}
```

For example, given a tree `t`, this could be used as follows:

```
Tree u = t.createFreshCopy();
t.mirror();
```

Then the tree `u` should be exactly as `t` originally was before mirroring.
You are *required* to do this by executing `new` statements (one for each node of the tree).
You are *not allowed* to use the Java clone facility.

Hint: Ask your left and right subtrees to create fresh copies of themselves (and return them for you). Then use their fresh copies to generate a fresh copy of yourself, using a `new Tree` statement.

5. [**5 points**] Include a *recursive* method in `Tree.java` that will store the depth-first traversal of a tree into a given array, as follows:

```
// Returns the last position of the array
public int saveDepthFirst(int a[], int whereToStart) {
   // Your code is in here
}
```

For example, for the tree

```
        4
       /  \
      /    \
t =  5      6
    / \    / \
   /   \  /   \
  7    8  9    10
```

the program fragment

```
   int a[] = new int[13];
   int beginning = 4;
   int end = t.saveDepthFirst(a, beginning);
```

should fill the entries of the array as follows:

```
   ? ? ? ? 4 5 7 8 6 9 10 ? ?
```

and set the variable `end` to 10 (the last position of the array).

Hint: Store your root at position `whereToStart`. Ask your left subtree to store its depth-first traversal at position `whereToStart + 1`, and return the next available position to you, then ask your right-subtree to store its depth-first traversal, starting from that returned position. Don't forget to be polite and pass the right's result back to *your* caller, using a return statement with the value returned by the call to the right sub-tree.

6. [**5 points**] Recall the definition of a *binary search tree* (BST): All the nodes in the left branch are smaller than the root, all the nodes in the right branch are bigger than the root, and the left and right subtrees are themselves BST. Include a *recursively defined* method

```
public Tree find(int n) {
    // Your code is in here
}
```

that assuming you are a binary search tree, will return a (reference to) the subtree whose root is equal to `n`. If such a subtree doesn't exist, you should return *null*. Your algorithm must be *efficient*: It should not search the whole tree, but one and only one path going down from the root.

Hint: Check your root: What you are looking for maybe there. If not, decide whether to ask left or right to do their jobs, but not both.

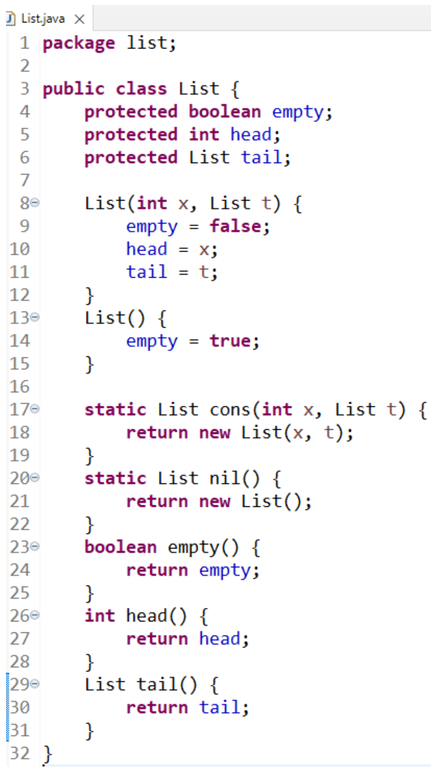
7. [**5 points**] By using BST from no. 6, include a *recursive* method in `Tree.java` to insert a new node in your BST in the correct position (which will be a leaf):

```
public Tree insert(int n) {
    // Your code is in here
}
```

Return the new tree if you succeed, and the current tree if the element is already there and hence cannot be inserted.

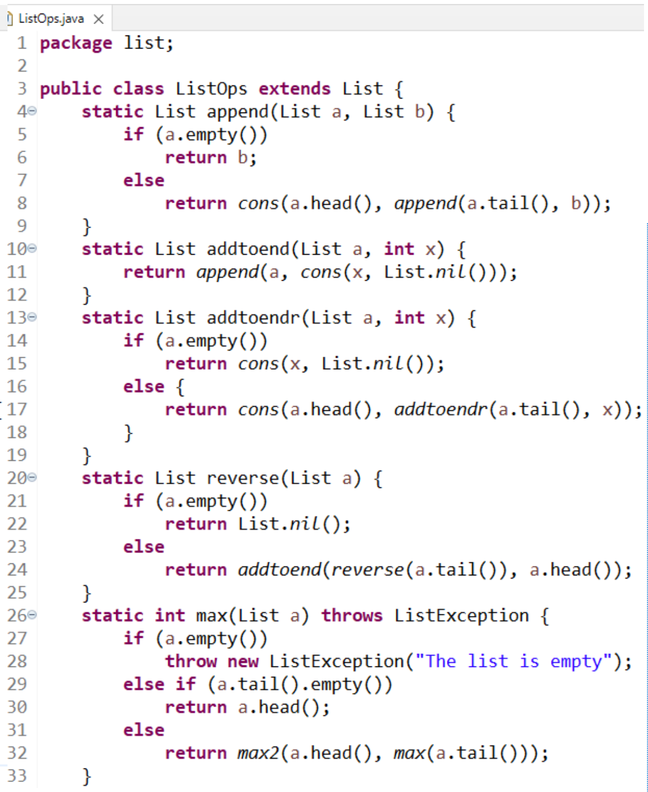
Hint: Suitably modify your solution for *finding* an element (no. 6). If you find the element, then it cannot be inserted. If you don't find it, you have reached a leaf, and your element must be inserted as a child of that leaf (and hence the leaf will not be a leaf any longer).

No. 8-15. In the package `list`, we have the following Java classes, i.e., `List.java`, `ListException.java`, `ListOps.java`, and you must fill with more methods in the following questions which should be written in `Mid.java`. ***Only make use of all of the methods in*** *`List.java`* ***and*** *`ListOps.java`* ***for your works***. Use `MidTest.java` from the package `list` to test your works.



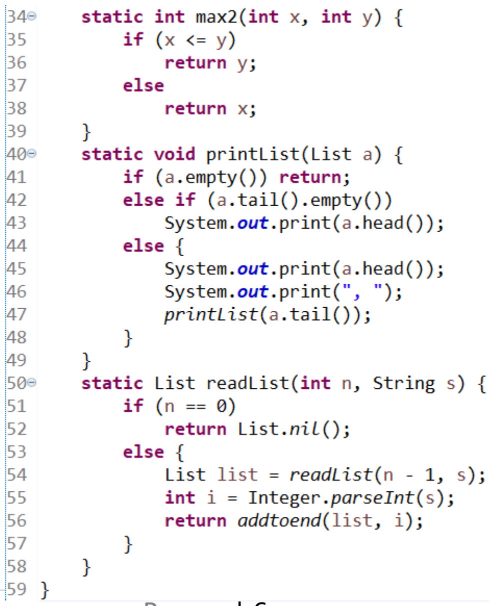
List.java

```java
package list;

public class List {
    protected boolean empty;
    protected int head;
    protected List tail;

    List(int x, List t) {
        empty = false;
        head = x;
        tail = t;
    }
    List() {
        empty = true;
    }

    static List cons(int x, List t) {
        return new List(x, t);
    }
    static List nil() {
        return new List();
    }
    boolean empty() {
        return empty;
    }
    int head() {
        return head;
    }
    List tail() {
        return tail;
    }
}
```



ListOps.java

```java
package list;

public class ListOps extends List {
    static List append(List a, List b) {
        if (a.empty())
            return b;
        else
            return cons(a.head(), append(a.tail(), b));
    }
    static List addtoend(List a, int x) {
        return append(a, cons(x, List.nil()));
    }
    static List addtoendr(List a, int x) {
        if (a.empty())
            return cons(x, List.nil());
        else {
            return cons(a.head(), addtoendr(a.tail(), x));
        }
    }
    static List reverse(List a) {
        if (a.empty())
            return List.nil();
        else
            return addtoend(reverse(a.tail()), a.head());
    }
    static int max(List a) throws ListException {
        if (a.empty())
            throw new ListException("The list is empty");
        else if (a.tail().empty())
            return a.head();
        else
            return max2(a.head(), max(a.tail()));
    }
```



```java
    static int max2(int x, int y) {
        if (x <= y)
            return y;
        else
            return x;
    }
    static void printList(List a) {
        if (a.empty()) return;
        else if (a.tail().empty())
            System.out.print(a.head());
        else {
            System.out.print(a.head());
            System.out.print(", ");
            printList(a.tail());
        }
    }
    static List readList(int n, String s) {
        if (n == 0)
            return List.nil();
        else {
            List list = readList(n - 1, s);
            int i = Integer.parseInt(s);
            return addtoend(list, i);
        }
    }
}
```

8. [**5 points**] Include a *recursive* method in `Mid.java` that will negate a list:

   ```
   static List negateAll(List a) {
      // Your code is in here
   }
   ```

   Given a list of integers `a`, write a method that returns a new list with all the elements of `a` with sign negated, i.e., positive integers become negatives and negative integers become positives. Example:

   ```
   [2, -5, 8, 0] ==> [-2, 5, -8, 0]
   ```

9. [**5 points**] Include a *recursive* method in `Mid.java` that will perform searching for an element:

   ```
   static int find(int x, List a) {
      // Your code is in here
   }
   ```

   Given an integer `x` and a list `a`, write a method that returns the position of the first occurrence of `x` in `a`. Positions are counted as `0, 1, 2, …`. If `x` does not appear in the list, your method must return `-1`. Example:

   ```
   x: 3     a: [7, 5, 3, 8] ==> 2
   x: 2     a: [7, 5, 3, 8] ==> -1
   ```

10. [**5 points**] Include a *recursive* method in `Mid.java` that will check for positive:

    ```
    static boolean allPositive(List a) {
       // Your code is in here
    }
    ```

    Given a list of integers `a`, return a boolean value indicating whether *all* its elements are positive, i.e., ≥ 0.

11. [**10 points**] Include a *recursive* method in `Mid.java` that will find the positives:

    ```
    static List positives(List a) {
       // Your code is in here
    }
    ```

    Given a list of integers `a`, return a new list that contains all the positive elements of `a`. The elements should appear in the result in the same relative order as in `a`. Example:

    ```
    [2, 3, -5, 8, -2] ==> [2, 3, 8]
    ```

12. [**10 points**] Include a *recursive* method in `Mid.java` that will check the sorted-*ness*:

    ```
    static boolean sorted(List a) {
       // Your code is in here
    }
    ```

Given a list of integers `a`, this method must return a boolean value indicating whether `a` is sorted in increasing order. There can be duplicate copies of elements. But, sorted-*ness* would require that all the duplicate copies would appear together.

13. [**10 points**] Include a *recursive* method in `Mid.java` that will perform merging:

```
static List merge(List a, List b) {
   // Your code is in here
}
```

Given two *sorted* lists of `a` and `b`, your method must return a new *sorted* list that contains all the elements of `a` and all the elements of `b`. Any duplicate copies of elements in `a` or `b` or their combination are retained. Example:

```
a: [2, 5, 5, 8]   b: [5, 7, 8, 9] ==> [2, 5, 5, 5, 7, 8, 8, 9]
a: [2, 5, 5, 8]   b: [9] ==> [2, 5, 5, 8, 9]
```

14. [**10 points**] Include a *recursive* method in `Mid.java` that will remove duplicates:

```
static List removeDuplicates(List a) {
   // Your code is in here
}
```

Given a list `a`, this method must return a copy of the list `a` with all duplicate copies removed. Example:

```
[2, 5, 5, 5, 7, 8, 8, 9]   ==> [2, 5, 7, 8, 9]
```

Hint: Think of defining helper functions, i.e., `skip()` as no. 15 below.

15. [**5 points**] Include a *recursive* method in `Mid.java` that will skip a given element of the list:

```
static List skip(int x, List a) {
   // Your code is in here
}
```

Given a list `a`, this method must return a copy of the list `a` with a given element `a` skipped. Example:

```
x: 5, a: [2, 5, 5, 5, 7, 8, 8, 9]   ==> [2, 7, 8, 8, 9]
```

Hint: This method will be used at no. 14 above.

16. To avoid plagiarism/cheating, every student needs to pledge and declare, then she/he must submit her/his **signed pledge and declaration** as in the following. Failing to do so will be resulted in getting a 0 (zero) grade. Attach the **scanned/photo** of your *declaration* in your report.

    "By the name of Allah (God) Almighty, herewith I pledge and truly declare that I have solved midterm exam by myself, didn't do any cheating by any means, didn't do any plagiarism, and didn't accept anybody's help by any means. I am going to accept all of the consequences by any means if it has proven that I have been done any cheating and/or plagiarism."

    [Place, e.g., Surabaya], [date, e.g., 28 October 2021]

    <Signed>

    [Full name, e.g., Sri Purwaningsih]
    [StudentID, e.g., 05112040000xxx]

17. Have a wonderful day, guys! Good luck! 😊