10.1 PA10: Expression Trees

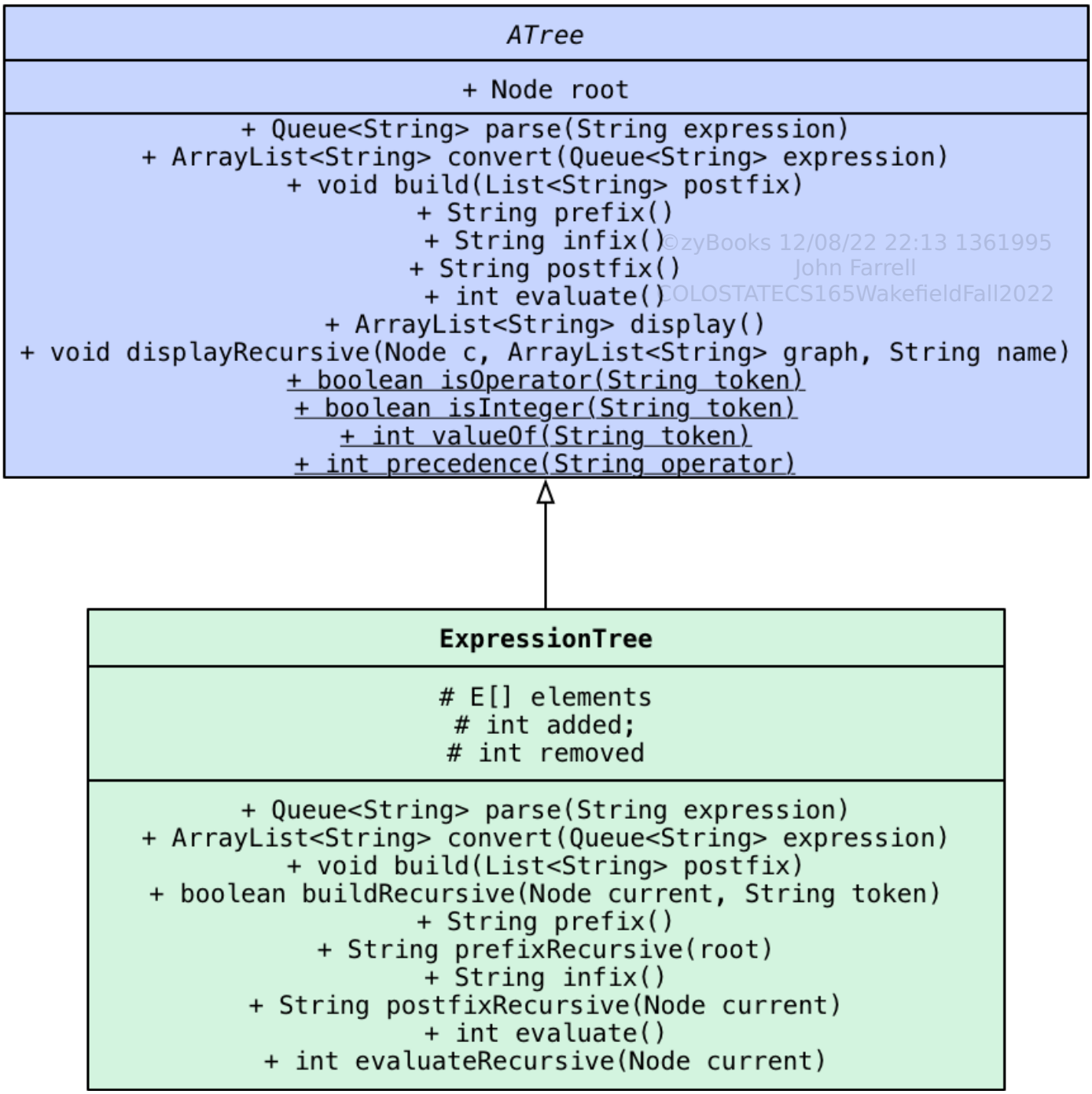
Objectives

- Learn how to parse an infix expression into a list of tokens.
- Convert a list of tokens from infix to postfix.
- Build a tree data structure to represent the expression.
- Traverse the tree in prefix, infix, and postfix order.
- Use the tree to evaluate the expression.

Directions

The following UML representation shows the structure of the classes you'll be working with:

©zyBooks 12/08/22 22:13 1361995
John Farrell
COLOSTATECS165WakefieldFall2022



This assignment covers tree building and tree traversal, in addition to conversion between prefix, infix, and postfix representations of expressions. The data structure is an expression tree, not a binary search tree as discussed in Chapter 25 of the Liang textbook. The lexical analysis involved in this assignment was previously done in a lab.

In the ExpressionTree class, implement the following methods (or their corresponding helper methods):

parse method

convert method (**Note: You may use the `convert` method you designed in the lab.**)

build method (**Note: You may use the `build` method you designed in the lab.**)

prefix method

infix method

postfix method

evaluate method

Testing

The Unit_Testing class is provided for testing, and is similar to the code used for automated grading. Make sure to read through the provided tests to understand how these methods interact with each other. You are encouraged to write more tests than are provided to practice with the code you have written and unit testing in general. If you run into bugs in your program, use the debugger to understand what your program is actually doing.

The unit tests provided are rudimentary and may not cover every case. That's your job to finish. It's suggested that you test with more complex expressions to make sure your code works for all cases. Looking at the code coverage of your tests is also a great way to make sure you haven't missed any cases.

Specifications

Your program must meet the following specifications:

The code must produce the correct upload file and assertion messages.

Work on your own, as always.

You must submit the completed version of ExpressionTree.java.

Assignments should be implemented using Java 1.8.

Read the syllabus for the late policy.

We will be checking programs for plagiarism, so please don't copy from anyone else.

Grading Criteria

100 points for perfect submission.

0 points for no submission, will not compile, submitted class file, etc.

Simpler Tests: 10 points each

Harder Tests: 15 points each

For each test, we vary the input to include more operators and operands.

Total: 100 points

422352.2723990.qx3zqy7

LAB ACTIVITY | 10.1.1: PA10: Expression Trees | 0 / 100

Downloadable files

`ExpressionTree.java` and `P7_Unit_Testing.java` **Download**

Current file: **ExpressionTree.java** Load default template...

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4  import java.nio.charset.MalformedInputException;
5  import java.util.*;
6
7
8  /**
9   *
10  */
11 public class ExpressionTree extends ZTree {
12
13     public Queue<String> parse(String expression) {
14         Queue<String> infix = new LinkedList<>();
15         StringTokenizer tokenizer = new StringTokenizer(expression, "(?<=[-+*()%/])|(?=[-+
16         while(tokenizer.hasMoreTokens()){
17             String token = tokenizer.nextToken();
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **ExpressionTree.java** (Your program)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin
working on this zyLab.
```


10.2 Lab 18 - Expression Tree - Parse Expression

Module 11: Lab 18: Expression Tree - Parse Expression

Expression Trees

This lab includes the following .java files:

L18/
└── ExpressionTree.java
└── TestCode.java *

*TestCode.java is the *main* in zyBooks.

Here is the starter jar with if you would like to code in a different environment: L18.jar.

Note: the jar has a line "Enter an expression:" so that you can enter in your input after running your code. In zyBooks, this line is removed because you enter in your input before you run your code.

Expression Trees are an important part of computer science because they can help us to evaluate expressions and change the notation of our expression very quickly between postfix, infix and prefix once the tree is built.

IMPORTANT NOTE

You only have to complete the methods described below in the "Completing the Code" section. The rest of it will be done as your programming assignment.

Expression Trees - Overview

The program you will be writing takes an expression in infix notation and then builds a tree out of it. The steps to this process are shown in the diagram below. Your Job is to complete steps 3 and 4

for today's lab, and the other parts will be in programming assignment 10. (In other words, you may modify your code from this lab and use it in PA10.)

1. Get input: reading string which represents infix expression from command line
2. Tokenize input: turn input into a list of tokens (a token is an operand or operator)
3. Convert this infix expression to postfix
4. Build the expression tree from the postfix expression
5. Do tree traversals to get infix, prefix and postfix expressions
6. Evaluate the postfix expression

©zyBooks 12/08/22 22:13 1361995
John Farrell
COLOSTATECS165WakefieldFall2022

infix: "(5+6)*12 - 12/3"

postfix: "5 6 + 12* 12 3 /-"

prefix: "- / 123 * 12+56"

program flow:

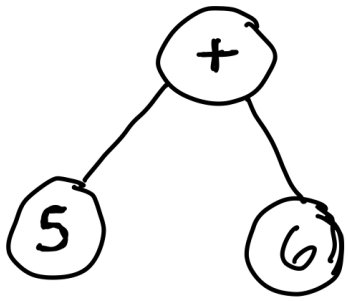
parse "5+6" -> {"5","+","6"}

input — list of tokens

-> convert to postfix: {"5","6","+"}

↳ build reverse list: {"+","6","5"}

-> build tree



=> traverse tree to get postfix, prefix and infix

=> evaluate takes the postfix and evaluates returning the answer.

Completing the Code

Some of the methods are completed for you. The methods have ample description in the comments above the functions. Furthermore, you should make use of the javadocs here: javadocs.

If you need a hint on how to write the convert method, take a look at the shunting-yard algorithm.

You must complete these methods for this Lab:

1. convert()
2. buildRecursive()

Testing the Functionality:

When you have completed the above methods go to the file TestCode.java. Run this class and enter the expression

```
5 + 6 * 9 - 13 / (5 * 6)
```

Your program should print the following:

```
Original Expression: 5 + 6 * 9 - 13 / (5 * 6)
Infix Tokens: [5, +, 6, *, 9, -, 13, /, (, 5, *, 6, )]
Postfix Tokens: [5, 6, 9, *, +, 13, 5, 6, *, /, -]
Build: complete
```

422352.2723990.qx3zqy7

LAB ACTIVITY | 10.2.1: Lab 18 - Expression Tree - Parse Expression | 0 / 7

Downloadable files

`ExpressionTree.java` and `TestCode.java` **Download**

Current file: **ExpressionTree.java** Load default template...

```
1  import java.util.*;
2
3  /**
4   *
5   */
6  public class ExpressionTree{
7      public Node root;
8
9      /**
10      *
11      */
12     public class Node {
13
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **ExpressionTree.java** (Your program)

Program output displayed here

Coding trail of your work What is this?

```
11/1 T----- min:13
```

10.3 Lab 19 - Linked List: A Simple Iterator

Module 9: Lab 19: Linked List - A Simple Iterator

Lab 19: Linked List - A Simple Iterator

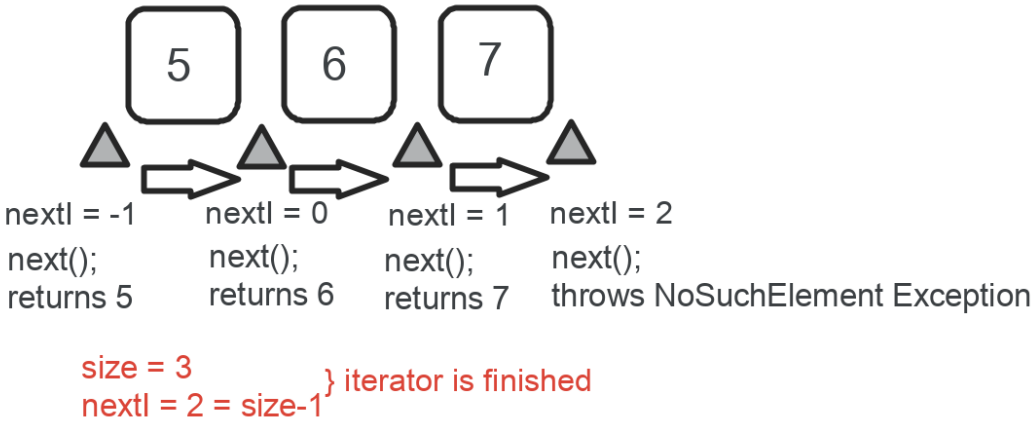
This lab includes the following .java files:

L19/
- MyLinkedList.java
- IteratorTest.java*

* This is the **main** class in zyBooks and the only one that will run.

Note: you may download MyLinkedList.java below and code in your preferred environment. It has its own main that is similar to IteratorTest.java.

Iterators are a very helpful way to move through data structures like Linked Lists because we can process the data one element at a time. In the image below, you will see a simple list with elements 5, 6, and 7 in it. When we call next(), it should first return 5, then 6, and then 7. If we try to call it again, we will get a NoSuchElement exception because we have gone past the end of the list.



For more details on iterators in Java, the Oracle documentation is a good reference.

Completing The Code

This Lab has two parts: first, completing a very simple singly linked list and second, implementing a very simple iterator. To complete the code finish these methods:

1. add()
2. getNode()
3. hasNext()
4. next()

Testing

Your output should look like this:

```
Testing add():
Data:548
Data:122
Data:560
Data:798
Data:523
Data:336
Data:887
Data:594
Data:315
Data:192
size:   expected: 10 -> actual: 10

Testing getNode():
item at index 0:        expected: 1 -> actual: 1
item at index 1:        expected: 2 -> actual: 2
item at index 2:        expected: 3 -> actual: 3

Testing Iterator:
548
122
560
798
523
336
887
594
315
192
size: 10
next Index: 9
You threw the correct error!
```


422352.2723990.qx3zqy7

LAB ACTIVITY 10.3.1: Lab 19 - Linked List: A Simple Iterator 0 / 7

Downloadable files

`MyLinkedList.java` **Download**

Current file: **MyLinkedList.java** Load default template...

```
1  import java.util.List;
2  import java.util.ListIterator;
3  import java.util.NoSuchElementException;
4  import java.util.Random;
5  public class  MyLinkedList<E>{
6      public class Node {
7          E data;
8          Node next;
9          public Node(E data) {
10             this.data = data;
11         }
12         @Override
13         public String toString(){
14             return data.toString();
15         }
16     }
17     Node head;
```


Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **MyLinkedList.java** (Your program) →

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.