Recursive Programs

Overview

Write **two** programs that solve problems which can be solved recursively:

- You must solve the following:
  - Horizontal Ruler, based on the Ruler program covered in class
- Solve **one** of the following:
  - Eight-Queens Problem, based on the ProblemSolver approach
  - Prefix Expression Evaluation, a completely new problem

1. Horizontal Ruler

1.1. Problem Description

In class, we went over a recursive solution to printing an English Ruler with adjustable lengths and number of tick marks. It produced output vertically, like this:

```
Ruler of length 3 with major tick length 3
--- 0
-
--
-
--- 1
-
--
-
--- 2
-
--
-
--- 3
```

For this program, you are to print out rulers horizontally, like this:

```
Ruler of length 3 with major tick length 3
|       |       |       |
|   |   |   |   |   |   |
| | | | | | | | | | | | |
0       1       2       3
```

1.2. Notes

- You must start with the code we went over in class: Ruler.java
- Your solution must be recursive.
- Turn in only your source file: HorizontalRuler.java.
- Make sure your class is not in a package (that is, it is in the *default* package).
- Your output must include your name.
- Your program should run the three test cases that are already in Ruler, that is:
  - Ruler of length 2 with major tick length 4
  - Ruler of length 1 with major tick length 5

    - Ruler of length 3 with major tick length 3
- Don't overthink this! This problem is not that different from the original.
- Hint: Because of the way printing works, you will need to print the top line first (just the tops of the longest tick lines), then the second line (with top two longest tick lines), etc. Finally, you can print the numbers.

1.3. Required Main Class

HorizontalRuler

1.4. Required Input

Not applicable

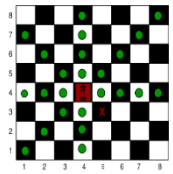
1.5. Required Output

Your output should look like the following:

```
Horizontal Ruler – Your Name

Ruler of length 2 with major tick length 4
|               |               |
|       |       |       |       |
|   |   |   |   |   |   |   |   |
| | | | | | | | | | | | | | | |
0               1               2

Ruler of length 1 with major tick length 5
|                               |
|               |               |
|       |       |       |       |
|   |   |   |   |   |   |   |   |
| | | | | | | | | | | | | | | |
0                               1

Ruler of length 3 with major tick length 3
|       |       |       |
|   |   |   |   |   |   |
| | | | | | | | | | | |
0       1       2       3
```

2. Eight-Queens Problem

2.1. Problem Description



A chess board is an 8x8 matrix on which chess pieces are placed. A real chess game has a variety of pieces that have different rules defining how each can move and whom they can attack. For example, the queen can attack any other piece on the same row, column, or diagonal.

Your program is to find solutions to the "Eight Queens Problem". Place eight queens on a standard chess board such that no queen can attack another. There are 92 unique solutions!

2.2. Notes

- You must use the `PuzzleSolve` class and `PuzzleTest` interface to solve this problem.
- Your solution must be recursive.
- Turn in only your source file: QueensPuzzleSolver.java. It will have an inner class which defines your implementation of `PuzzleTest`.
- Make sure your class is not in a package (that is, it is in the *default* package).
- Your output must include your name.
- Not that you would *ever* look to cheat, but there are many other solutions to this problem online that do not use this approach. Please don't try to turn in one of those!
- Hint: What is your Universe? The Queen positions 1 – 8. Generate all possible combinations of these positions and test each one. Each candidate represents the position of that queen on that row. Therefore, you don't have to worry about queens attacking on rows and columns, that's handled by the nature of your solution. You just need to check diagonals. Determining whether two queens can attach on the diagonal is simpler than you may think. Look at some examples by hand and see if you can determine the simple mathematical relationship.

2.3. Required Main Class

QueensPuzzleSolver

2.4. Required Input

Not applicable

2.5. Required Output

Your output should look like the following:

```
Eight-Queens Puzzle – Your Name

Solution #1
Q . . . . . . .
. . . . Q . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .
```

```
Solution #2
Q . . . . . . .
. . . . . Q . .
. . . . . . . Q
. . Q . . . . .
. . . . . . Q .
. . . Q . . . .
. Q . . . . . .
. . . . Q . . .
```

… The other 89 solutions have been left off …

```
Solution #92
. . . . . . . Q
. . . Q . . . .
Q . . . . . . .
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . . Q .
. . . . Q . . .
```

3. Prefix Expression Evaluation

3.1. Problem Description

Mathematical expressions usually use *infix* notation in which the operator is *in between* the operands: (1 + (2 * 3)). With prefix notation, the operator comes first: (+ 1 (* 2 3)).

You are to write a program that evaluates expressions written in prefix notation. The values will be all integers and the only operators you need to handle are +, -, *, and /, all of which retain their traditional meanings.

Note that with prefix notation, we are not limited to just two operands. That is, we can add any number of values: (+ 1 2 3 4) evaluates to 10. This also works for - and /. (- 10 1 2 3) is interpreted as 10 - 1 - 2 - 3 and evaluates to 4.

Here is the recursive definition of your expressions:

expression: ***integer_value***
or: **(** *operator value_list* **)**

value_list : *expression*
or: *expression value_list*

Operator: **+** or **-** or **/** or *****

3.2. Notes

- Your program can assume that the input is well-formed. For example, there will be no bad operators, floating point values, unmatched parentheses, or missing arguments. All the tokens in the input will be separated by whitespace (space, tab, or newline) which means that you can use `Scanner.getNext()` to get each token.
- Your program is to continue to accept expressions until the user enters "done". Ignore case on this comparison.
- Your output must include your name.
- Your solution must be recursive.
- Turn in only your source file: PrefixEvaluation.java.
- Make sure your class is <u>not</u> in a package (that is, it is in the default package).
- <u>Hint:</u> Consider writing a single recursive method `int eval(String e)` which evaluates e. If e is an integer (the base case), just return its value. If e isn't an integer, it must be start with a "(" (recursive case). Read the operator next, then read, evaluate and store (ArrayList?) expressions until you find the matching ")". Once you have your values (the operands to the operation), perform the required operation and return its result.

3.3. Required Main Class

PrefixEvaluation

3.4. Required Input

A series of prefix expressions followed by "done".

3.5. Required Output

For each expression, evaluate the expression and print the result. You can use the examples below as a set of test cases. Your output should look like the following (input is in GREEN).

```
Prefix Evaluation – Your Name

Enter an expression: 3
Evaluates to 3

Enter an expression: ( + 1 2 )
Evaluates to 3

Enter an expression: ( / 16 3 )
Evaluates to 5

Enter an expression: ( - 10 1 2 3 )
Evaluates to 4

Enter an expression: ( / 120 2 3 4 )
Evaluates to 5

Enter an expression: ( + 1 ( - 12 10 ) 3 ( / 20 5 ) 5 ( * 2 3 ) )
Evaluates to 21

Enter an expression: ( + ( - ( * ( / 12 4 ) 2 ) ( + 3 7 ) ) 10 )
Evaluates to 6

Enter an expression: Done
```