Object Oriented Programming

Assignment 2

Calculator

Marks available: 100

Percentage of overall marks from this assignment: 35%

Date issued: 20/10/20

Deadline: 09/11/20, 17:00

Introduction

For this assignment you will create a simple calculator.

Submission instructions

Submit your work by the deadline above as a **zip** archive. Submit only .java source files in your archive unless told otherwise.

Make absolutely sure you have are not submitting .class files instead of .java files.

Each Java file must have a package declaration at the topic of it (these declarations must be on the very first line of the file). The final part of this declaration varies according to the assignment and exercise. The package declaration for this assignment is:

```
package com.bham.pij.assignments.calculator;
```

Do not copy and paste this package declaration from this pdf! You will get unexpected characters in your files that will render them untestable. Type it in.

Your package declaration must be on the very first line of your file. Do not put anything above the package declaration, not even comments.

Do not forget to add the semicolon at the end of the package declaration. All characters in a package name should be lower case.

In each of these exercises, you are not restricted to only creating the required methods but you must create **at least** the required methods.

All of the required methods must be `public`.

None of the required methods should be declared as `static`.

Do not use the `Arrays` class or any other class from the Java Collections Framework, apart from `ArrayList` which you **can** use.

Do not use any class, other than the one you are writing, that can evaluate mathematical expressions.

Do not use non-ASCII characters anywhere in your files, even as comments.

All strings can be treated as case insensitive unless otherwise stated.

IMPORTANT: Do NOT put any code that gets **user input** in the required methods below. If you do, your work can't be tested. This means that you should not use `Scanner` in any required method or in any method called by a required method, etc. Please ask if unsure about this.

Introduction

For this assignment you will create a calculator application. Each exercise involves adding more functionality to the calculator. You are advised to do the exercises in the order in which they are listed.

You need to create the `Calculator` class yourself. There is no prepared class for you to download for this assignment. The assignment requires only that one class. If you choose to create other classes that is up to you but not necessary. Doing that also risks over-complicating your program leading to errors.

The requirements for the assignment only involve you creating the class and the required functionality. However, you will need to also create a `main` method in your class so that you can actually run the program yourself to test it.

The only validation that is required is stated in the exercises below.

As you do each exercise, you are strongly recommended to check that the functionality you added in the *previous* exercise still works. This is called `regression testing`.

Exercise 1: A Basic Calculator [50 marks]

For this exercise you will create a `Calculator` class and add some basic functionality to it.

The Basic Calculator is able to evaluate simple arithmetical expressions involving integer and floating point operands and the operators: $*$, /,+,−. The expressions to be evaluated will be entered at the keyboard. The result will be printed to the console.

The functionality of the basic calculator is as follows.

Evaluating simple arithmetical expressions

The calculator can evaluate expressions of the following form:

operand[space]operator[space]operand

For example, $3 + 4$, $2 * 10.5$, $10/5$, $39.255 - 47$.

Thus, the operands are any valid integer or floating point number (in `float` format) and the operator is one of $*$, $/$,$+$ or $-$. There is a space character between each operand and the operator.

For the Basic Calculator, if the user enters an expression that is not precisely of the form shown above then the program should output "Invalid input.".

If the user enters an expression involving a 'divide-by-zero' error then the program should also output "Invalid input." in that case too.

The `Calculator` class

You must create a class called `Calculator`. The `Calculator` class must have the following public methods:

```
public Calculator()
```

The default Constructor.

```
public float evaluate(String expression)
```

This is the most important method in the class. This method receives a String containing an expression (in the format described above), evaluates the expression and returns the result as a `float`.

If the String `expression` does not contain a valid expression (as described above) or contains a divide-by-zero attempt then this method should return the value `Float.MIN_VALUE`.

The following requirement is very important. Do not use the Scanner or other input method in the `evaluate()` method or in any method that it calls.

You should also include the following *get* method in your class:

```
public float getCurrentValue()
```

This method returns the current value (result) that the calculator has evaluated. For example, if getCurrentValue() is called after the user has entered the expression $3 + 4$, it would return the value 7. If the user has not entered a valid expression, as defined above, this method should return zero.

Exercise 2: Memory and History [15 marks]

In this exercise you will expand the functionality of the calculator to include memory and history functions.

The memory function works as follows. If the user presses presses the 'm' key the program should store the most recent calculator result (if it was valid, zero otherwise).

If the user types "mr", the program should print to the console the stored memory value (or zero if no value has been stored).

If the user presses the 'c' key, the program should clear the memory by setting it to zero.

You should also modify your `evaluate()` method so that it can use the memory value, as follows.

If the user wishes to use the memory value in a calculation, they can use a shortened form of input, for example:

+4
∗11

The format of these expressions is *operator[space]operand*.

If the memory value is m, in the case of the above two expressions, the calculator would compute $m+4$ and $m*11$. For example, if the previous result was 10 and the user pressed the 'm' key, for the two expressions above the output would be 14 and 110 respectively.

For this exercise you should also add the following *get* and *set* methods to your `Calculator` class, and the `clearMemory()` method.

```
public float getMemoryValue()
public void setMemoryValue(float memval)
public void clearMemory()
```
[1].

The calculator should also have a history function which works as follows. If the user presses 'h' then the program should print to the console all of the results since the program was started, on one line, separated by spaces.

You should also add the following method to your class. This method returns the history value at the index in the parameter. If the parameter is zero, the method should return the first value that was added to the history. If the parameter is 1, the method should return the second value that was added to the history, and so on.

```
public float getHistoryValue(int index)
```

Exercise 3: Basic bracket functionality [15 marks]

For this exercise you should expand your `evaluate()` method so that it can evaluate expressions of the following form:

$(3.5 + 4.5) + (2 - 1)$

Other examples:

[1] Of course, this method is equivalent to `setMemoryValue(0)`

$(2 * 10) - (3.85 + 12)$

$(3 * 12)/(3 + 3)$

The terms within the brackets should be separated by spaces as for Exercise 1. Brackets should **not** be separated from their adjacent number by a space. The operator between the brackets **should** be separated from the brackets by a space. That is:

(oper1[sp][op1][sp]oper2)[sp]op2[sp](oper3[sp][op3][sp]oper4)[2]

Where:

op{n} is an operator.
oper{n} is an operand.
[sp] is a space character.

No other form of expression involving brackets needs to be handled[3].

Exercise 4: Arbitrary length expressions [20 marks]

For this exercise you should expand your `evaluate()` method so that it can handle arbitrary length expressions (without brackets). This will mean you will now need to take account of the order of precedence of arithmetic operations.

The method should be able to correctly handle (using order of precedence) expressions like the following:

$3 + 4 * 6$

$3 * 0 + 2/6$

$12 - 2 * 10 + 2 * 6/2$

The spacing rules are the same as for Exercise 1.

[2] Apologies: this was the only format in which I could get this on one line. Look at the examples instead if this is confusing.

[3] Yes, this represents quite limited bracket functionality but handling arbitrary brackets is tricky. You can add arbitrary bracket functionality if you wish but there are no marks for that and it won't be tested