1 About this Assignment

1.1 Objectives of the assignment

We have quite a few things on our plate this week. Our objectives with this assignment are

1. to get more experience with String manipulation and arithmetic expression evaluation;
2. to learn to implement methods
3. learn to write simple conditional statements and loops;
4. start using the Eclipse integrated debugger.

This may seem like a lot but for the most part it is a continuation of last week’s lab. You learned how to extract elements of a string. This time you will do it in a context where the elements to extract actually mean something. The only really new element is the introduction of loops and conditional statements. For the time being, just get used to them. We will spend time in class in the next couple of weeks to study their mechanism more in depth.

Read the assignment completely through once. Do not stop at the first unclear sentence you encounter; sometimes things are explained and detailed a bit later. Then you should start all over again. This time stop to ask questions when a point remains fuzzy, code your solution, and move on to the next section. Before you hand in your project, make sure that you have completed all the tasks listed in this assignment and that your project folder contains all the files you are supposed to hand in, and only these files (i.e. don’t forget to “cleanup” your project folder).

1.2 Handouts

If everything works out fine, we might be able to download the handouts from the web site.

The single handout for this second lab assignment is a pdf (Acrobat) containing the present document (no code handout today).
2 What to Do, Part I: Analyze the Problem

2.1 Objective of the program

The program you are going to develop this week will evaluate expressions of the form

\[(a + b + c) \times (d + e + f + g) \times (h + i + j)\]  \hspace{1cm} (1)

where \(a, b, c, d, e, f, g, h, i,\) and \(j\) are integer literals, and arbitrary numbers of space characters (possibly none) can be added to pad around any of the symbols in the expression. Examples of expressions you would have to evaluate are:

- \((10 + 5 + 2) \times (0 + 1 + 0) \times (2 + 1 + 1)\)
- \((5 + 2 + 4) \times (1 + 2 + 1) \times (1 + 1 + 1)\)
- \((1 + 1 + 2) \times (2 + 3 + 5) \times (0 + 1 + 1)\)

The values your program should return for the above expressions are 68, 132, and 40 respectively. Your main task in this assignment is to implement the function that evaluates expressions in the above format.

2.2 The way to proceed

This problem is a bit more complex than what we did last week, but you can probably still manage to solve it by writing the same kind of code that you wrote for the first lab. On the other hand, you are going to find very soon that this style of programming will not lead you very far, so you are better off trying as soon as possible to learn to proper way to solve this type of problem. Even if it seems to be more work at first, you will see in the long run that you will save a lot of time and effort by working that way.

It is very important that you don’t try to write a single piece of code that solves the whole problem. instead, you should try to break your problem into a set of smaller problem, and then implement solutions to these simpler problems under the form of methods. This approach, which consists in dividing the original problem into smaller ones, until we arrive at a size of problem that we can code easily, is called divide and conquer. Of course, as you become more experienced as programmers, the size of problem that you consider easy to solve in one shot will increase, but at the moment, we are better off starting small.

2.3 Looking at the problem to solve

If you look more carefully at the generic form of the expression, (1), you will notice that it is the product of three terms of the form

\[(a + b + c)\]  \hspace{1cm} (2)
What that means is that in order to evaluate the expression, what we have to do is:

1. divide the expression into three terms of the form in Eq.(2): $term_1$, $term_2$, $term_3$.
2. evaluate our three terms to get three values, $val_1$ and $val_2$, $val_3$.
3. compute the product of $val_1$, $val_2$, and $val_3$.

Now, if you look at what has to be done in Step 2, you will see that we are going to be doing exactly the same thing for each of the 3 terms: given a string that stores an expression of the form in Eq.(2), extract the first and the second number from the string, and compute their sum. There are basically two ways to proceed when one has to repeat several times the same operations for different input data.

The first way consists in repeating the code that accomplishes the operation, as many times as needed, possibly with changes in the names of the variables used. This solution has several drawbacks. First, it makes your code unnecessarily long. Second, experience will teach you that “cut & paste with a few minor edits” is an error-prone strategy (there is always one variable that you will forget to rename). Finally, it is not elegant.

The other way is write in your class a method that accomplishes the task to perform, and invoke this method for your different data sets. Let’s revisit the “evaluate term” problem. For this problem, you expect to be dealing with a string that starts with ‘(‘, possibly followed by some space characters, followed by digit characters, possibly followed by some space characters, followed by a ‘+’ sign, possibly followed by some space characters, followed by digit characters, possibly followed by some space characters, followed by the ‘)’ character. What you have to do with this string is to extract its three substrings that encode numbers, convert these three substring to their numerical value, and add these values to get the value of your term. The operations I just described are exactly the same for any “term” string. So we are going to write a method that takes a string as its input and processes it:

```java
public static int evaluateTerm(String termString) {
    // where to store the value of the three numbers
    int number1, number2, number3;

    // somehow extract the value of the two numbers
    // from the string

    return number1 + number2 + number3;
}
```
If your method was properly written, then I should be able to invoke your method by sending it a properly formatted string, and get in return the value of the expression stored in that string. So, if I write

```java
String myString = "(7 + 1+ 4 )";
int myVal = evaluateTerm(myString);
```

then the value of `myVal` after this code fragment has been executed should be 12.

**Project 1.** Implement a method that takes as parameter a `String` in format (2) and returns as an `int` the value of that expression.

### 2.4 Evaluate the whole expression

Now that you know how to write a method that evaluates a simple term, you are going to use it in another method that evaluates the whole expression.

**Project 2.** Implement a method that takes as parameter a `String` in format (1) and returns as an `int` the value of that expression. This method must invoke the method described in the above task assignment.

**Report 1.** Try to explain in simple words (and in plain English) how you would proceed if you had to evaluate an expression that contains a large number of factored subexpressions. We have not seen yet the way to do this in Java, so a complete, working solution is not what is expected from you. What I would like you to come up with is a rudimentary algorithm for this problem.

Please note that a bunch of code or pseudo code, or anything that contains techno-babble or sentences such as “basically I would write a while loop that blah blah blah” are not valid answers for this part of the assignment.

**Report 2.** You should not need a reminder to do this, but make sure to discuss possible limitations of your program.

### 2.5 Correctly format your output

This is different from code formatting. Here what you have to do is fix the code of your `print` and `println` statements so that what is printed by your program in the console conforms to the following format (given in the form of an example). Let’s assume that the input string entered by the user is

```
(8+ 2 + 5 ) *(0+2+1) *( 1+1 + 0 )
```
Then your program should print the following in the console:

```
The expression you entered is
   (8 + 2 + 5) * (0 + 2 + 1) * (1 + 1 + 0)

The value of this expression is 90.

Bye!
```

Look carefully at the above example. Note that the expression you print had all extra space characters removed, except for 1 padding space character around each operator sign ‘+’ and ‘*’.

---

**Project 3.** Modify the code of your program so that your output now conforms to what is shown in the above example.

---

### 3 The Eclipse Integrated Debugger

#### 3.1 How to activate the debugger

Until now we have only used “run” the application, either using the entry in the “Run” menu or by clicking on the run button, but today we will experiment with debugging. To do so, you need to change the “perspective” of your project. In the “Window” menu, entry “Open perspective”, select “Debug” (Figure 1)

![Image of Eclipse window with Open Perspective menu selected](image.png)

Figure 1: Change to the debugging perspective.

The layout of the project window changes considerably when you switch to the “debug” perspective. For example, you will certainly notice the row of buttons at the top of the “Debug”
These control buttons will allow us to execute our code step by step (step over), to execute a block of code in one step (resume) or to “step into” a method that is invoked.

If you scroll down in the window that shows your source code, you will notice a big colored dot next at the line of code that initializes the variable \texttt{NbOfDays1} (Figure 3). Eclipse may insert by default such a breakpoint at the first line of executable code of your program (the lines that precede this one are either comments or variable declarations).

If you click on the “Debug” button, this will launch the execution of the application in the debugger, and this execution will stop at the first breakpoint it encounters. If you look carefully at the source window, you may notice a small arrow drawn on top of the breakpoint (Figure 4). This small arrow is the debugger’s way of indicating the line of code that is about to be executed next.

Now, click once on the “step over button” (see Figure 2). The arrow will advance to the next line of your code (Figure 5(a)).

The important thing to notice there is that the assignment statement that has just been executed is reflected in the “Variables” window (which should be by default in the upper right corner of the workspace). This window now displays the name of the variable that has been initialized and the value of that variable (Figure 5(b)).

If we click once more on the “step over” button, the arrows go one step further and another variable appears in the “Variables” window (Figure 6).

\footnote{I know that this is a bad identifier. This was done on purpose in an older assignment. Really.}
Figure 4: Execution has stopped at the first breakpoint. Notice the arrow drawn over the breakpoint.

Figure 5: After one “step over”: (a) The arrow has moved to the next line; (b) The variable initialized and its value appear in the “Variables” window.

There are basically two ways to execute code while in the debugger: You can either execute it step by step, as we have been doing, or you can execute a block of lines by jumping to the next breakpoint you have selected in the source file. To add a breakpoint to the program, all you have to do is right-click (control-click for a Mac) on the line number for the line where you want to set the breakpoint, and then select “Toggle breakpoint”, as shown in Figure 7.

Figure 6: Status after to “step over” clicks.
Figure 7: Set a new breakpoint at a line where you want execution to stop.

Having set the next breakpoint, you can now execute in one step all lines of code between the current one and the next breakpoint. To do so, click on the “resume” button (Figure 8).

Figure 8: The “resume” button.

After you have executed this block of code, you will notice that all the variables that have been initialized or been assigned new values are now listed in the “Variables” window (Figure 9).

3.2 Return to the “Java perspective”

To do this, you can select the “Java perspective” from the “Window” menu, as we did earlier to select the “Debug perspective”. Alternatively, you can also click on the Java button in the upper-left corner of your workspace (Figure 10).
4 What to Do, Part II: Conditional Statements and Loops

4.1 Multiple evaluations

If you want to evaluate multiple expressions you can of course repeat multiple times the same block of code that

- asks the user to enter an expression;
- computes the value of the expression;
- outputs the value of the expression

As you do this you will be happy that you used methods for your implementation because that means that the you have much less code to repeat. Still, this approach is not good because

- you need to know in advance how many expressions you want to evaluate. If you want to evaluate one more, you need to add source code to your program and recompile it.
- whenever you cut-and-paste code and modify the new copy, the odds are pretty high that you will forget one small change. Such errors are very hard to spot.

What we will do instead is use a loop that will execute multiple times the code we want to repeat.
4.2 The while loop

First add the following block of code to the end of your program

```java
int counter = 0;
while (true) {
    counter = counter + 1;
    JOptionPane.showMessageDialog(null, "number of times in the loop: " + counter);
}
```

What this statement says is that the block of code between the curly braces should be repeated as long as the expression between parentheses following the keyword `while` evaluates as `true`. Since that expression is `true`, what we get is an infinite loop.

The first thing that I would like you to do is to modify the above code to run the loop only a given number of times. For this, you are going to use the variable `counter` for more than display. You can produce a logical expression that evaluates as `true` or as `false` by comparing numbers. For example:

- `counter == 4` evaluates as `true` if the value stored in `counter` is equal to the value of 4, that is, 4; It evaluates as `false` otherwise.
- `counter != 4` evaluates as `true` if the value stored in `counter` is different from 4; It evaluates as `false` if the value of `counter` is 4.
- `counter > myNumber` evaluates as `true` if the value of `counter` is strictly greater than the value of `myNumber`; It evaluates as `false` otherwise.
- `2 * counter >= myNumber` evaluates as `true` if the value of `counter` multiplied by two is strictly greater than the value of `myNumber`; It evaluates as `false` otherwise.

Similarly, the operator `<` tests whether the value of the expression on the left side is strictly less than the value of the expression on the right side. The operators `<=` tests whether the value of the expression on the left side is less than or equal to the value of the expression on the right side. Please note that, once more, Java deals with the values of expressions.

| Project 4. Add an input dialog that asks the user how many times the loop should be executed and make sure that your program exits the loop after the right number of times. |

4.3 Integrate the loop to your expression evaluation code

| Project 5. Modify the code of your main method so that the user is asked first how many expressions should be evaluated. Then the program should repeat as many times as the user chose to (1) ask for an expression, (2) evaluate the expression, (3) display the value of the expression. |
4.4 For extra credit (6 pts)

Another form of conditional statement is the `if` statement. One form of this statement is

```
if (condition)
  statement1;
else
  statement2;
```

For example, the following block of code contains a valid `if` statement

```
int n = 4;

// interesting things happen to n inbetween

boolean nHasChanged;
if (n == 4)
  nHasChanged = false;
else
  nHasChanged = true;
```

Project 6. Modify the code of your main method so that if the user enters “end” as one of his/her expressions, then the loop will simply terminate, even if the number of expressions initially specified has not been reached. If the user enters “pass” then no evaluation is performed for this input, but the loop continues its execution until the required number of expressions has been entered.

5 What to Hand in

5.1 End-of-session evaluation

You are not expected to complete the assignment by the end of the lab session, but you are definitely expected to have done some work during that session. Try to use the lab session to make sure you understand everything about the assignment. Ask questions; try things; ask more questions.

You should not leave the lab before your work has been evaluated. This first evaluation is worth 10 pts out of 100 for the complete assignment. If you leave before you have been evaluated, these points are lost with no chance of a later evaluation.

5.2 Your project

Next week, you will be asked to upload to your account on the EnVision server a cleaned-up project folder containing
1. a working project file (.mcp),
2. your project's source file (.java),
3. your report file.

The details of the operation will be discussed next week (for one thing, they are still unknown at the time of this writing).

5.3 Printed copy of the report

You should hand in a printed copy of your report at the beginning of the next lab session. If your report is not ready at the beginning of the session, a late penalty (never less than one day, or 10%) will be applied. There is no use typing the report during the lab since the penalty is the same whether you return the report at the end of the lab or the next day at the beginning of the class.

6 How You Will Be Evaluated

6.1 Point distribution

The maximum number of points is 100, but extra points could be awarded for excellent aspects of the project or report. The point distribution for this assignment is as follows:

**Execution evaluation**
- End-of-session evaluation: 10 pts
- Execution of the project handed in: 30 pts

**Source Code**
- Identifier names: 12 pts
- Good indentation and general readability: 13 pts
- Judicious comments well positioned in the code: 15 pts

**Report**
- Discussion of important aspects of the project: 10 pts
- General quality of the writing and presentation: 10 pts

6.2 Various point penalties

- Project left accessible on the workstation: -5 pts
- Project folder incomplete or not properly cleaned up: -5 pts
- Report file missing from the project folder: -5 pts

**Late penalties**
- Per day late (only complete assignments are graded): -10%

If you submit a project late, then it is your duty and job to notify the TA (with Cc. to the instructor) that the project is finally available for download on the EnVision server. If you fail to do so, then the “late penalty clock” will keep ticking until the TA gets around to checking your folder on the EnVision server and notices your project. Unless specifically asked to do so, do not mail your project folder as an attachment.