# Comp 401-001 - Assignment 1: Writing a Number Scanner

**Date Assigned: Tue Aug 18, 2015**

**Completion Date: Fri Aug 28, 2015 (11:55 pm)**

**Early Submission Date: Wed Aug 26, 2015 (11:55 pm)**

In this assignment, you will revise your programming skills by doing an assignment involving the use of many of the concepts that are a pre-requisite for this course, which include loops, and methods (procedures). In addition, you will learn how to scan a string.

| | | | | | |
|---|---|---|---|---|---|
| JDK Download | PowerPoint<br>PDF | | | | |
| Eclipse Install & Use | PowerPoint<br>PDF | PDF | | | |
| Checkstyle with UNC Checks Install | PowerPoint<br>PDF | | | | |
| Debugging in Eclipse | PowerPoint<br>PDF | PDF | | | |
| Relevant Java Syntax | PowerPoint<br>PDF | PDF | | | lectures.java_syntax_overview Package |
| Scanning | PowerPoint<br>PDF<br>YouTube<br>Mix | Docx<br>PDF<br>Drive | Scanning Visualization | Number Scanner | lectures.scanning Package |

## Assignment Specification

Write a Java program that processes input lines until the user enters a terminating line beginning with the character '.'. Each of the other lines is expected to be a string consisting of digits and space characters. The program scans the string, converts each digit sequence into a number and prints these numbers and their sum and product. It does this for each line other than the terminating line.

Thus, the tokens produced by your scanner are numbers. Later your program will recognize other kinds of tokens such as words and quote characters.

You can assume that there:

1. is exactly one space character after each token (including the last one).
2. is at least one token in each input line.
3. a line consists of only spaces and digits.

The following example illustrates the nature of the expected program behavior, where the input is in a special color:

2 3 0100
Numbers: 2 3 100 Sum: 105 Product: 600
4 6 20
Numbers: 4 6 20 Sum: 30 Product: 480

Though in this example we have only three tokens on each line, in general, input lines can have a variable number of tokens. If you decide to use arrays in this assignment, you can assume a maximum number of tokens. You can also assume at least one token on each line.

As your string consists of only space and digits, you do not have to worry about negative numbers. Thus, '-' is an illegal character.

You do not have to do any error checking. Thus, your program can terminate abnormally if the user does not follow these constraints.

You can concatenate strings using the + operation as in:

String helloWorld = "hello" + "world";
System.out.println ("The string is: " + helloWorld);

## Extra Credit

1. Allow a token to be succeeded or preceded by a variable number of blanks as in the input string "  2  3  0100  ". You should not scan the string multiple times.
2. Do not terminate the program after encountering the first illegal (unexpected) character. Print the illegal character and continue scanning assuming the character had not been input. How you recover from or report the errors is up to you – for example, you can break up 123.23 into two legal numbers, 123 and 23, or a single token, 12323. Similarly you can report errors as they occur or report them together. We will not answer questions about nature of error reporting.
3. Make your scanner a separate class responsible only for detecting tokens. Ideally, the class should not store tokens (for e.g. in an array or array list). Hint: if you know the

Iterator interface, make your class implement this interface. We have not studied interfaces so far, so this is for students who already know interfaces.

## Constraints

1. Java has libraries that make the writing of this program trivial. Therefore, we will place constraints on what aspects of Java you can use. You can use any of the Java features given in the section on scanning and "Conventional programming in Java for those who know conventional programming (Look on your own)". The only library functions you should use are the Character.isDigit(), Scanner.nextLine(), substring(), charAt(), length() and the Integer.parseInt() functions. Character.isDigit() is like Character.isUppercase() except that it tells us whether a character is a digit rather than whether it is an uppercase letter. substring(), charAt(), and length(), applicable to any string, as explained in the class material. Intreger.parseInt() takes a string consisting of digits and converts into an integer. Thus, Integer.parseInt("100") returns the integer 100. It assumes that the number represented by the string is small enough to be stored as an int. Your solution should make the same assumption. You will the Scanner class to read lines; make sure you use only the nextLine() method~~s~~. You essentially have to make sure you do not use any function that automatically breaks the line into tokens- which is the task your code has to perform In this and all other assignments, check with us before you use some Java feature not presented in the class material. *Check with us only if you ~~think~~ explain why you think it is impossible to do the assignment with the constraints imposed on you, otherwise it needlessly increases our work. If you do not provide such an explanation, we may not respond to your post.*
2. You should not scan the string multiple times, that is, loop through the string more than once. If you determine the indices of the start and end of a token and then call substring to get the token, this is not scanning twice. Determining the indices is scanning, the rest is simply data copy, which you cannot avoid.
3. You should decompose your program into at least two methods, that is, you should not write a monolithic main method. This will demonstrate your ability to write and call methods. The more meaningful methods you write, the more points you will get. If you write a method that simply calls another method and takes no other step, you will get no credit. The main method counts as one of the two methods.
4. You should not use arrays or array lists unless doing so make your program more modular. Arrays are preferable to array lists as they have less baggage and make your coding more challenging. Both of them will increase the footprint (memory size) of your program, hence the modularity requirement. If you use an array list, you can call any of the methods defined by it such as get() and size().
5. Subsequent assignments will teach you to create a multi class program, but for this assignment it is sufficient to create a single class. However, if you feel comfortable doing so, do try and create multiple classes.
6. If we have not specified some aspects of the extra credit or regular work, you are free to choose your own interpretation and please do not ask for clarification as there will be none to give.

**Formatted:** Font: Times New Roman

**Formatted:** Font: Italic

**Formatted:** Font: Italic

6.7. Do not use a "star" import such as import java.util.*, as that will include many banned imports.

7.8. Follow the programming practices and principles you have learned so far on how to write elegant code. In particular, do not write obscure code, but if you do, explain it through comments; and do not comment for the sake of commenting. A program without comments is acceptable as long as you think it would be clear to the graders. You should follow the case conventions.

## Possible Hotswap Error Because of Infinite Loop

This program reads input in a loop until the user enters a special "end of input" line. While testing your program, you may not actually enter this special line, and start editing the previously run program while the program is waiting for the next input. This will result in a "hot swap failed" error from Eclipse. To prevent this from happening, see the Eclipse install slides on how to terminate a program

## Algorithm

Read this only if you have trouble developing your own algorithm. The small print is encouraging you to first think of the problem on your own.

The class example kept track of a single marker in a single loop as the size of the token was constant (1). Now you will need to also determine the length of the token, leading to a nested loop. You will need to store the token in a string before you call parseInt(). You could store the string and/or the corresponding integer in an array or ArrayList (in case you know about ArrayLists). However, this approach results in more space being used as the computer must allocate space for the maximum number of tokens in a line rather than a single token. The advantage of this approach is that the program can become more modular. The array could be shared by multiple methods (as a global static variable or a parameter). One method could scan the tokens in a line, another could find the sum (product) and yet another could print the results.

## Submission Instructions

The submission instructions for this and future assignments:

1. Upload the assignment project directory in Sakai. It should include the source and binary folders. You will need to zip it to upload it.
2. Copy the rubrick into a text document and fFill -and upload the rubrick for the assignment indicating the (regular and extra credit) features you have implemented correctly. The rubrick will be posted as soon as possible and before the early submission date. Call the file rubrick.txt. You can fill the rubrick by simply deleting what you did not do or put notes next to each item explaining the degree to which you did each part.
3. You are also responsible for demonstrating to us that that these features do work correctly. You should do so by submitting electronic static screen shots of the console window (on Windows, the Snipping tool is a great way to capture screens). The screenshots should show that you have implemented all the regular and extra credit

features in the assignment. These should be sufficient to tell the grader that the program works - they should not have to run the program to verify this. If you do not show evidence that a feature works, we will assume that it does not. Put these in a folder and upload it in Sakai. Provide as many screenshots as you need o show your work - one may be enough if you have good error recovery. Each screenshot should be uploaded as a separate file. The files should not be in the project folder.

4. To make it easy for the TAs and the grader program to find your main class, put it in a package called main, and call it Assignment1. In general, for assignment N, name the main class as AssignmentN, and put it in the main package. Other classes could be in the same package or a separate package. If you are using CheckStyle, in the Infos section of Problems, you should see a message of the form: classDefined: (Assignment1.java:1) Class matching main.Assignment(.*) defined, unmatched expected classes/tags []

*Good luck with your first 401 program!*