Comp 401 - Assignment 1: Writing a Number Scanner

Date Assigned: Tue Aug 20, 2013
Completion Date: Fri Aug 30, 2013 (11:59 pm)
Early Submission Date: Wed Aug 28, 2013 (11:59 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.

Resources relevant to this assignment are reproduced below except that you can ignore the material on main args.

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.

<table>
<thead>
<tr>
<th>110 and 401</th>
<th>Eclipse Install and Basic Use (Look on your own)</th>
<th>PowerPoint</th>
<th>PDF</th>
<th>Warm-up Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Debugging in Eclipse (Look on your own)</td>
<td>PowerPoint</td>
<td>PDF</td>
<td>Warm-up Chapter</td>
</tr>
<tr>
<td>401</td>
<td>Conventional programming in Java for those who know conventional programming (Look on your own)</td>
<td>PowerPoint</td>
<td>PDF</td>
<td>Warm-up Chapter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>lectures.java</td>
</tr>
</tbody>
</table>
Assignment Specification

Write a Java program that processes input lines until the user enters a 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.

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.

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 studies 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 “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(), 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. Integer.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. The class material also includes several ways to read a line of input. The course material gives you two ways to read input—one that involves dealing with input and one that does not. You can use either. If you use the Scanner class, make sure you use only the nextString() or nextLine() methods. If your version of Java does not have this method, simply use the other, older, approach to reading lines in the class material. You essentially have to manually 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 it is impossible to do the assignment with the constraints imposed on you, otherwise it needlessly increases our work.
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.

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.

7. 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.

---

**Submission Instructions**

The submission instructions for this and future assignments:

1. Upload the assignment directory in Sakai. The assignment folder will be created and linked as soon as soon as the TAs are assigned.

2. Fill the rubric for the assignment indicating the (regular and extra credit) features you have implemented correctly. Again, the rubric should be posted as soon as possible and before the early submission date.

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 (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.

4. To make it easy for the TAs 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.
**Good luck with your first 401 program!**

**Extra Credit**

Allow (a) a number to be succeeded or preceded by a variable number of blanks as in “2 456 25 3000”, (b) an arbitrary number of numbers in a line. 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 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.

**Submission Instructions**

The submission instructions for this and future assignments:

1. Submit source code of your class(es) and the debugging screen shots, and the screen shots showing executions of the program on test data. In general, you should submit screenshots that 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.

2. Upload the assignment directory, as explained in the instructions linked from the course page.

3. To make it easy for the TAs 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.

*Good luck with your first 401 program!*