Comp 401 - Assignment 2: Object-Oriented Scanning for Numbers, Words, and Quoted Strings

Date Assigned: Thu Aug 27, 2015

Completion Date: Fri Sep 4, 2015

Early Submission Date: Wed Sep 2, 2015

This work will build on your previous assignment. The assignment has been broken into two parts to allow you to build on your previous code incrementally. You can do both parts together.

In the first part, you will get more practice scanning. This is the complicated part. The second part will involve creating properties in a manner that pretty much mimics what was done in class.

We have covered enough so far to do part of the next assignment, which is given in the looking ahead part.

Relevant new class material for this assignment:

Class Dual Role	PowerPoint PDF YouTube Mix	Docx PDF Drive		Number And Word Scanner Bean	lectures.state.properties Package

Part 1: Word and Quoted Strings

Extend your Java program of the previous assignment to print a greater variety of tokens. So far you were recognizing and printing tokens that were digit sequences. You should also recognize and print the following two token types:

- *Word*: a sequence of uppercase and lower case letters.
- *Quoted string*: a sequence of characters enclosed in double quotes.

You can assume that the first character of the scanned string is a letter, digit, or quote character, and that each token (number/word/quoted string) is followed by exactly one blank.

You can also assume that an opening double quote will always be matched by a closing double quote. As before, you can halt the program on encountering something unexpected such as an illegal character or the end of the string without a matching double quote. However, as before, you must print all tokens before the first error is encountered, and if you are doing extra credit, continue scanning, if possible, after the first error.

This time, you do not have to compute a sum or product. Instead, before you print a token, give an indication of its type (word, number, or quoted string). Thus, if an input line is:

move Arthur 30 40 say ”Quest?” say “2” ”$%!@”

You should print the following tokens for this input line:

word: move

word: Arthur

number: 30

number: 40

word: say

quoted string: Quest?

word: say

quoted string: 2

quoted string: $%!@

To make automatic grading robust, follow this output format, making sure each token is output on a separate line, the token descriptor is one of word:, number: and quoted string:, and there is a space between the token descriptor and the succeeding token.

This is the output for a particular input line. As before, the program should accept multiple input lines, ending with a “.”.

In a program, you can enclose a double quote simply in single quote characters as in ‘”’ to create a character value representing the double quote.

As you also see in the example above, all characters within quotes are to be printed – you should not, for instance, remove spaces.

Copying the example string directly into the Eclipse window can cause problems, so you will have to rewrite it.

As in the first assignment, you can test this part by writing a single monolithic main class.

You can concatenate strings using the + operation as in:

String helloWorld = “hello” + “world”;

System.out.println (“The string is: “ + helloWorld);

Part 2: Scanner Bean Class

Create a separate Bean class (a class that has one or more properties – see the reading material for the definition of a Bean and a property) that defines a single instance stored editable String property named ScannedString for storing the input string and scanning it. Thus, the class has both a getter and setter instance method for this property and an instance variable (whose name does not matter) that is read and written, respectively, by the getter and setter. The setter method not only sets the value of the property (by assigning it to the instance variable) but also scans its string argument (not the input, which is read by main) and prints the tokens in this string in the manner described in part 1. (The actual code for scanning the token and printing the tokens can be in this class or a separate class. Thus, if you did extra credit to create an iterating scanner, the bean class would call the code in the iterator.) The setter does not store the tokens, it changes only the variable holding the scanned string. Later, you will modify this class to create a dependent read-only array property for storing the tokens.

The main class now instantiates the Bean class to create a scanner Bean object. It does not directly scan each input line or print the tokens in it. Instead it simply assigns the input line to the editable property of the Bean object by calling the setter method with the line read, which results in the tokens in the line being printed. Thus, the behavior of Part 1 and Part 2 are the same from the point of view of the user providing input and viewing output. The difference is in the way the code is implemented. The getter method is not called by main or any other code in your program. In this assignment, its purpose is to teach you how to create a reusable Bean. In later assignments, we will make use of it.

In the Bean class, you should import the following classes:

```
import util.annotations.EditablePropertyNames;
import util.annotations.PropertyNames;
import util.annotations.StructurePattern;
import util.annotations.StructurePatternNames;
import util.annotations.Tags;
```

Right before the declaration of the Bean class, you should put the following annotations:

```
@Tags({"ScannerBean"})
@StructurePattern(StructurePatternNames.BEAN_PATTERN)
@PropertyNames({"ScannedString"})
@EditablePropertyNames({"ScannedString"})
```

These will allow CheckStyle and our grader program to ensure you are meeting the requirements. These imports require you to associate oeall22.jar with the project. (Right click project, select properties, Build path, add external jar) .

Constraints

1. All of the constraints of the previous assignment apply unless they are overridden below.

2. You can use all of the Java features allowed in the previous assignment and those taught so far in the lectures and readings. In addition you can use the Character.isLetter() method. Character.isLetter() is like Character.isUppercase() except that it tells us whether a character is a letter rather than whether it is an uppercase letter.

3. You can have an arbitrary number of non-public methods in the Bean class. The specified property determines only the public methods in the class.

4. It will be best to create a new project for each assignment, which starts off as a copy of the last assignment, and is then modified to add new behavior. You can just copy the project folder, give it a new name, and then ask Java to create an existing Java project, as described in the slide deck on installing and using Eclipse.

5. Follow the package naming conventions from assignment 1 to ensure you do not get illegal import warnings for importing your own classes.

~~4.~~ To get some of these constraints checked automatically download a new UNCCheckstyle file (UNCChecks), and copy it into the Eclipse plugins folder as before, and restart Eclipse.

Formatted: No bullets or numbering

Algorithm Hints

Going from finding a single kind of token to multiple types of tokens may require you to restructure your program. With a single type of token you could write a single loop that finds both the start and end of a token. Conceptually this is not the best thing as the same loop in (a single method) is doing two things – skipping over token and non-token characters. In the case of multiple tokens, the loop has to remember which token it is scanning and have different ways of skipping over token characters based on the kind of token. It will be easier to approach this problem by defining getTokenLength(String aString, int aStartIndex) for each kind of token. Thus, you can have a function getWordLength(), getNumberLength(), getQuotedStringLength(). Each of these functions has its own loop that skips over token-specific characters. The functions are

called from a looping method that finds the token starts and token types. In this approach, you do one thing in one method and one loop - finding the token start means skipping over non token characters (white space) and finding the token ends means skipping over token-specific characters. These are only hints – feel free to ignore them.

Extra Credit

1. Allow an arbitrary number of spaces before and after each token.
2. Give an error message if the closing double quote is missing.
3. Look also for and print (a) the plus token, consisting of the single character, '+' and (b) the minus token, consisting of the single character, '-'.' The type of these tokens is "sign". *Do not make this token a part of the number token.* Thus, if the argument is "move – 100 " you should print three different tokens and not two:

   word: move

   sign: -

   number: 100

4. Write your own method for determining if a character is a letter rather than using the Java Character.isLetter() method.

Debugging

It is easy to make mistakes writing a program of this complexity; therefore you will probably need a debugger for this and later assignments. To encourage you to do learn the debugger quickly, we require that you demonstrate the ability to set break points, step into and over a statement, and examine the stack, which are explained in the class material. Use these even if your program works correctly. You must submit ~~a~~ screen shots showing you know how to use these features on this programming assignment. These can be similar to the ones shown in the class material (see section on debugging in eclipse or Eclipse install and use). To print a Windows window, put the mouse in the window and press the Alt and PrintScreen buttons simultaneously. This puts the window in the cut/copy buffer. Now you can paste the window into a Word or PPT document. If you are using a different IDE, show debugging shots for that IDE.

Submission Instructions

- These are the same is in the previous assignment except your document should also contain debugging screen shots.
- If you finish part 1 and part 2, give screen shots only for part 2.

- Be sure to follow the conventions for the name and package of the main class.

Good luck!

Looking Ahead

Scan for two additional 1-character tokens: "{" and "}", which we will refer to as the start and end tokens.

For each kind of token you detect above (number/word/quoted string/plus (extra credit)/ minus(extra credit), start token, end token), create a class whose instances store a token of that kind. In addition, create two token classes for the start and end tokens. Let us refer to these classes by the kind of the tokens they represent.

The number class defines two properties: (1) an editable String property storing a legal string representation of a number, and (2) a readonly int property representing the int value of the string. Thus, if the editable property is the string "00200" the readonly property should be the **int** 200. You can assume that the editable property will always be assigned a legal value. The class should also define a constructor for assigning an initial value to the editable String property.

The word class also defines two properties: (1) an editable String property storing a legal string representation of a word, and (2) a readonly String property that is a lower case representation of the string. Thus, if the editable property is the string "MoVE," the readonly String property should be the String "move". Again you can assume that the editable property will always be assigned a legal value. This class also should define a constructor for assigning an initial value to the editable String property.

The other classes define a single editable String property representing a legal quote, plus, minus, start and end string, respectively. Of course, if you have not done the extra credit part to recognize a sign, then you need not define the plus and minus classes.

Thus, each of these classes defines an editable property defining a legal token string associated with the class; and the word and number classes define an additional readonly property storing an alternative representation of the token string.

Extend your scanner bean class to use these token classes. The setter method of the scanner bean breaks up its parameter into various token strings. For each of these token strings, it does the following. It (a) creates an instance of the corresponding token class, (b) assigns the token string to the editable String property of the instance (using the constructor), and (c) prints (using System.out.println()), on different lines, the (i) instance itself, and (ii) all properties of the instances. If you detect errors, then the method prints these also on the console. The tokens are not stored in this assignment.

To illustrate, if an input line is:

MoVe 050 {saY “hi!”}

this line should be assigned to the editable property of the scanner bean class, which will produce output of the form:

<Token ToString>
MoVe
move
<Token ToString>
050
50
<Token ToString>
{
<Token ToString>
saY
say
<Token ToString>
“hi!”
}
<Token ToString>
Here, <Token ToString> is a placeholder for some actual string println() produces when it prints an instance of a token class.

Do not submit the solution to this part – you will do so as part of the next submission.