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

Early Submission Date: Wed Sep 6, 2017 (+5%)

Completion Date: Fri Sep 8, 2017

First Late Date: Tue Sep 12, 2017 (-10%)

Second Late Date: Fri Sep 15, 2017 (-25%)

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. The first part gives you more practice with scanning. This is the complicated part The second one gives you practice with class decomposition in general and instantiatable Bean classes with properties in particular. It pretty much mimics what was done in class., but if you do not mimic the expected Bean conventions, we will not be able to grade your assignment properly.

In the previous assignment, we forced certain class and method names on you such as ScanningIterator and indexOf, so that our automatic grader can check if you have defined them. From this assignment onwards, you will have the flexibility to choose your own names of program elements, but to allow the grader and us to understand their purpose, you will have to use tags, which can be considered as structured comments. To use these tags, you will need to use oeall22.jar, which is included in the local checks jar.

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 Checks File	lectures.class_dual_roles.statics Package Git (Static) lectures.class_dual_roles.instances Package Git (Instances)

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 “Holy Grail” ”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: Holy Grail
quoted string: 2$ %!@

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 “.”.

As you 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 main class.

You can concatenate strings using the + operation as in:

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

Part 2: Class Decomposition and Scanner Bean Class

The goal of this part is to help you decompose your program into multiple classes. There are many good ways to do such decomposition. We see one such decomposition here.

Create a separate Bean class (a class that has one or more properties – see the reading material and praxis for the definition of a Bean and a property). For those of you who know constructors, it defines no constructor or a constructor with no arguments – thus we should be able to instantiate it with no constructor arguments. It defines a single instance stored editable String property named ScannedString for storing the input string and scanning it. Thus, the class has both instance (non-static) getter and setter methods for this property and an instance variable (whose name does not matter) that is read and written, respectively, by the getter and setter. You should make sure that the getter returns the value passed to the argument of the setter by reading the instance variable assitned by the 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 (it does not read the input, which is the task of main) and prints the tokens in this string in the manner described in part 1. It does so by calling the scanString() method, which should become a static method in the Bean class. The index() of method should become a static method in the Bean class (assuming no ScanningIterator extra credit) or remain as a static method in the ScanningIterator class (assuming the extra credit).

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 once to create a scanner Bean object. It does not directly call scanString() in the Bean class. Instead it simply calls the setter method with each 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.Tags;
```

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

```
@Tags({"ScannerBean"})
public class … { … }
```

These will allow CheckStyle and our grader program to ensure you are meeting the requirements without imposing certain names on the classes. As mentioned above, these imports require you to associate oeall22.jar or Comp401LocalChecks.jar with the project. (Right click project, select properties, Build path, add external jar).

Testing

You should test not only the input/output behavior but also each individual method you have changed or added in this assignment such as scanString() and the ScannerBean getter and setter methods. Make sure calling the ScannerBean setter on an instance of the class is the same as calling the static scanString() method on ScannerBean. Also ensure the getter returns the last value set. Testing of methods is optional. Testing of I/O behavior is required and will be evaluated through the screenshot(s).

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. To detect whether a character is a double quote, you need to represent a quote as a character value and compare the character with this value.

   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.
3. Follow the constraints in the previous assignment about Scanner instantiation.
4. You can have an arbitrary number of non-public and public methods in the Bean class. The specified property determines only some of the public methods in the class.

5. 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. This way you also get to copy the consent forms for local checks and check style.
6. If you create a package other than main start its name with "grail" to ensure you do not get illegal import warnings for importing your own lasses.
7. Each class should be preceded by the keyword public, which ensures that it is in a separate file.
8. Make sure your project is self-contained and does not reference other projects such as JavaTeaching, recitation and previous assignment submissions.

For Checkstyle checks, you will need to convert the associated checks file to a new configuration and associate it with the project. For local checks, you will need to download a new version of the jar file. If you created this project by copying assignment 1, all you will need to do after downloading is select project and execute the refresh command.

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 and assume that the end of the line serves also as the closing quote.
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
   ```

   You can assume that, like other tokens, the sign token has a space following it. You are free to assume no space or a variable number of spaces between it and the next token 100

It is up to you how you handle inputs such as "abc123 123ab abc"+". According to our rules, these are not valid strings, but you can (without getting credit for it) break this string into the tokens abc 123 123 ab abc "+" by assuming that a character that does not belong to the current token starts a new token.

Algorithm and Structuring Hints

Again look at this only if after you have thought about the problem.

Recognizing words is very similar to recognizing numbers – the indexOf operation can be used in a very similar way. In both cases you use spaces to delimit the token, and based on the first character of the token, you can decide which one to recognize.

The tricky part is quoted string. Again you need to use the first character of the next token to determine if it is a quoted string. However, the end of the token is decided by not a space but a quote, and you must remember to skip over it before you find the next token.

If you implemented the ScanningIterator, it still returns a string, and you can use the first character of the string to determine how to classify the string. Later your iterator will return token objects.

While we do not require this – it would be useful to modularize the scanString() method by creating separate methods for scanning different kinds of tokens. For instance, we can have one method for scanning a number and another for scanning a word. scanString() uses the first character of token to determine which of these methods is called to do the remainder of the scanning. Depending on how much extra credit you have done, this can result in a much shorter scanString() method. In the case of one student, an 80 line scanString() method became a 20 line one and 4-6 other specialized token scanning methods (he was doing the extra credit and looking ahead parts).

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 use the debugger solo – without TAS demonstration - 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 do not have to break your program, just show how to step through a program. You must submit 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) but now the code in the screenshots must be from this assignment. 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. The screenshots will not be scrutinized very carefully – all we will check is if you have executed the steps.

Submission Instructions

- These are the same is in the previous assignment, **except you should submit debugging screen shots (as separate individual unzipped files) in addition to the tests.**
- 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 so that its full name is: **main.Assignment2.**

Possible Mistakes

An important mistake to watch out for is confusion between variables and properties.

If we say, define a property called ScannedString, it is not sufficient or necessary to define a variable called ScannedString. You can call the variable anything. It is good to, of course, give it the same name as the property, but all variable names start with lowercase. So a good name is: scannedString. We always start a property name with upper case to distinguish it from a variable. And you need a variable only for stored properties. Some properties are not stored.

But you do need a getter and maybe even a setter based on whether the property is editable or not. If we ask you to define a property named P of type T, you need to define a public getter with the header:

```
public T getP();
```

If we ask you to make it an editable property, you need also a setter with the signature:

```
public void setP(String newVal);
```

If you do not follow these conventions, our grader cannot understand your code, and these conventions are made for code understanding.

Other possible mistakes:

1. Calling illegal functions (such as String indexOf and split) and using illegal imports.
2. Not calling the indexOf() function for finding tokens (related to the above).
3. Not moving scanString() from main to the scanner bean.
4. Copying scanString() and not deleting the on in the main class.
5. Not allowing spaces in quoted strings.
6. Not submitting debugging shots.
7. Scanner Bean class is called Bean. Almost all of our classes will be Bean classes. So this name is too generic.
8. No Tag for the Bean class - it should have the tag "ScannerBean".
9. The Scanner Bean class has static getter/setter methods and static variables -- these should be instance methods and variables.
10. The setter method in the Scanner Bean does not set an instance variable - it should set a variable that is returned by the getter.
11. No getter method corresponding to the setter - each property should have a getter, even though it is not used in this assignment.

12. The scanString() method in the Scanner Bean class changes an instance variable - it should not change any global variable.
13. The scanString() method in the Scanner Bean class does not produce any output - It should be the method printing out the tokens in a loop.
14. The getter or setter methods are named getInputString and setInputString rather than getScannedString and setScannedString.
15. The main class does not instantiate the Scanner Bean class -- you need to use the new operation on the Bean class.
16. Scanner instantiated more than once in the main program- just need one instantiation after the main called.
17. ScannerBean instantiated more than once in the main program - just need one instantiation, as setScannedString() resets the input string.
18. Naming the main class AssignmentTwo or some other class rather than Assignment2.
19. A get method does not return the last value set by the corresponding set (GetsReturnSets).

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.