Comp 401 - Assignment 3: Creating and Interacting with Bean Classes

Date Assigned: Wed Sep 5, 2012

Completion Date: Fri Sep 14, 2012

Early Submission Date: Wed Sep 12, 2012

So far, even though you have used an object-oriented language, you have done conventional rather than object-oriented programming. In this assignment, you will get practice with class instantiation, instance methods, instance variables, properties, and Bean conventions.

Each of the classes you will create will be simple, but you will create a number of classes. The assignment is divided into several parts to make it easy to incrementally create these classes.

This assignment builds on the previous assignment. In that assignment, each token string you extracted from the input string (number/word/quoted string) was stored in a String, char array, or int value (e.g. "10", "move"), which in turn was printed on the console. Now, you will store tokens in instances of token classes defined by you, which will use Bean conventions to define readonly and editable properties.

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

Your scanning will be done in instance rather than class (static) methods. The scanner class will have its own property related to scanning. Your main class will instantiate the scanner class and set properties to process input.

You will also use ObjectEditor to interact with instances of some of the classes.

Finally, the classes in your project should be put in at least three different packages. Each package should contain logically related classes. A package can consist of a single class.

The assignment may be changed in minor ways in response to student questions. So please look for changes in track changes mode before you submit. The changes will also be announced on Piazza.

The following material is relevant to this assignment.

110 and 401	Object-first Introduction to Programming (8/27, 8/29)	PowerPoint	PDF	Objects Chapter			lectures.objects Package
110 and 401 (look on you own)	Packages	PowerPoint	PDF	Objects Chapter			
110 and 401	State and Properties (9/5,	PowerPoint	PDF	State Chapter	Video		lectures.state.properties Package

Part 1: Token Classes

For each kind of token you detected in the previous assignment (number/word/quoted string/plus (extra credit)/ minus(extra credit), create a class whose instances store the token. 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. We will cover constructors on Monday 9/10.

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.

Part 2: Scanner Class

The scanner class defines a single editable String property. Thus, the property has both a getter and setter method. The setter method for the property 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.

Main Class

The main class defines a main method that like the main method of the previous assignments, reads a series of input lines ending with a "." To scan each of these lines, it creates and uses a single instance of the scanner class. Thus, it simply assigns each input line to the editable string property of the instance.

To illustrate, if an input line is:

MoVe 050 {saY "hi!"}

this line should be assigned to the editable property of the scanner 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.

The main class can, of course, contain multiple methods

ObjectEditor

You should also use ObjectEditor to display and interact with an instance of the word, number, and scanner class, and create a video or screen shots to show this interaction. Interacting with an instance means assigning some legal value to the editable property of the instance and showing the result of the assignmen. You do not have to use ObjectEditor to create the instance. Before your main method starts reading user input, it should create three instances and ask ObjectEditor to display (edit) these instances.

Constraints

1. You should only use all of the Java features allowed in the previous assignments and presented in class. In addition, you can use the string toLowerCase() method.
2. In general, *unless you are writing a scan function that accesses a global index variable*, in this and other assignments, you should not define a function with side effects.
3. You can have an arbitrary number of non-public methods in the classes. The properties determine the public methods in the classes.
4. You should not have any static methods and variables in the non-main classes.
5. Each of the non-main classes will follow Bean conventions. In each of these classes put a StructurePattern annotation indicating that the classes are indeed Beans. We will cover this annotation on Monday 9/10.

Extra Credit

Do not use the String toLowerCase(), Character toLowerCase() or any other library method that converts case. Instead, write your own- function that has the same behavior as String toLowerCase()– that is, takes a String argument with mixed case, and returns a lowercase representation of it. The method should convert a character into an int by using the (int) cast and use then cast an int to a character by using the (char) cast. Ideally, your code should not know the exact int to which a character corresponds and should only know the relative difference between the ints to which characters correspond.

Formatted: Normal

Formatted: Font: +Body (Calibri), 11 pt, Not Italic, Font color: Auto

Controlling the width of the text fields displayed by ObjectEditor

You may find that the default width of the text fields created by ObjectEditor is not sufficient to display the values of the properties you define in this assignment. In oeall8.jar and later versions, you can associate the getter of a property with a ComponentWidth annotation, which takes the desired width (in pixels) of the text-field used to display the property, as shown below:

```
@ComponentWidth (800)
```

```
public String getHeight() {
	return height;
}
```

The annotation above ensures that the “height” property is displayed in a text field whose width is 800 pixels.

You will need to import ComponentWidth as

import util.annotations.ComponentWidth

depending on the version you use. The best approach is to let Eclipse tell you what the import should be. All annotations are in the util.annotation package.

In all versions of ObjectEditor (e.g. oeall3.jar), you can set the component width of all properties using the following call:

ObjectEditor.setDefaultAttribute(AttributeNames.COMPONENT_WIDTH, 800);

In addition, you can set the component width of property P of class C to width by executing the method:

ObjectEditor.setPropertyAttribute(C.class, P, AttributeNames.COMPONENT_WIDTH, width);

For example:

ObjectEditor.setPropertyAttribute(ABMISpreadsheet.class, “height” , AttributeNames.COMPONENT_WIDTH, 800);

Submission Instructions

- These are the same is in the previous assignment except your document should also contain ObjectEditor screen shots or a pointer to a video .

Good luck!