Comp 401 - Assignment 3: Objects with and without properties

Date Assigned: Wed Sep 7, 2011
Completion Date: Fri Sep 16, 2011
Early Submission Date: Wed Sep 14, 2011

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. You will also learn to use ObjectEditor to interact with objects.

The assignment has two parts. The first part requires definition of an object with no properties, while the second one requires definition of an object with properties. Both parts provide alternative ways to do the scanning in the previous assignment.

The format of the scanned string is the same as before – the only difference is how the scanned string is specified.

**Part 1: Object with No Properties**
Define a class with a single public instance method (you can have as many non-public methods as you want) that takes a String parameter and returns a String value. It should scan the String for the tokens of Assignment 2 and return a String that is a concatenation of these tokens and their token descriptions. The description of a token is the same as in the previous assignment. Thus if the parameter of the method is the String:

```
move 30 say "you say hello" move 100 say "I say goodbye" "$%!@
```

the return value is:

```
command: move number: 30 command: say quoted string: you say hello command: move number: 100 command: say quoted string: I say goodbye quoted string: $%!@
```

You can put semicolons or parentheses to better distinguish the tokens – for instance you can put parentheses around string tokens.

If you detect and recover from illegal characters, then simply print them on the console. You should not include them in the return value.
Notice ObjectEditor does not require you to put double quotes around String values. As a result, it also does not require you to escape these values. ObjectEditor also does not display quotes around the strings it displays.

The class is “stateless” in that it has no properties. However, it can have instance variables storing data shared among the (public and non public) methods of the class.

**Part 2: Object with two Properties**
Define a class that defines a stateful version of the class of Part 1. Using Bean conventions, defines two String properties. The first property is an independent editable property that stores a String to be scanned, while the other one is a dependent readonly property that stores the concatenation of the tokens in the scanned string along with their descriptions. Thus, if you (call a setter method to) assign the following String to the first property:

```
move 30 say "you say hello" move 100 say "I say goodbye" "$%!@"
```

the value (returned by the getter method) of the second property becomes:

```
command: move number: 30 command: say quoted string: you say hello command: move number: 100 command: say quoted string: I say goodbye quoted string: $%!@
```

Again, ObjectEditor does not require double quotes or print double quotes. Again, if you detect and recover from illegal characters, then simply print them on the console.

**Main Class**
Define a single main class to instantiate and use the two classes above and display the instances using ObjectEditor. The main method of the class should take a single String argument and perform the following tasks:

1. Create an instance of the class you create for the first part of the assignment.
2. Use this instance to scan the main argument and print on the console the tokens and their descriptions. This means, you will call the (public) method you defined in the first part with the main argument as the parameter, and print on the console the value returned by this method. This step does not require the use of ObjectEditor.
3. Use ObjectEditor to display this instance.
4. Create an instance of the class you create for the second part of the assignment.
5. Use this instance to scan the main argument print on the console the tokens and their descriptions. This means you will assign the main argument to the independent editable property of the instance (by calling the appropriate setter method) and print on the console the value of the dependent readonly property (by calling the appropriate getter method).
6. Use ObjectEditor to display this instance.

Thus, if the main argument is:

"move 30 say "you say hello"
move 100 say "I say goodbye" $%!@
"

When you run the program, two ObjectEditor windows will be displayed and the output of the program will contain the tokens and token descriptions (and associated error messages) twice:

command: move number: 30 command: say quoted string: you say hello command: move number: 100 command: say quoted string: I say goodbye quoted string: $%!@

command: move number: 30 command: say quoted string: you say hello command: move number: 100 command: say quoted string: I say goodbye quoted string: $%!@

Create screenshots showing the main argument and associated console output for one test case.

In addition, interact with each ObjectEditor window to interactively scan one example string, and create screenshots to show this interaction. This means you should interactively (a) call the method defined in part 1, and (b) get and set the properties defined in part 2; and create screenshots to show these two interactions.

**Errors and + and – tokens**

If you recognize the + and – tokens, then include these tokens in the return value in the first part and the dependent property in the second part. Print all error messages on the console, as mentioned above.

**Constraints**

1. You should only use all of the Java features allowed in the previous assignments and presented in class.

2. **You should not define getters that directly or indirectly change state. This means that scanning should be done in a setter and not a getter.**

3. In general, unless you are writing a scan function that accesses a global index variable, you should not define a function with side effects.

4. You can have an arbitrary number of public methods in the two classes.

5. **You should not have any static methods and variables in the two classes – so convert static methods-vars from the previous assignment to instance methods-vars.**

**Extra Credit**

1. Finish the assignment by the early completion date.
2. In part 2, instead of printing the errors on the console, store them in a third dependent readonly property.

**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 oedall8.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:

```java
@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

```java
import util.annotations.ComponentWidth
```

depending on the version you use. The best approach is to let Eclipse tell you what the import should be.

In all versions of ObjectEditor (e.g. oedall3.jar), you can set the component width of all properties using the following call:

```java
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:

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

For example:

```java
ObjectEditor.setPropertyAttribute(ABMISpreadsheet.class, "height", AttributeNames.COMPONENT_WIDTH, 800);
```

**Submission Instructions**

Same as before.
Good luck!