Comp 401 – Assignment 7: Inheritance Galore!

Date Assigned: Thu Feb 26, 2009
(Early) Completion Date: Thu Mar (17) 19, 2009

In this assignment, you will extend previous assignments, getting a significant amount of experience with inheritance. You will extend your scanner class of assignment 4 to iterate token objects rather than strings. These objects will be defined by a type hierarchy. You will override the toString() method of Object to display string values represented by these tokens. Finally, you will refactor your code of assignment 5 and 6 to reduce code duplication.

**Token type hierarchy**

Your scanner of assignment 4 reduces all tokens to the single type: String. Implement a class and interface hierarchy for representing these tokens. The type hierarchy should allow a programmer to use the `instanceof` operation on an object to determine if it is a:

- token or some other object,
- string, number or word token,
- `join`, `leave`, `move`, `message`, `undo`, or `redo` word token. It should be possible to distinguish between these tokens using just `instanceof` – without invoking `toString()` or some other method on the token object. As you can see, these tokens correspond to user commands you will be supporting in later assignments.

Thus, given an object representing the word `move`, a programmer should be able to use the `instanceof` operation to determine that a) it is a token, b) it is a word token, and c) it is the `move` word token. Let us call an object that is an instance of a type in this type hierarchy as a **token object**.

**Iterating token objects and overriding toString()**

Create a new scanner that iterates token objects in addition to strings. This means that the scanner user can ask for the next token as a string or as a token object. This implies that a single pointer is maintained to mark the position of the next string/token object. In fact, in this assignment you will not be manipulating this pointer directly.

In addition, ensure that the `toString()` method can be called on a token object to get its string value. In other words, the `toString()` method can be used to convert the token object back to the corresponding string value. Thus, if your previous scanner iterated the String `move`, the new iterator should, in addition, iterate a token object whose `toString()` value is `move`. 
To implement the scanner, you may need to change several classes in the token type hierarchy by defining additional instance variables, constructors, and methods. Make sure you do not duplicate code in the classes in the hierarchy— that is, reuse implementations of the variables, constructors and methods when appropriate.

Make the new scanner class a subclass of the scanner of assignment 4. As the previous scanner class iterated strings, this class will implement a new interface (or a set of interfaces) that iterates both token objects and strings. The methods that produce the next string and token object must have different names (as they both take no arguments). You are free to re-factor the previous scanner to make it more amenable to sub-classing. For example, you can rename \texttt{next()} in the previous scanner to \texttt{nextString()} and can create additional internal methods. However, you should not change the behavior of the public methods implemented by the previous scanner (unless you are fixing bugs), though you can change their names, as illustrated above.

The method returns the next token object should give an error if the next token is a word but not one of the user commands mentioned above. The behavior of the method that returns the next string should remain unchanged.

\textbf{Adding Inheritance to Assignment 5 and 6}

Use inheritance in your code of assignment 5 and 6 to reduce code duplication. To do so while following ObjectEditor naming conventions, you may have to define empty classes and interfaces. This is not bad style.

\textbf{Abstract Token Classes}

By the time you get to this part, we should have covered abstract classes. Make appropriate classes in the token type hierarchy abstract.

\textbf{Extra Credit}

Create token types representing additional avatar operations you have implemented and want the user to execute from a command line such as \texttt{smile}.

Allow a user to enter multiple words (chosen by you) for the same command. For example, the words \texttt{mv}, \texttt{move}, \texttt{MOVE}, \texttt{Move} could all map to the token type representing the move command. To implement this part, you can use the String \texttt{toLowerCase()} method. It is not necessary to distinguish between the synonyms using the \texttt{instanceof} operation. However, the \texttt{toString()} value of synonyms should show what the user actually entered.

\textbf{Submission Instructions}

1. Submit a print out of your code at the start of class on the (early) submission date together with screen shots showing your code working in various cases, and a document identifying how you support various style and functionality features. As always, write appropriate test programs to demonstrate the features.
2. Upload the assignment directory in blackboard. In general, for all assignments, you should do so by midnight of the day the assignment is due. But do not change the code after you submit it in class.