Assignment 1

COSC346 - Object Orientated Programming

Adam Sherlaw

Throughout this project, many object orientated principles have been applied:

Inheritance: Inheritance was used extensively throughout the project. All rules in our grammar are subclasses of the Grammar Rule class. By inheriting the Grammar Rule class, we are able to extend it to implement a more specific rule. By using inheritance, we reduce the amount of code that is written and makes creating new rules easier as we only add new behaviours and capabilities on top of the inherited behaviours.

Design patterns: The singleton design pattern is used to ensure that only one instance of the pattern exists. A singleton pattern is used with the dictionary to ensure that only one instance of the dictionary is ever created. Whenever the dictionary is required, a pointer to the single instance of the dictionary is returned. Using a singleton allows us to globally access the dictionary, without having potential data anomalies.

Coupling and Cohesion: Coupling and cohesion are present in all classes. The degree of the coupling between classes and the degree of cohesion inside the classes, gives us a way to evaluate how well the classes meet the object orientated design principles. Each of the rule classes are coupled by a single string input and output, which means that the classes have a low amount of coupling. Each rule class handles a single 'rule operation' from the total grammar, meaning that the rule classes are highly cohesive. By having low coupling and high cohesion, means that our classes are following good design rules thus code reuse and readability are good.

Polymorphism: Polymorphism is where a class inherits a generic method from the parent class and overrides the method with a more specific implementation. We use polymorphism in the rule classes where we override the parse method of the GrammarRule with a more specific implementation if a specific implementation is required. If we do not override the method, then the parent's method is used.

Alternative Structure: An alternative structure to the code would be to use procedural programming as if the (simple) grammar were to not change, then a procedural structure would be preferable (for a one off implementation). If the grammar is to be added to, then an object orientated structure is preferable as rule objects are easily created with less refactoring of code as each object is responsible for its parsing and operations.

Next Time: If I were to do this kind of project again, I would work from the smallest 'case' first. By working from the bottom up, and implementing the simplest elements and stopping conditions first. This would make testing individual elements of the code easier by building on top of known good code.

OO Appropriate: Yes, object orientated is appropriate as adding onto or removing from the grammar that is already implemented is a straight forward as we create a new rule classes or remove old classes. Doing so procedurally would require much more significant code change and also the changing of programs structure.

Mark: I believe that my project follows the principles of good object orientated programming in a sensible and practical manner. In addition to following good principles of object orientated programming, my project successfully implements Grammar Five along with an optional extra - Subtraction. This optional extra was fairly trivial to implement, so worth around half a mark. Focus on elegant code and readable formatting give me confidence to propose the mark of 19 out of 20.

Post-Mortem:

Upon completion of the project there are a few things to comment about:

The reasoning for the placement of the Variable and VarStore classes is due to the suitableness of the potential containing files. The Expression, Token and Grammar Rule files all have one purpose, to contain rules. Placing the storage of variables into any of these files seemed unsuitable as the storage of a variable is not a rule. This is why I have created a separate file called Variable and define the variable store class and the variable type there.

The reasoning behind creating a Variable type is that a variable is not a rule, it is merely an object with a name and a value. As a variable is not a rule it has been placed along side the VarStore class in the Variable file.

I have also chosen to use a Dictionary over an array as a dictionary allows for the look up of items via key values. As the variable has a name, it seemed appropriate to use this as the key value to which the variable object is stored to. This left me with an efficient way to lookup and access stored variables.

I implemented an OperatorSet token that parses a given regular expression string. The reasoning behind the OperatorSet is that I found to parse for two literals (say a Plus and a Minus sign) required two instances of literal parse to achieve the same result as one instance of the operator set parsing for a set instead. The operator set token is also used for the parsing of variable names as the variable names are a combination of letters and numbers.