Design Pattern Writeup

Pattern 1: Model View Controller

This design pattern is implemented almost entirely in our program. We have model classes such as Diagram, Class, basically anything that models the things the user is interacting with on the UML diagram. For the GUI we have the windows and dialogs as views and the event handlers on buttons for the controller. For CLI, we have an explicit view and controller. The controller executes commands and returns text results to the view.

Pattern 2: Memento

The Memento design pattern was implemented via our time machine class. Per the definition, without violation of encapsulation, we captured and externalized an object's (our diagram) internal state. This allowed for us to restore the next and previous states of our diagram for use in both undo and redo.

Pattern 3: Singleton

The Singleton pattern is also implemented in the TimeMachine. C# has the concept of a static class, so we decided to implement it this way. This allows the one global interface without us having to worry about managing an instance of the object.

Pattern 4: Private Class Data

The Private class data structural pattern is used within a few of our classes. One example would be within our relationship.cs model file. In this file, we define an attribute List<string> _validTypes that is defined as private static readonly due to the nature of it only being grabbed once. Similar to this, we have various buttons throughout our main window that are defined in the same way, since we really only need to be accessing them once.

Pattern 5: Prototype

This pattern is used in multiple places. It is employed in the model classes as an implementation of the ICloneable interface. We primarily use this for cloning the diagram and publicly exposing copies of stored classes instead of the actual references.

Pattern 6: Template

This pattern is employed in the AttributeObject class. NameTypeObject is a specialization of this type, but still has the AttRename method from the abstract class.

Pattern 7: Interpreter

We introduced an interpreter design pattern via our CLI implementation. Per the definition of the interpreter design pattern, our CLI provides a way to evaluate grammar or expression. We implemented an expression interface which is told to interpret commands, this involves symbol processing, etc.