**Oregon Institute of Technology**

Command Lab Winter 14  
Due Tuesday March 1st

**Goals:**

Refactor an existing codebase to use the command pattern

**Overview:**

The command pattern comprises two parts: Command(s) and an Invoker. The invoker typically wraps additional functionality around the invocation of the command. For this lab, you will use the CommandLogging demo as your starter code. From our discussions in class, we know that the CommandLogging demo uses the decorator, singleton, and command patterns to implement a “SuperSafeSpreadSheet” that saves\records the SetValue command to a binary file and can “replay” them later to recover lost work.

In today’s lab, we are going to create a new “batching” invoker that executes the SimpleSpreadSheet.SetValue() method only when 5 SetValue commands have been received. The new BatchInvoker will wait until it has received a batch of 5 SetValue commands. Once it has received a full batch of 5 SetValue commands it will execute all five commands and then wait until another full batch has been received.

The best place to start is to add a public BatchInvoker.cs file to the Command project.   
  
In your BatchInvoker class instantiate a private List data member that contains ICommand types.   
  
Also, add a public void Execute(ICommand command) method. This method should add the command parameter to the list and, when five commands have been added, it should foreach through the list executing the commands. Lastly, after executing the commands, it should clear the list to make room for the next 5 commands. Add Thread.Sleep(500) after clearing commands from list.

1. Add a public SpreadSheetBatchCommandDecorator.cs file to the Decorator project.  
     
   This class will need to inherit from the SpreadSheetDecorator class. Add the requisite namespaces.  
     
   Create a private instance of the BatchInvoker—name it ”invoker.”  
     
   Add the proper concrete decorator c’tor.  
     
   Override the SetValue() method of the SpreadSheetDecorator. In the overridden SetValue() method—create an instance of a SetValue command. Next, call invoker.Execute(command). Where “command” is the instance of the SetValue command just created.
2. In Program.cs, add a new private static void BatchSpreadSheet() method.   
     
   Inside BatchSpreadSheet(), create an instance of a simplespreadsheet. Decorate the simplespreadsheet with an instance of the SpreadSheetBatchCommandDecorator. Call the UpdateSpreadSheet() and ValidateSpreadSheet() helper functions, passing in the decorated spreadsheet, to prove your code still works as designed.
3. Comment out all code in Main() and add a call to BatchSpreadSheet(). Run the app—it should not throw any exceptions.
4. Create a sequence diagram of the events that occur when the SpreadSheetBatchCommandDecorator’s SetValue() method is invoked by client code.  
     
   Show the creation events. Show the event sequence for the first 4 SetValue() calls. Lastly, show the sequence of events on the 5th SetValue() call.
5. Update the UML diagram (handed out in class) by replacing classes that are no longer used with the new classes needed to batch commands.