Lab 9

- Due Friday by 20:59
- Points 0
- Available after 4 Nov at 14:30

Starter code (https://northeastern.instructure.com/courses/192553/files/29026622/download) . Use as per directions below.

Objectives

The objectives of this lab are:

- Use the command design pattern to support an extended set of operations for a class.
- Implement the command design pattern.

2 Introduction

The command design pattern offers a way to encapsulate extended functionality for an object. Specifically, each command object represents a unique way to use the existing set of operations of an object A to implement a new operation without changing the object A. As the new functionality is encapsulated within an object (the command object), adding newer operations is more scalable. Command objects also offer a way to enforce common functionality on all operations (such as undoing an operation).

Consider an object that offers a fixed set of functionality. A macro represents a unique sequence of using this functionality to implement a "meta operation". Macros are supported in many programming languages as an efficient alternative to functions to create often-used meta functionality. Spreadsheet programs allow writing macros using existing functionality to create customized high-level operations. The advantage of a macro is that once written, it can be used as a "single function" just like any of the set functionality. In this way, macros are a scalable way to create a growing extended set of functionality.

3 Provided code

Load the code in a new project in IntelliJ. Generate a class diagram for it to better understand its design (In IntelliJ Ultimate Edition, right-click on the src/ folder of the project, select Diagrams->Show Diagram->Java Classes.

The following is a brief summary of the provided code:

1. `SpreadSheet` : interface that represents a spreadsheet.
2. `SparseSpreadSheet` : a particular implementation of `SpreadSheet` .
3. `SpreadSheetController` : the controller of an application that uses a spreadsheet.
4. `SpreadSheetProgram` : a class defining the `main` method of this application.

5. Some tests for the `SparseSpreadSheet` and `SpreadSheetController` for illustrative purposes.

4 What to do

For the purposes of this lab, we will assume that the `SparseSpreadSheet` has been tested and works correctly.

4.1 Part 1: Designing support for macros

One could write newer functions by repeatedly adding them to the `Spreadsheet` interface and the `SparseSpreadSheet` implementation. Alternatively one could create new interfaces that extend the `Spreadsheet` interface to add newer functions, and then implement them by reusing the `SparseSpreadSheet` class. Both of these approaches are not scalable/sustainable, because the number of newer functionality is endless.

Instead, we will make our spreadsheet support a macro. A macro will be given to the spreadsheet as a function object that operates on it. Thus, the spreadsheet needs to have only one additional functionality: the ability to accept macros as function objects.

1. Create a new interface named `SpreadSheetMacro` that represents a macro. This is our "command" interface.
2. Add a new method to this interface that takes in a `Spreadsheet` object and executes this macro on it.
3. Extend the `SpreadSheet` interface in a new interface called `SpreadSheetWithMacro`. Add a method `void executeMacro(SpreadSheetMacro macro)` that will accept an object of your macro interface and execute it on the spreadsheet.
4. Implement this new interface in a class named `SpreadSheetWithMacroImpl` that reuses the provided spreadsheet implementation in some way. Implement the new method you added above, so that the function object passed to it is executed on this spreadsheet.

The above steps create a new spreadsheet that, in addition to its original functionality, now accepts and runs any macro.

Confirm your design by showing the TA before moving on.

4.2 Part 2: Implement a macro

We will implement a specific function as a macro: assign a specific value to an entire range of cells.

1. Implement the interface that represents your macro in a new class called `BulkAssignMacro`. This class represents the "bulk assign macro".
2. Write a constructor that will take in the necessary arguments for this operation: the range of cells to set, and the value to set them to. The constructor should have the usual checks for invalid parameters.
3. Write a test for this macro when provided valid inputs, as well as at least one test that verifies its intended behavior for invalid ranges of cells.

4. Implement the required method in this class to set the range of specified cells in the spreadsheet passed to it, to the specified value.

Show your work to the TA before moving on.

4.3 Part 3: Enhancing the controller

In this part we will enhance the given controller so that it will also support the "bulk assign" operation as a script command. The syntax of this method will be bulk-assign-value from-row from-col-num to-row to-col-num value. The specified rows will still be in the letter format (i.e. 'A', 'AA', etc.).

1. Write a new controller called `MacroSpreadSheetController` that extends the given controller.
2. Write an equivalent constructor in the new controller that takes in the same parameters.
3. Write a test for this controller that verifies that the script with the above syntax works correctly.
4. Override the `processCommand` method. This method should check if the command given to it is "bulk-assign-value". If so, then it should get the additional arguments, and use the macro object to execute this operation (what would you have to do to make sure this works?). Otherwise, it should delegate to the inherited `processCommand` method.
5. Verify that the test you wrote above passes on your implementation.
6. Override the `printMenu` method so that it includes this new command.

Show your test passing to the TA before moving on.

4.4 Part 4: More macros!

Now that we have a framework to add macros, we can extend the functionality more!

Write macros that will support the following script commands, using the same sequence as above.

- average from-row-num from-col-num to-row-num to-col-num dest-row-num dest-col-num. This will compute the average of a range of cells and put it in the specified destination cell. For example average A 1 B 10 C 2 computes the average of the 20 values in A1:B10 and stores it in the cell C2.
- range-assign from-row-num from-col-num to-row-num to-col-num start-value increment. This will set a row or column of cells to a range of values starting at the given value and advancing by the given increment. For example range-assign A 1 A 10 1 1 will assign A1:A10 to values 1, 2, 3, ..., 10 respectively.

Show the TA your work after implementing each macro.

Fix the style and submit this version to the submission server.

5 Questions to ponder/discuss

Macros add extra functionality to the spreadsheet. But can we create macros that use other macros? In general, does the command design pattern support "meta commands"? How?

What are the limitations or drawbacks of your macro design (or in general, the command design pattern)? Try to identify specific problems that you may face now or in the future in this specific application, rather than high-level limitations.