CS4015/SWE4403 Project Requirement

In this project, you will write an object-oriented program in Java or other programming language, using 12 or more design patterns described in GoF book. For each additional pattern beyond the required 12 patterns, there will be 1% bonus toward your final total score.

You also have option to do a different program of your choice than the following one with the same pattern implementation requirement. Before you do so, please talk to Dr. Du about your project first.

Before doing the following project, you should read Chapter 2 “Case Study: Designing a Document Editor” of GoF book and understand how the patterns are used in the case study.

The program lets the user format an HTML file by inserting/deleting HTML tags into/from a text file through a GUI. Note that the user can only edit HTML tags but not change the text.

The following are the HTML editing operations that your program should support.

- The program can display the original text file (without HTML tags) on screen.
- The program can display the HTML source file with added tags to the original text file on screen.
- The program can save the HTML source file with the same name as the original text file name with extension ".html" in the same directory where the original text file is stored.
- The program can insert an opening paired tag or a singleton tag before a character in the text file, and insert a closing paired tag after a character in the text file. Note that a tag can be inserted before or after an existing tag.
- The program can remove any existing tag. If the tag is a paired tag, the both corresponding opening and closing tags should be removed together.
- Assume that in the HTML file one paired tags cannot overlap with another paired tags (but nesting is okay). The program can support checking violation of this rule.
- The program can support undo and redo user HTML editing actions.

For simplicity, the following is a subset of HTML tags that can be inserted into the text file. Assume no attribute is associated with tags.

- Paired tags

<HTML> <HEAD> <TITLE> <BODY> <H1> <H2> <H3> <B> <I> <U> <P>

- Single tags

The following suggests how to implement your program with some of GoF patterns.

1. Composite

   Implement the composite pattern in your program in a similar way to the case study example of the GoF book to store the text in a tree structure with Character as leaf and Row as composite.

2. Builder

   Use a builder to build the above composite tree structure.

3. Singleton

   The builder should be a singleton

4. Flyweight

   Make Character objects that are leaves of the tree as shared flyweights as described in the GoF book. Each shared flyweight will be created or given by a flyweight factory.

   Note that the intrinsic state of a character flyweight is its actual ASCII character, and the extrinsic state is the position (row and column) of the character in the document.

5. Decorator

   Make HTML tags as decorators of characters on the text. When a tag is inserted before or after a character, a decorator of the character (where the character is actually a character flyweight) is created and the parent node (row object) of the character will point to the decorator object instead. For multiple tags of a character, you can create decorator of decorator for each tag insertion.

   Note that a decorator of a character is NOT a shared flyweight, but the character is a character flyweight.

6. Bridge

   Use the following List interface as the implementation of Row and Column classes, or Composite abstract class where Row and Column are its subclasses, as you did for question 1 of assignment 2.

```
interface List {
```

```
public int count(); // return how many elements in the list
public Object getAt(int index); // return the element at the given index in the list
public Object first(); //return the first element of the list
public Object last(); //return the last element of the list
public boolean include(Object x); //return true if x is in the list
public void append(Object x); //append x at the end of the list
public void prepend(Object x); //insert x in front of the list
public void delete(Object x); //remove the first appearance of x in the list
public void deleteFirst(); //remove the first element of the list
public void deleteAll(); //remove all elements of the list
public void replace(int index, Object x); //replace the object at the index by x
}
```

7. Adapter

Write 2 class based or object based adapters to adapt one of the Java ArrayList and Vector to the List interface, as you did in the assignment.

Note that for 6 and 7, you can also use Jave List interface and its ArrayList or Vector implementation without using Adapter.

8. Abstract Factory

Use a factory object to create nodes in the composite structure. For the composite nodes, create either ArrayList implementation or Vector implementation.

9. Iterator

Implement the iterator structure for the composite pattern in a similar way to the case study example of the GoF book. Note that based on the structure, you need to write Iterator interface, PreorderIterator class, ListIterator class, and NullIterator class.

To display the original text and the text with HTML tags, and checking the rules for paired tags in the HTML, you need to traverse the tree using the iterators in the same way as the case study example of the GoF book.

10. Visitor

To display the original text and the text with HTML tags and checking the rules for paired tags, you need to visit each node of the tree during traversing the tree for the corresponding task. Implement such visits using Visitor pattern. You need to write Visitor interface and 3 concrete visitor classes: DisplayTextVisitor, DisplayHTMLVisitor, and CheckPairTagVisitor.

Note that you can display the text from the original text file, instead of traversing the tree, since the text will not be modified during user editing. However to support future extension to the program to also allow text editing (insert and delete characters) by modifying the tree structure, your program should display the text from the tree.

11. Command and Memento

    Use the command and memento patterns to implement undo/redo.

12. Façade

    Divide your program into two subsystems: user-interface subsystem and application subsystem.

    The user-interface subsystem is responsible to receive the user's input commands and ask the application subsystem to perform the action, and display the action results returned from the application subsystem.

    You should use Façade pattern to implement the application subsystem. Only through the façade object by calling the façade's methods, can the user-interface subsystem communicate to the application subsystem. The Façade pattern should encapsulate all classes within the application subsystem.

13. Observer

    Use the observer pattern to implement the relationship between the user-interface subsystem and application subsystem.

Write a separate document or use comments in your program to explain what and where the patterns are used. You have freedom to implement the program using different GoF patterns or applying patterns in different ways than the above suggested applications of patterns.