Team 4

Black Magic

UNIT 3 HTML EDITOR

Submitted: 4/18/13

Chad Koppes

Brad Bensch

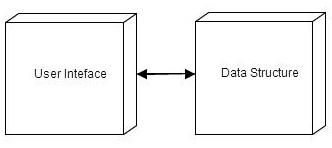
Jenny Zhen

William Spaw

Kevin Mulligan

**I. High Level Architecture**

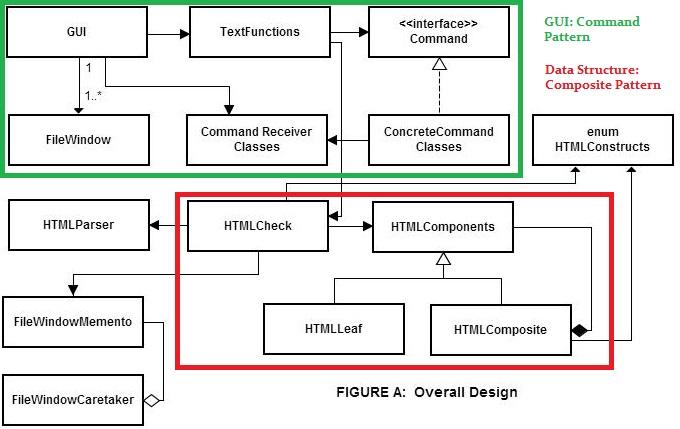
The high level design of this project is based on the interplay between two major features: the GUI representation of the text field for HTML editing and the backend data structure that provides the foundation for manipulating the text.



Although these two modules need to interact, our goal was to keep that interaction to a bare minimum so that changes could be easily implemented on one side without significant ramifications on the other. To this end, on this very high level, the user interface and data structure pass two pieces of data back and forth, a string that represents the text entered into a file, and a root node of a tree data structure that holds the string representation of a well-formed HTML file. This keeps these two modules separate and easy to handle, test and extend. The design pattern choices were made to maintain this level of separation and to keep the overall design as simple as possible while still meeting the necessary requirements.

**II. Rationale For Main Design Decisions**

Delving a level deeper in the design reveals how the major design patterns were chosen to support both the primary architecture as well as the individual requirements. The two major design patterns we used to support the broad architectural goal above were Command and Composite. The Command pattern provides a level of separation between the user interface and the functionality of the buttons and interactions happening there. The Composite pattern provides the foundation for the data structure that supports the HTML functionality required by the project. (See Figure A below) The highlighted portions in Figure A show the general class structure of the Command and Composite patterns and how they relate to one another through the class HTMLCheck.



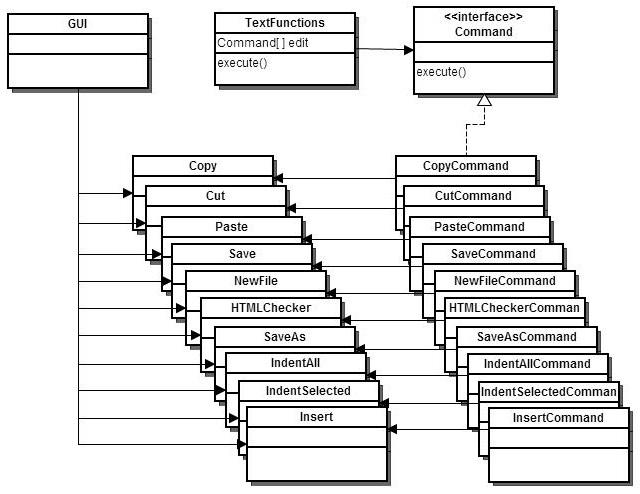
The other main classes shown here, including the Memento pattern in the lower left, are supporting the two principal design segments.

The Command Pattern (in green) is itself working to separate the functionality of the GUI classes from the actual implementation of the commands that it organizes. This allows us to easily add or change functionality to the GUI without interrupting functionality that already exists. FileWindow objects are generated for each individual file that is opened in the text editor. The Composite Pattern (in red) is using the inherently nested nature of HTML to create a composite. When a file is saved or the “HTML Check” button is activated HTMLCheck is sent a string representation of the file from which it generates a tree data structure using the HTMLComposite. If the HTML is well-formed the tree is held as a recursively built root node holding string data representing the text in the file and can then be further manipulated. In the next section will go a little deeper into the design to show how the patterns and classes are supporting the requirements.

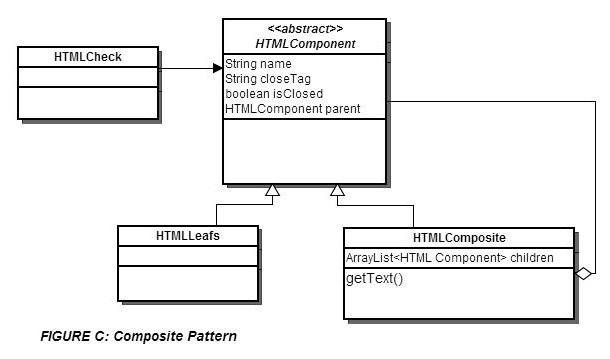
**III. Design and Requirements:**

Many of the release one requirements included typical features found on any text editor: saving files, opening files, multiple open files, cut and paste editing, etc. Most of these features were handled by the interaction of the GUI and Command Pattern. Figure B show this design pattern in more depth.

**FIGURE B: GUI and Command Pattern**



Our design represents a fairly typical Command pattern design. The GUI class is functioning here as the client and is, of course, the originator of the specific action listeners. The TextFunctions class is the invoker and asks the Command to carry out specific actions based on what the client has requested. The set of ConcreteCommands and Receiver classes show the specific actions based on the requirements. The receiver classes themselves hold the actual logic for carrying out the specific requests to indent, insert, etc.

Our use of the Composite pattern in release one turned out to be much more useful for release two than it did really for any of the requirements for release one. As mentioned above, the Composite is a natural choice for the nested nature of HTML files and so served well for the requirements requiring saving the state of an HTML file or regenerating HTML files in part or in whole. See Figure C below for a deeper look at the data structure side of the project. In order to better explain how this pattern works to meet the requirements, a couple variables and methods have been included. For each tag that is well-formed, a node is generated that holds the string representation of the opening and it’s paired closing tag (name, closeTag) and a parent node is attached to form the tree. The boolean isClosed is a flag that is used to indicate whether the children of this node should be made visible or not in the text editor. This is the foundation for the functionality behind linked mode and outline mode where text needs to be partially hidden.

This leaf nodes hold string text that appears between the tags including white space so that it can be reproduced accurately. Finally, the getText() method recursively returns all text (both composite nodes and leaf nodes) to reproduce the data held within it. HTMLCheck is the class that acts as the client in this pattern.

Finally, the undo feature for release two has been accomplished through the use of a memento Pattern using a simple stack. The state of the text document is maintained through memento objects based on the root node data structure and saved in FileWindowCaretaker. From there if the undo button is pushed the caretaker simply pops the memento holding the root node of the tree from that state save and can repopulate the text with the data contained in the tree. Also, the addition of the HTMLParser class is a third party helper for navigating some of the parsing issues with HTML documents. It is named Jericho-HTML and is an open source parser available at sourceforge (<http://jericho.htmlparser.net/docs/index.html>). Although it has a great deal of functionality, we only used it take a string representation of a file and parse whether or not the each word was a valid tag, close tag, or text segment within the document.

**IV. Design Principles:**

The design we have used we feel has achieved a high level of cohesion with good separation of concerns. The command pattern provides easy extensibility for the user interface and future functionality that might be added. Our coupling is a little higher than we would like, and could probably use some cleaning up, but we believe that it is a logical coupling based on established design principles and so could be understood by future developers. The two main segments of the program, the GUI implementation and the data structure, know very little of one another. The data structure simply takes a string representation of a text file to parse and build its tree, the GUI only uses a root node of a tree with text data contained within it to use for display manipulation for the user. This maintains a good separation of concerns and allows for for fairly easy debugging and extension on either side without disturbing the other.

**V. Changes to Release Two**

One of the things we strived to accomplish in release one was a design that would directly support what we suspected would be requirements for release two. The use of the composite pattern in release one provided a good foundation for release two and we did not find it necessary to make major design changes. The only major change was an addition of the Memento pattern and that fit in fairly nicely with our previous design.

The difficulties we encountered had more to do with tweaking the implementation of the design to accommodate the new features. For example, our text parsing for release one was too simplistic for release two and so had to be revised. Our composite pattern need two additional state changes to accommodate end tags and a boolean state for knowing if the node is collapsed or not. Our GUI needed some requisite changes for new buttons and pop-up windows, and therefore we added some classes to the command pattern. Overall we found that our use of patterns in release one really helped in release two.

**Class Responsibility Collaboration Cards**

|  |  |
| --- | --- |
| Command |  |
| Responsibilities | Collaborators |
| Provides an interface for concrete Command objects.  This class has the following methods:   * public void execute() - calls the command that performs the actual concrete command. | Concrete Commands, TextFunctions, GUI |

|  |  |
| --- | --- |
| Copy |  |
| Responsibilities | Collaborators |
| Copies highlighted text to Clipboard.  This class has the following methods:   * public Copy(GUI editor) - constructor that stores the editor * public void doCopy() - performs the actual copy command by copying the selected text to the clipboard.   This class has the following attributes:   * private GUI editor - the HTML editor to get text from. * private String text - the selected text to copy. | Clipboard, FileWindow, CopyCommand |

|  |  |
| --- | --- |
| CopyCommand |  |
| Responsibilities | Collaborators |
| Invokes copy, but does not perform the actual command.  This class has the following methods:   * public CopyCommand(Copy copy) - constructor that stores the instance of copy, that performs the actual command. * public void execute() - calls the command from the instance of Copy   This class has the following attributes:   * Copy copy - the instance of Copy that performs the actual command. | Copy |

|  |  |
| --- | --- |
| Cut |  |
| Responsibilities | Collaborators |
| Remove highlighted text from textarea, save it to clipboard.  This class has the following methods:   * public Cut(GUI editor) - constructor that stores the editor. * public void doCut() - performs actual cut command by removing the selected text and saving it to clipboard.   This class has the following attributes:   * private GUI editor - the HTML editor to get text from. * private String cut - the selected text to cut. | CutCommand |

|  |  |
| --- | --- |
| CutCommand |  |
| Responsibilities | Collaborators |
| Invoke Cut, but does not perform the actual command.  This class has the following methods:   * public CutCommand(Cut cut) - constructor that stores the instance of Cut, that performs the actual command. * public void execute() - calls the command from the instance of Cut.   This class has the following attributes:   * Cut cut - the instance of Cut that performs the actual command. | Cut |

|  |  |
| --- | --- |
| HTMLCheckerCommand |  |
| Responsibilities | Collaborators |
| Invokes HTMLCheck, but does not perform the actual command to check for well-formed (and valid) HTML..  This class has the following methods:   * public HTMLCheckerCommand(HTMLCheck html) - constructor that stores the instance of HTMLCheck, that performs the actual command. * public void execute() - calls the command from the instance of HTMLCheck.   This class has the following attributes:   * HTMLCheck html - the instance of HTMLCheck that performs the actual command. | HTMLCheck |

|  |  |
| --- | --- |
| Insert |  |
| Responsibilities | Collaborators |
| Add open and close HTMLConstruct tags to textarea.  This class has the following methods:   * public Insert(GUI editor) - constructor that stores the editor. * public void doInsert() - performs the insertion of opening and closing HTML tags to the textarea.   This class has the following attributes:   * private GUI editor - the HTML editor to add text to the selected file window. | GUI, InsertCommand |

|  |  |
| --- | --- |
| InsertCommand |  |
| Responsibilities | Collaborators |
| Invokes Insert, but does not perform the actual command.  This class has the following methods:   * public InsertCommand(Insert insert) - constructor that stores the instance of Insert, that performs the actual command. * public void execute() - calls the command from the instance of Insert.   This class has the following attributes:   * Insert insert - the instance of Insert that performs the actual command. | Insert |

|  |  |
| --- | --- |
| Indent |  |
| Responsibilities | Collaborators |
| Handles indenting of selected/all text and sets the length of the indent.  This class has the following methods:   * public Indent(GUI editor) - constructor that stores the editor. * public void indentSelected() - gets the selected text to prepend spaces to the editor based on the specified indentation length. * public void indentAll() - gets all of the text and prepends spaces to all of the lines in the editor based on the specified indentation length. * public void setSpacing(int spacing) - sets the length for a single indentation. * private void prependText() - adds spaces in front of a line of text to mimic an indentation.   This class has the following attributes:   * private GUI editor - the HTML editor to indent lines of. * private int spacing - the length of a single indentation (based on the length/number of spaces). | GUI, FileWindow, IndentAllCommand, IndentSelectedCommand |

|  |  |
| --- | --- |
| IndentAllCommand, IndentSelectedCommand |  |
| Responsibilities | Collaborators |
| Invokes indent, but does not perform the actual command.  This class has the following methods:   * public IndentAllCommand(Insert insert), public IndentSelectedCommand(Insert insert) - constructor that stores the instance of Indent, that performs the actual command. * public void execute() - calls the command from the instance of Indent.   This class has the following attributes:   * Indent indent - the instance of Indent that performs the actual command. | Indent, TextFunctions |

|  |  |
| --- | --- |
| NewFile |  |
| Responsibilities | Collaborators |
| Adds new FileWindow to GUI.  This class has the following methods:   * public NewFile(GUI editor) - constructor that stores the editor. * public void doMake() - opens a new blank file as a new tab within the HTML editor.   This class has the following attributes:  private GUI editor - the HTML editor to add a new tab representing a file window to. | GUI, NewFileCommand |

|  |  |
| --- | --- |
| NewFileCommand |  |
| Responsibilities | Collaborators |
| Invoke command to add a new file, but does not perform the actual command.  This class has the following methods:   * public NewFileCommand(NewFile newFile) - constructor that stores the instance of NewFile, that performs the actual command. * public void execute() - calls the command from the instance of NewFile.   This class has the following attributes:   * NewFile newFile - the instance of NewFile that performs the actual command. | NewFile, TextFunctions |

|  |  |
| --- | --- |
| Open |  |
| Responsibilities | Collaborators |
| Opens a file window to display a newly opened file.  This class has the following methods:   * public Open(GUI editor) - constructor that stores the editor. * public void doOpen() - opens the file chooser to select a file, and opens that file to display it in a new file window.   This class has the following attributes:   * private GUI editor - the HTML editor to display a newly opened file. | GUI, OpenCommand |

|  |  |
| --- | --- |
| OpenCommand |  |
| Responsibilities | Collaborators |
| Invokes the command to open a file, but does not perform the actual command.  This class has the following methods:   * public OpenCommand(Open open) - constructor that stores the instance of Open, that performs the actual command. * public void execute() - calls the command from the instance of Open.   This class has the following attributes:   * Open open - the instance of Open that performs the actual command. | Open, TextFunctions |

|  |  |
| --- | --- |
| Paste |  |
| Responsibilities | Collaborators |
| Takes text from the clipboard and inserts it to the current location in the textarea.  This class has the following methods:   * public Paste(GUI editor) - constructor that stores the editor. * public void doPaste() - gets the text that was saved to the clipboard and pastes it to the current location in the textarea.   This class has the following attributes:   * private GUI editor - the HTML editor to get text from. * private String text - the text to paste from the clipboard. | FileWindow, Clipboard, PasteCommand |

|  |  |
| --- | --- |
| PasteCommand |  |
| Responsibilities | Collaborators |
| Invokes Paste, but does not perform the actual command.  This class has the following methods:   * public PasteCommand(Paste paste) - constructor that stores the instance of Paste, that performs the actual command. * public void execute() - calls the command from the instance of Paste.   This class has the following attributes:   * Paste paste - the instance of Paste that performs the actual command. | Paste, TextFunctions |

|  |  |
| --- | --- |
| Save |  |
| Responsibilities | Collaborators |
| Save file; if name is null, use saveAs.  This class has the following methods:   * public Save(GUI editor) - constructor that stores the editor. * public void doSave() - checks to see if the filename is empty; if it is not, save the file. Otherwise, an exception is thrown.   This class has the following attributes:   * private GUI editor - the HTML editor to save the file of. * File file - the file to save. | GUI, FileWindow, SaveCommand |

|  |  |
| --- | --- |
| SaveAs |  |
| Responsibilities | Collaborators |
| Save a file, giving it a name.  This class has the following methods:   * public SaveAs(GUI editor) - constructor that stores the editor. * public void doSave() - tries to save the file.   This class has the following attributes:   * private GUI editor - the HTML editor to save the file of. * File file - the file to save. | GUI, FileWindow, SaveAsCommand |

|  |  |
| --- | --- |
| SaveCommand/SaveAsCommand |  |
| Responsibilities | Collaborators |
| Invokes Save or SaveAs command, but does not perform the actual command.  This class has the following methods:   * public SaveCommand(GUI editor, Save save, SaveAs saveAs) - constructor that stores the instance of Save and SaveAs, that performs the actual command. * public void execute() - calls the command from the instance of Save/SaveAs.   This class has the following attributes:   * Save/SaveAs - the instance of Paste that performs the actual command. * private GUI editor - the HTML editor to save the file of. | Save, SaveAs, TextFunctions |

|  |  |
| --- | --- |
| TextFunctions |  |
| Responsibilities | Collaborators |
| Invokes required action when a button/dropdown in the GUI is selected.  This class has the following methods:   * public TextFunctions(GUI editor) - constructor that stores the editor, and creates a list of concrete commands. * public void actionEvent(int spot) - calls the action required of the selected command.   This class has the following attributes:   * public static final int SAVE, COPY, CUT, PASTE, NEW, CHECK, SAVEAS, OPEN, INDENTAll, INDENTSELECTED, INSERT, UNDO - constant values that represent that index of each command in the list of commands. * private GUI editor - the HTML editor to get commands of. * Command[] edit - the list of possible commands. | Save, SaveAs, Insert, Open, NewFile, Paste, Copy, Cut |

|  |  |
| --- | --- |
| UndoCommand |  |
| Responsibilities | Collaborators |
| Invokes the undo command, but does not perform the actual command.  This class has the following methods:   * public UndoCommand(GUI editor) - constructor that stores HTML editor to undo a command of. * public void execute() - calls the command from the instance of GUI to undo a command.   This class has the following attributes:   * private GUI editor - the HTML editor to undo a command of. | GUI, FileWindow |

|  |  |
| --- | --- |
| HTMLCheck |  |
| Responsibilities | Collaborators |
| Check if the HTML in the selected file window is well-formed (has valid HTML tags, has proper closing tags for required tags). Builds the tree structure for the HTML tree.  This class has the following methods:   * public HTMLCheck(GUI editor) - constructor that stores the editor. * public HTMLComponent doCheck() - builds an arraylist of all HTML tags in a file, while checking if they are valid tags that exist in HTML and if they are opening/closing tags. * private void buildTree(HTMLComponent rootNode) - builds a tree of nodes that represent opening HTML tags. Closing tags are indicated by a isClosed boolean value as false. * private void validTree() - checks to see if the root node has no parent, since the check is at the top of the tree. * public static boolean checkValid(String tag) - checks if the given tag is a valid HTML tag that exists in the HTMLConstructs. If it is invalid, the node is not added to the tree. It is ignored. * private void addLeafNode(String name) - takes the string of text between nodes and adds it to the tree as a leaf. * private void addHTMLNode(String tag) - takes a valid opening HTML tag and adds it as a node on the tree. * private boolean checkCloseTag(String tag) - takes a valid closing HTML tag and tries to match it to the current, local nodes in the tree. | GUI, FileWindow, HTMLCheckerCommand |

|  |  |
| --- | --- |
| HTMLComponent |  |
| Responsibilities | Collaborators |
| Abstract representation of HTML tree nodes.  This class has the following methods:   * public HTMLComponent(String tag, String closeTag, HTMLComponent parent) - constructor to create a node. * public String getName() - gets the opening tag. * public void setCloseTag(String closeTag) - sets the value of the closing tag. * public abstract boolean isClosed() - checks to see if the tag was properly closed. * public boolean checkValidTag() - checks to see if the given tag is a valid HTML tag. * public abstract String getText() - gets a textual representation of the node. * public abstract boolean add(HTMLComponent child) - adds a child to the parent node in the tree.   This class has the following attributes:   * private String name - the opening tag. * private String closeTag - the closing tag. * public boolean isClosed - indicates if node was closed. * public HTMLComponent parent - parent of this node. * public ArrayList<HTMLComponent> children = new ArrayList<HTMLComponent>() - list of children associated with this node; null by default. | HTMLLeaf, HTMLComponent, HTMLConstructs, HTMLCheck |

|  |  |
| --- | --- |
| HTMLComposite |  |
| Responsibilities | Collaborators |
| A node in the HTML tree with proper opening and closing tags.  This class has the following methods:   * public HTMLComposite(String tag, String closeTag, HTMLComponent parent) - constructor. * public String getName() - gets the opening tag. * public void setCloseTag(String closeTag) - sets the value of the closing tag. * public abstract boolean isClosed() - checks to see if the tag was properly closed. * public boolean checkValidTag() - checks to see if the given tag is a valid HTML tag. * public abstract String getText() - gets a textual representation of the node. * public boolean add(HTMLComponent child) - attempts to add a child to this parent node. * public boolean addChild(HTMLComponent child) - adds a child to the list of children. | HTMLComponent |

|  |  |
| --- | --- |
| HTMLLeaf |  |
| Responsibilities | Collaborators |
| HTML Node with no closing tag.  This class has the following methods:   * public HTMLLeaf(String tag, String closeTag, HTMLComponent parent) - constructor. * public String getName() - gets the opening tag. * public void setCloseTag(String closeTag) - sets the value of the closing tag. * public abstract boolean isClosed() - checks to see if the tag was properly closed; true by default. * public boolean checkValidTag() - checks to see if the given tag is a valid HTML tag. * public abstract String getText() - gets a textual representation of the node. * public boolean add(HTMLComponent child) - attempts to add a child to this parent node; false by default. | HTMLComponent |

|  |  |
| --- | --- |
| GUI |  |
| Responsibilities | Collaborators |
| Client; user interface for the HTML editor.  This class has the following methods:   * public static void main(String[] args) - launches the HTML editor. * public GUI() - constructor to create an empty file window with no files open. * public GUI(File startFile) - constructor to create a file window with the startFile opened. * public void open(File file) - given a file, open and add to the view. * public String getExtension(File f) - returns the extension of a given file. * public Clipboard getClipboard() - gets the clipboard. * public String getText() - gets the HTML text. * public FileWindow getActiveFileWindow() - gets the active editor window. * public JFileChooser getFileChooser() - gets the file chooser for saving/opening. * public void open(), cut(), copy(), paste(), newFile(), save(), saveAs(), undo(), check(), IndentAll(), indentSelected() - calls the specific command in invoker. * public void toggleWordWrap() - toggles the active window’s word wrap status. * public void close() - closes the program; first closing all tabs, checking for well formed HTML, and saving if selected. * public void setMessage(String s) - method for setting dialog messages used as return for HTML check. * public void newTab() - opens a new, empty tab. * public void closeSelectedTab() - closes the currently selected tabs.   This class has the following attributes:   * String message - dialog message. * private JFileChooser fileChooser - browser for files on computer. * private Clipboard clipboard - stores copied/cut text. * private TextFunctions invoker - for invoking commands. * private ArrayList<FileWindow> windows - list of windows for displaying open files in the editor. * private JTabbedPane tabbedPane - provides tabs for multiple file windows. * private String text - stores open/close tags. | TextFunctions, FileWindow, HTMLConstructs |

|  |  |
| --- | --- |
| FileWindow |  |
| Responsibilities | Collaborators |
| Textarea that is opened for each newFile.  This class has the following methods:   * public FileWindow(GUI g) - default constructor; no file.. * public FileWindow(GUI g, File file) - constructor with a given file name. * public String getSelectedText() - returns the currently selected text in the text area. * public int getSelectedStart() - returns the starting position of the currently selected text. * public int getSelectedEnd() - returns the end position of the currently selected text. * public void insertText(String text) - inserts the given text into the text area at cursor position. * public File getFile() - returns this window’s file. * public String getAllText() - returns all the text in the editor window. * public JTextArea getTextArea() - getter for this window's text area. * public void setTextArea(String text) - setter for the window's text area. * public boolean isSaved() - check if this file has been modified or not. * public void save() - mark that this file has been saved. * public void setRootNode(HTMLComponent node) - set the fileWindow's root node to the given node. * private void setState(FileWindowMemento memento) - set this fileWindow's state given a memento. * public void undo() - return this fileWindow to the previous state. * public void saveState() - save the current state as a memento and give it to the caretaker.   This class has the following attributes:   * private boolean saved - dirty file marker. * private GUI parent - the user interface. * private File currentFile - active, open file. * private JTextArea textArea - contains the text of the file. * private HTMLComponent root - root of the HTML tree. * private FileWindowCaretaker caretaker - holds onto mementos, decides when mementos are created/used. | GUI |

|  |  |
| --- | --- |
| Clipboard |  |
| Responsibilities | Collaborators |
| Stores copied/cut text.  This class has the following methods:   * public Clipboard() - constructor. * public String getText() - get the previously copied/cut text. * public void setText(String text) - saves the given text to the clipboard. | GUI |

|  |  |
| --- | --- |
| HTMLConstructs |  |
| Responsibilities | Collaborators |
| List of valid HTML tags.  This class has the following methods:   * HTMLConstructs(String open, String close) - default constructor for tags with an open and close. * HTMLConstructs(String open, String arg, String close) - constructor for tags with an argument, such as, image. * public String getOpenTag() - getter for enum's opening tag. * public String getCloseTag() - getter for enum's closing tag. * public String getArgument() - getter for enum’s argument.   This class has the following attributes:   * private final String openTag - opening HTML tag. * private final String closeTag - closing HTML tag. * private final String arg - optional argument. | HTMLComponent, HTMLComposite, HTMLLeaf, HTMLCheck |

|  |  |
| --- | --- |
| LinkView |  |
| Responsibilities | Collaborators |
| List of all urls in the textarea.  This class has the following methods:   * public LinkView(FileWindow w) - constructor. * public void refresh() - updates the linked view. * public void close() - closes the linked view.   This class has the following attributes:   * private FileWindow window - displays the linked view. | FileWindow |

|  |  |
| --- | --- |
| FileWindowCaretaker |  |
| Responsibilities | Collaborators |
| Manages the addition and retrieval of mementos for the file window.  This class has the following methods:   * public FileWindowCaretaker() - constructor. * public void addMemento(FileWindowMemento newState) - adds the given memento to a list to keep for file window. * public FileWindowMemento retrieveMemento() - returns a memento from the list to give to file window.   This class has the following attributes:   * ArrayList<FileWindowMemento> savedStates - list of mementos saved for the file window. | FileWindow, FileWindowCaretaker |

|  |  |
| --- | --- |
| FileWindowMemento |  |
| Responsibilities | Collaborators |
| Saves the current state of the file window.  This class has the following methods:   * public FileWindowMemento(HTMLComponent rootNode, boolean savedState) - constructor. * public HTMLComponent getRootNode() - gets the root of the HTML tree from that state. * public boolean getSavedState() - gets the saved state.   This class has the following attributes:   * private HTMLComponent rootNode - root of the HTML tree for this memento. * private boolean savedState - saved state for this memento. | HTMLComponent, FileWindow, FileWindowCaretaker |

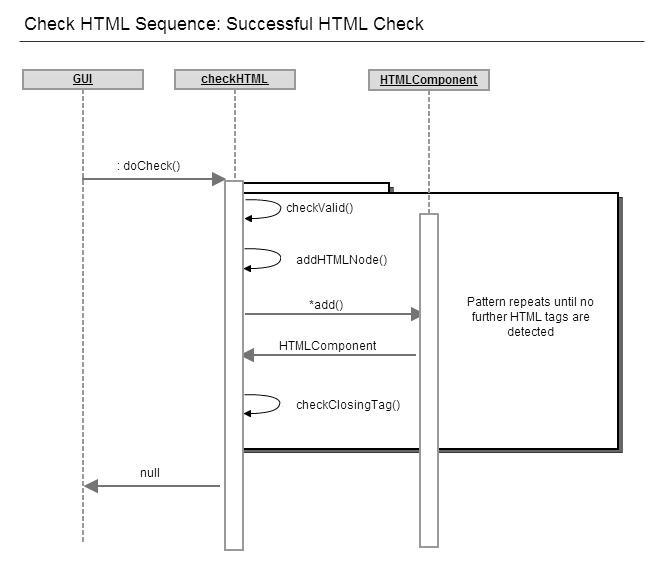
**Pattern Usage Templates**

|  |  |  |
| --- | --- | --- |
| **Name:Composite** |  | **GoF pattern:Composite** |
| **Participants** |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |
| HTMLComposite | Composite | Acts as a common node in the data structure holding any number of leafs or of itself as children. |
| HTMLComponent | Component | The abstract class from which HTMLComposite and Leaf are extended. It holds the common methods for both. |
| HTMLLeaf | Leaf | Holds the text that exists between HTML nodes. |
| HTMLCheck | Client | Generates the tree when given a string representation of an HTML file using HTMLComponent. |
| **Deviations from the standard pattern:** | None. |  |
| **Requirements being covered:** | Supports the generation and manipulation of the HTML Text in, for example, linked mode. If a class needs the HTML text to be manipulated, it receives a root node based on the Composite Pattern. |  |

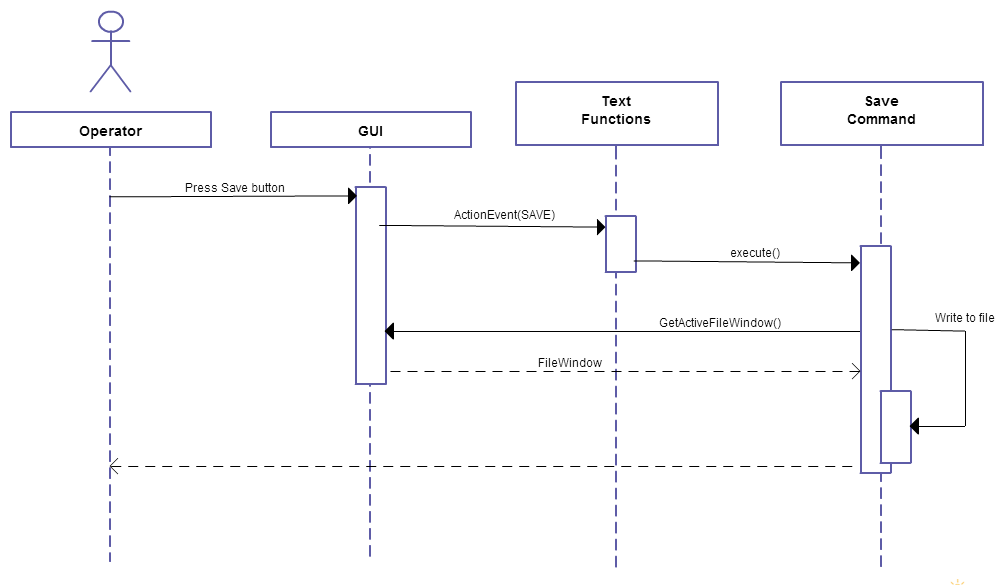
|  |  |  |
| --- | --- | --- |
| **Name: Command** |  | **GoF pattern: Command** |
| **Participants** |  |  |
| Class | Role in pattern | Participant's contribution in the context of the application |
| GUI | Client | Holds the actionlisteners that indicate a button has been pushed and an action requested. |
| TextFunctions | Invoker | Holds the list of commands and calls them according to the requests from the client based on the Command interface. |
| Command | Command | The interface that holds the execute() method and is extended by ConcreteCommand. |
| CopyCommand, CutCommand, InsertCommand.... et. al. | ConreteCommand | Gives the execute() command to the specific receiver who will execute the request. |
| Copy, Cut, Insert, ...et. al. | Receivers | Holds the actual logic that is run to perform the action requested. |
| **Deviations from the standard pattern:** | None |  |
| **Requirements being covered:** | All of the GUI functionality - save, cut, paste, indent, etc. |  |

|  |  |  |
| --- | --- | --- |
| **Name: Memento** |  | **GoF pattern: Memento** |
| **Participants** |  |  |
| **Class** | **Role in pattern** | **Participant's contribution in the context of the application** |
| FileWindow | Originator | Generates a memento, saving the current HTML tree and document save status |
| FileWindowmemento | memento | Stores the root node object |
| FileWindowCaretaker | Caretaker | Holds the root node in an arraylist of HTMLComponent nodes. |
| **Deviations from the standard pattern:** | None |  |
| **Requirements being covered:** | The undo feature. |  |

**Sequence Diagrams**



2. Save method



**State of Implementation**

There are a fair number of issues with the implementation. We struggled to figure out how to implement a collapsable interface for the user in the text window. We believe though that if we could simply generate some indication of what is being clicked in the window and assign it to the requisite node in the data structure, that we would be able to manipulate the viewable data correctly. In other words, our data structure, we believe could support this implementation.