SE 362 - Unit 3 Project

HTML Editor in Java

Release 2

Team # 2 - IntelliHTML

Shannon Trudeau
Keegan Parrotte
Grant Kurtz
Calvin D Rozario
Kocsen Chung

Submission Date: 18 APRIL 2013

Table of Contents

Overview of Project

The HTML Editor is a Java-based GUI application that is capable of performing various functionalities regarding manipulation of HTML text and files. The main aspect of Release One was to develop an editor for HTML files. The project consists of a team of 5 students that focus mainly on applying appropriate patterns and processes based on given requirements. The functionalities and features are exclusively designed for ease-of-use, such as point-and-click, auto-indentation, multiple files, and many others. All basic features for editing files include: save, save-as, shortcut key, copy and paste, validate, indent, insert-tag and quit.

In Release Two, all implementations and requirements from Release One remain unchanged except there are more requirements and implementations to enhance the existing system. In addition, Release Two begins with new design patterns consisting of features and functionalities, such as undo, redo, link view, HTML preview, tags expanding and collapsing, and refactoring of many areas.

Design Discussion

The design that we established with release one really carried our implementation and design for release two. We were able to work with almost all of the constructs that existed in our design and simply add more to them in order to create a more robust program that could handle the new requirements. The command pattern was a great example of how versatile our design was because we were able to very easily add new commands with ease. One aspect that we changed was adding a mediator to the command pattern in order to decouple commands from one another, which proved to be not quite as useful as we had hoped, but it still captured the intent quite well. The mediator pattern also made our design much stronger and able to better support more commands if we needed to.

Our design still really has two major subsystems: the file management and the command and GUI portion. The added portion of release two was that we were able to implement the strategy pattern for the links; the two strategies being the order in which the links were displayed to the user, either alphabetically or by appearance. We were also able to easily implement a memento pattern in order to save state for the undo actions that were listed in the requirements for release two. Some of the requirements were simple enough that they did not require the implementation of any patterns in order to be satisfied because our initial design was able to easily support them. One example of this was the rendering capability for an HTML file, which was easily handled by the JTextArea.

The same basic idea of our design was carried through to release two, meaning that our original patterns were kept the same and simply built off of. Our GUI was still observed by the HTMLFile for changes to the text and the commands were still executed through the user and encapsulated into objects that were passed to a new mediator that could then call other commands as necessary. We tried to maintain cohesion by only allowing commands to have knowledge of what they absolutely needed and thus had an instance of a CommandDistributor that allowed the commands to access the proper files if necessary through the FileManager. We also still had our driver App class which instantiated the GUI. In the end, our design was stable enough to satisfy most of the requirements of release two quite easily. The only trouble that occurred was in implementing code folding, which was difficult to implement because of our initial design. We were using an XML parser in order to validate the HTML, and in order to have the ability to fold it, we would have had to be keeping track of the tags tree in some sort of composite pattern, which was the other way to implement the validation requirement. Since we didn't do validation that way, we didn't already have a way to integrate code folding into the design without completely redoing what we had designed for release one. So we ended up referring to a third party solution of an XMLEditorKit that allowed us to present the code in a foldable manner, but without the ability to edit it while in this state. Yet in the end we were still able to achieve something rather than nothing out of this requirement. This was still the only real difficulty our design presented when faced with the new requirements of release two, and as a whole the patterns that we implemented and the top level structure proved to be incredibly versatile. Difficulties like this are to be expected when new requirements arise and our design was substantial enough to handle almost all of them without too many additions.

Design Images/UML Class Diagrams

For Specific Class information, refer to the next-next section called "Class Responsibilities"

Release 1 UML Diagram

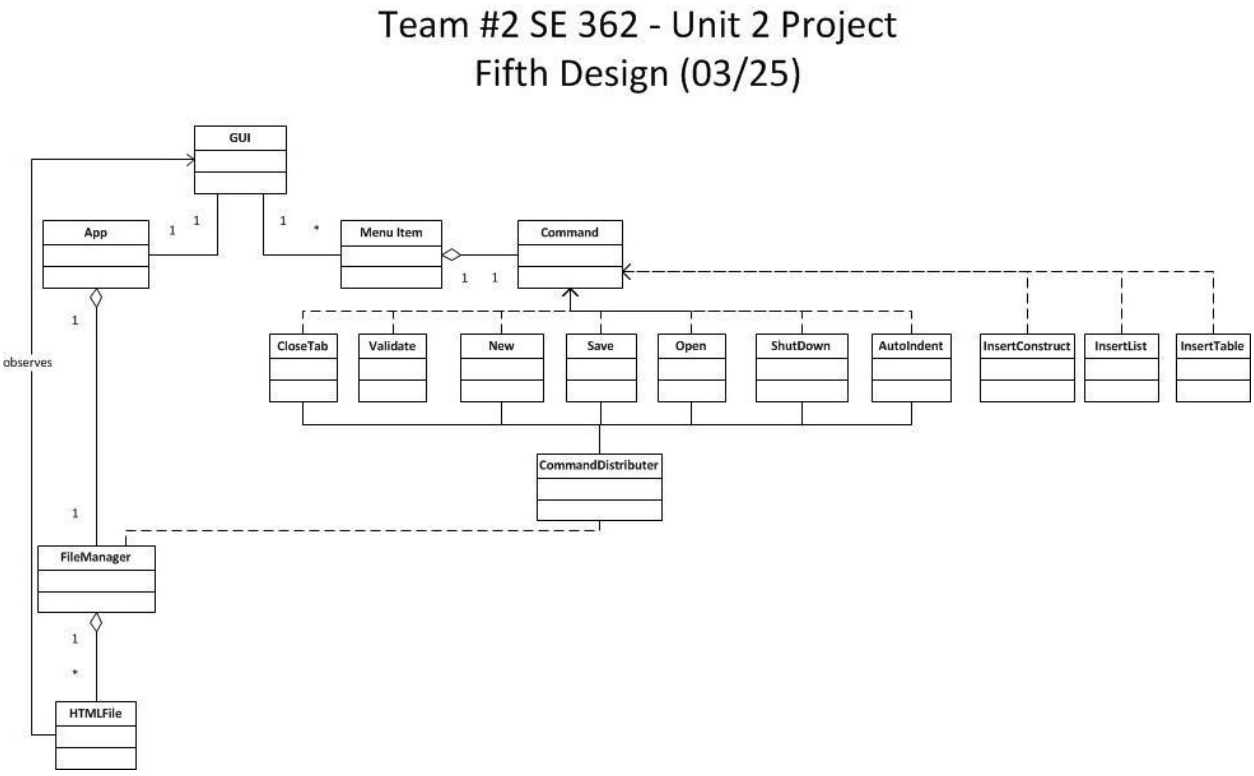


Figure 1 - Release 1 UML Class Diagram

This is our final design (number 5) for Release 1 of this project. Our main focus here is the command pattern. Almost every action is created as a common object and is then carried out appropriately by either the command distributor or the command itself. We also have a back end HTML File in charge of saving the document contents.

Release 2 UML Diagram

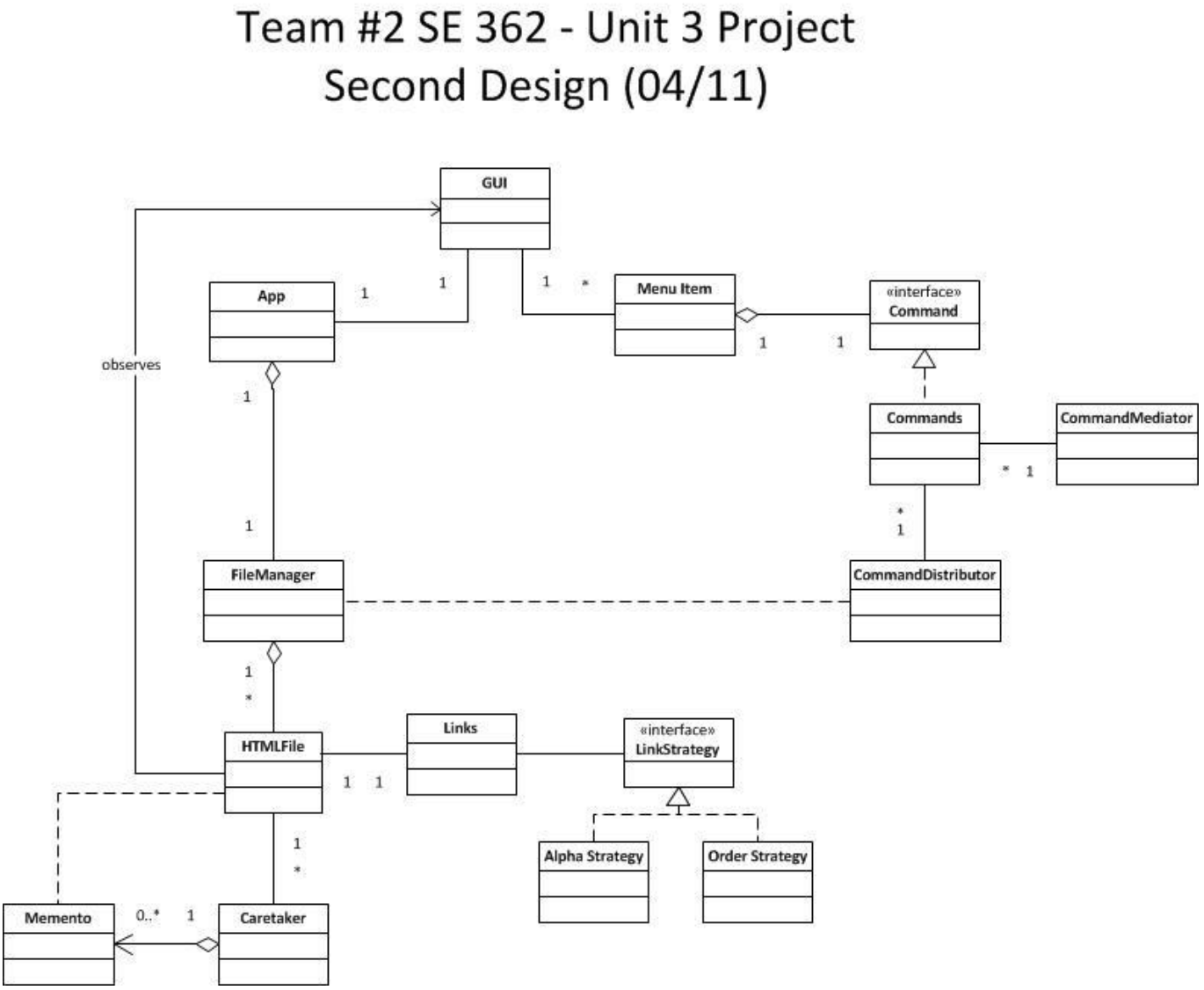


Figure 2 - Release 2 UML Class Diagram

This is the final document we have for Release 2. We decided to represent all the commands as one “Commands” class instead of many small classes for readability. Classes included in “Commands” that are not present in the Release 1 UML Diagram include RefreshLinks, OutlineMode, Undo, and others. In this design we gave HTML File more responsibility, by giving it a list of its own links and giving it the ability to create Mementos of its current state that can be referred back to in the event of an undo operation. We also added a mediator to our command pattern. The CommandMediator is notified when a command is completed, and from there will notify other appropriate commands to complete their objectives. For more specific details about how this works, see our Sequence Diagrams.

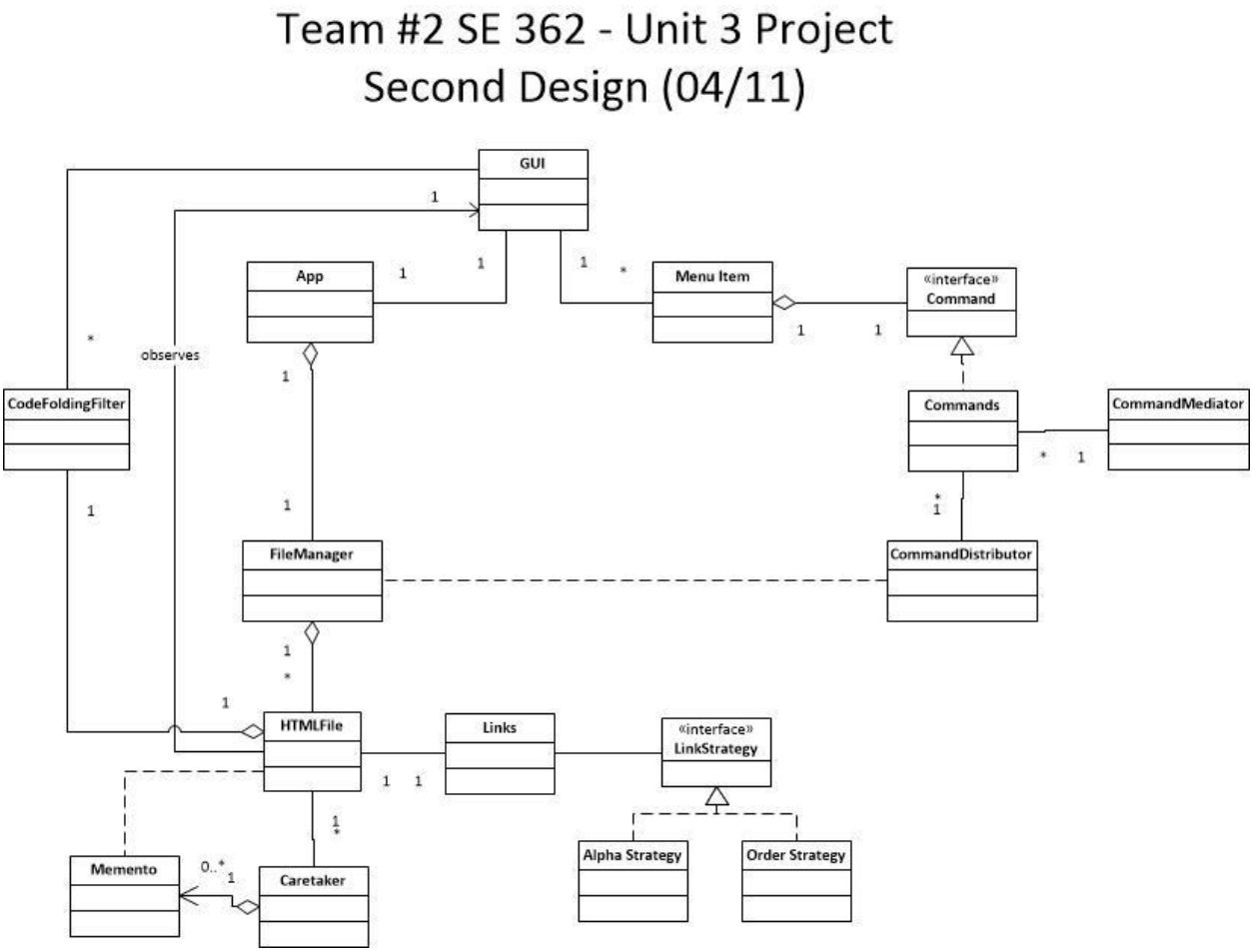
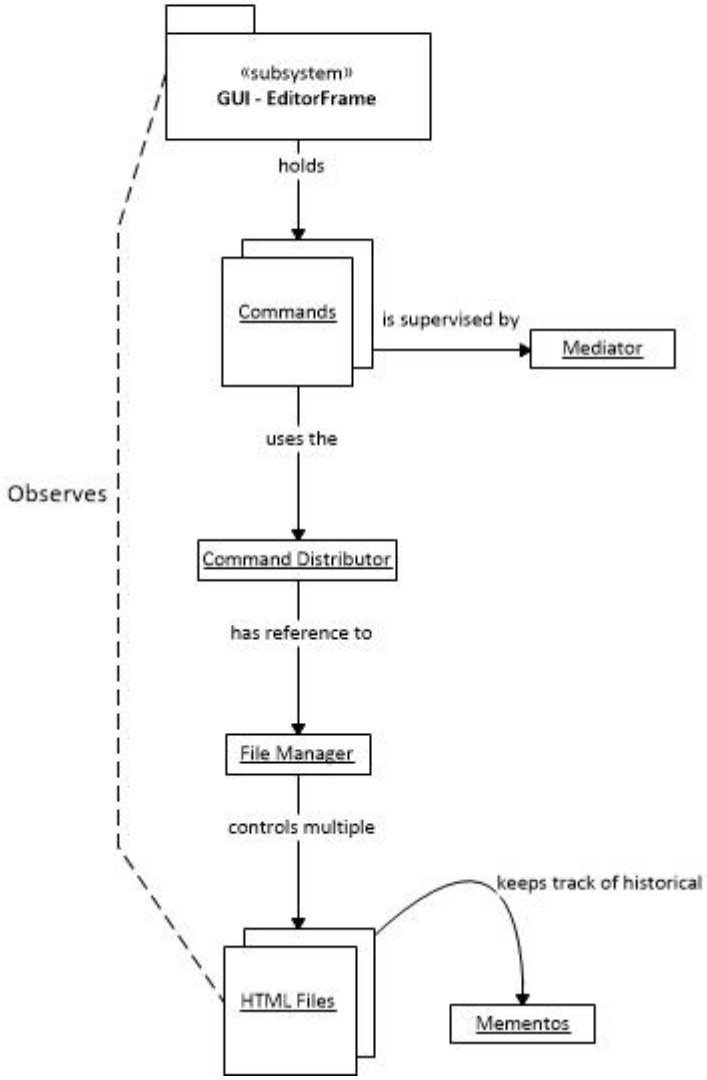


Figure 3 - Release 2 with Code Folding UML Class Diagram

We decided last minute to implement code folding into our program, and we were successful. We were able to display the code-folding, but the user cannot add/edit text whilst in outline mode. In other words, this feature is more like an add on rather than a design enhancement on our part.

Team #2 SE 362 – Unit 3 HTML Editor
Top Level Design Diagram R2 (04/14)



As seen above, the diagram is similar to that of R1 but slightly different. This time around, there is a mediator and mementos. The general structure is like a chain of commands; the editor receives user input and then the commands handle what has to be done. For added and organized functionality, the mediator makes sure he executes what has to be executed. The backend then has a wall of protection with the command distributor. As the chain continues, the file manager can perform work on the files, each of which has a memento to store the previous states. Finally, the HTML File weakly observes the changes done live on the editor to set it's save state and to periodically save states (mementos) for the undo.

Class Responsibilities In Release One:

Class: File Manager	
Responsibilities:	Manages all the separate HTML Files that are open, and performs operations on them within the context of the whole system. It is also responsible for handling the undo/redo state of the files with the new release.
Uses: HTML File	**Used by:** CommandDistributor, App

Class: HTML File	
Responsibilities:	This class is the backend representation of the file. It has responsibilities like check if it is in ‘save state’ and also keeps track of its previous states for undoing with the use of the Memento pattern.
Uses: Caretaker, Memento (indirectly)	**Used by:** File Manager

Class: Menu Item	
Responsibilities:	The menu item is the physical button that is within the EditorFrame. This is the invoker for the command. When a menu item is clicked, command(s) would get called
Uses: CloseTab, Validate, New, Save, Open, ShutDown, AutoIndent, InsertConstruct, InsertList, InsertTable, InsertImage etc. (*Command.java) All the commands	**Used by:**

Class: GUI/Editor Frame	
Responsibilities:	Constructs the window with all necessary components to present the editor to the user. Handles delegating execution of menu items. With the new release, the EditorFrame holds reference to a command mediator to organize the command executions.
Uses: CommandDistributor, CommandMediator	**Used by:** App, User

Class: TabFrame	
Responsibilities:	This is the JPanel component that each tab frame shows. It holds the JTextArea for editing the HTML and a JList for the Links view. This class allows all visible aspects to be controlled by this one class. In other words, its the component that each tab shows in the EditorFrame.
Uses: The text pane, the JList and the LinksListModel	**Used By:** EditorFrame

Class: CloseTabComponent	
Responsibilities:	This is the extension of a JPanel and allows the editor to have a button on the tab itself to close it. It's just a component at each tab and has been implemented to change the name of the tab whenever there is a new file name.
Uses:	**Used By:** EditorFrame

Class: Command	
Responsibilities:	Interface for each of the concrete command objects.
Uses: Menu Item	**Used by:** CloseTab, Validate, New, Save, Open, ShutDown, AutoIndent, InsertConstruct, InsertList, InsertTable, InsertImage etc. (*Command.java) All the commands

Class: ActiveContext	
Responsibilities:	This is the class that serves as a broad range of information for the commands to use. This class is kind of a surrogate of a Context pattern. The messy parameters and limitations for each command called for a organized way to pass around information. ActiveContext class contains information at a given point in time so that the command can use. The information included is the active JTextArea, the TabFrame Object, the index and any other attributes that may be needed.
Uses:	**Used By:** Commands

Class: App	
Responsibilities:	Main class, handles argument processing and execution of the main GUI and backend.
Uses: EditorFrame, CommandDistributor, FileManager	**Used by:** N/A

Class: Command Distrubutor	
Responsibilities:	Command Distributor works as a controller to determine which of the FileManager methods needs to be invoked in order to finish the command. It holds a firm reference to FileManager in case commands need to access it.
Uses: File Manager	**Used by:** CloseTab, Validate, New, Save, Open, ShutDown, AutoIndent, InsertConstruct, InsertList, InsertTable, InsertImage etc. (*Command.java) All the commands

Class: CloseTabCommand	
Responsibilities:	This command closes a tab when the user wishes to stop editing the current file.
Uses: Command, Command Distributor	**Used by:** MenuItem

Class: ValidateCommand	
Responsibilities:	Checks for well-formedness of an HTML file. For R2, the validation happens as a result of openning and saving due to the mediators help. Only once is it implicitly called from MenuItem (EditorFrame).
Uses: Command Distributor	**Used by:** CommandMediator, MenuItem

Class: NewFileCommand	
Responsibilities:	Creates a new file to be added to the GUI for HTML editing
Uses: Command, Command Distributor	**Used by:** MenuItem

Class: SaveCommand, SaveAsCommand	
Responsibilities:	Saves the file of the currently selected tab.
Uses: Command, Command Distributor, ValidateCommand	**Used by:** MenuItem

Class: OpenCommand	
Responsibilities:	Orchestrates the process of opening a file and ensuring all necessary objects know of the opened file.
Uses: Command, CommandDistributor, EditorFrame	**Used by:** MenuItem

Class: ShutdownCommand	
Responsibilities:	This command is launched on system exit and It checks if there are unsaved files. If there are unsaved files, it will ask whether the user would still like to quit even though there are unsaved files; allowing interruption of the exit sequence.
Uses: Command, Command Distributor	**Used by:** MenuItem

Class: AutoIndentCommand	
Responsibilities:	Handles inserting the correct number of tabs for a given line (if any).
Uses: Command, CommandDistributor	**Used by:** MenuItem

Class: InsertTableCommand	
Responsibilities:	Creates a well-formed HTML table element with the specified number of rows and columns.
Uses: Command, CommandDistributor, InsertTableDialog	**Used by:** MenuItem

Class: InsertConstructCommand	
Responsibilities:	Creates a well-formed HTML header, bold, italic or header (1-3) element.
Uses: Command, CommandDistributor	**Used by:** MenuItem

Class: InsertListCommand	
Responsibilities:	Creates a well-formed HTML list, numbered list, or dictionary list for the number of specified elements.
Uses: Command, CommandDistributor, SizeOfListDialog	**Used by:** MenuItem

Class: TabSelectedCommand	
Responsibilities:	Indents a large block of selected text a single tab.
Uses: Command, CommandDIstributor	**Used By:** MenuItem

Class: InsertATagCommand	
Responsibilities:	Command that prompts the user for the name and link of the <a> tag and then writes it into the current file.
Uses: EditorFrame	**Used By:** EditorFrame

In Release Two:

As indicated on the latest design - See latest UML Class Diagram

Class: Links	
Responsibilities:	The Links is a simple container for all the links within a file. It is just responsible for returning the links themselves and also for holding data for the Links Strategy pattern.
Uses: LinkStrategy	**Used By:** FileManager

Class: LinksListModel	
Responsibilities:	General model for the JList that displays the Links of a file. A model allows for future modification of the list in any way.
Uses:	**Used By:** TabFrame

Class: LinkStrategy	
Responsibilities:	This class declares the interface for the 2 strategies. Both strategies are just simple ways to sort the list for the links.
Uses:	**Used By:** Alpha and ConsecutiveOrder Strategy, FileManager

Class: Alpha and ConsecutiveOrder Strategy	
Responsibilities:	Both of these classes are implementations of LinkStrategy. They each return a different type of list depending on the desired type. Alpha returns the list in an alphabetically organized manner, and the latter returns the list in a order of appearance.
Uses: LinkStrategy	**Used By:** RefreshLinksCommand

Class: Command Mediator	
Responsibilities:	Command Mediator works as a nice way to know all the commands that are executed on a process that takes various commands. As an example, opening a file requires an openFile command, and then the mediator calls subsequent commands like the validate command and the refresh links command.
Uses: *Commands.java	**Used by:** *Commands.java , Any command that has subsequent commands.

Class: Undo Command	
Responsibilities:	This class is called whenever the user wants to undo to the previous state. The command distributor grants access to the FileManager who is in charge of choosing what file to undo and performs the action.
Uses: Command, CommandDistributor (HTMLFile indirectly)	**Used By:** MenuItem, EditorFrame

Class: CareTaker	
Responsibilities:	The responsibility of the CareTaker is to simply hold references to a set of mementos so that undo/redo's can be performed.
Uses: Memento	**Used By:** HTMLFile

Class: Memento	
Responsibilities:	The responsibility of the memento is simply to be the state of a single HTMLFile for undoing/redoing capabilities.
Uses:	**Used By:** HTMLFile, CareTaker

Class: InsertImageCommand	
Responsibilities:	Command responsible for prompting the user for the link of the image and then writing it into the file.
Uses: EditorFrame	**Used By:** EditorFrame

Class: RefreshLinksCommand	
Responsibilities:	This class is responsible for refreshing the view of the links that are in the document. This Command can be invoked explicitly or is called by the Mediator after some other commands are executed (i.e. it is executed after inserting a new A tag)
Uses: Links	**Used By:** EditorFrame, MenuItem

Class: Render Preview Command	
Responsibilities:	This class is an extra command that the group thought was excellent. This command pops up a view and parses the current HTML to see how the file looks like. It is an excellent way to see if the progress on the file is what is intended and also if all the tags are actually working the way the user intended to.
Uses:	**Used By:** EditorFrame

Class: Input Dialogs (*Dialog.java)	
Responsibilities:	These classes are the small dialogs that pop up whenever the system needs user input. For example, the URLDialog pops up when inserting an A tag to promp for a name and a url.
Uses: EditorFrame	**Used By:** EditorFrame

Class: View Outline Command	
Responsibilities:	Takes the current text in the active pane and displays it as an outline with code folding abilities. The user cannot edit the text while code folding is enabled.
Uses: Command, Command Distributor	**Used By:** CommandMediator

Class: Outline Editor Kit	
Responsibilities:	These are the third party files that are used. They are an implementation of an Editor Kit which is applied to the stock JEditorPane. This Actually parsed the html file and then displays an outline view.
Uses: TabFrame	**Used By:** TabFrame

Pattern Usage Description

In Release One:

GoF pattern:	Command	
Participants:	Command, MenuItem, CommandDistributor	
Class	**Role in pattern**	**Participant's contribution in the context of the application**
Command	Command	This is the interface we made for executing any command operation
New, Save, Open, CloseTab, Validate, ShutDown, AutoIndent, *Command	ConcreteCommand	Each of these classes does the command in the title
MenuItem	Invoker	When clicked, the menu commands will ask the specific concrete command to execute its command
CommandDistributor	Receiver	Distributes all of the ConcreteCommand excutable operations to the correct places
CommandMediator	Receiver	Calls subsequent commands after another command is executed. (i.e. Save is called, the mediator is in charge of validating and then updating links right after)
Deviations/Variations from the standard pattern:	The Command Mediator is also part of the execute method so that commands can be kept track of and subsequent commands are all called in one spot.	We sent a parameter to all of our execute methods in the concrete commands called the CommandDistributor. We needed to use this to fulfill every single command. Therefore because this parameter was “generic” in a sense that it was the same for each concrete command, we thought it was appropriate to send.

Requirements being covered:	● Many Commands ○ New/Open/Save ○ Insert tags ○ etc.	

GoF pattern:	Observer	
Participants	HTML File	GUI/Editor Pane
Class	**Role in pattern**	**Participant's contribution in the context of the application**
Editor Pane	ConcreteSubject	The Editor Pane is observed and notifies the HTML File if any changes are made to the file
HTML File	ConcreteObserver	The HTML File waits to be notified of changes, and then changes its “needs to be saved” variable to true, indicating a change has been made since the file has been opened
Deviations from the standard pattern:	We did not include a “Subject” Interface for the EditorPane. We just included all the functionality in the Concrete Subject instead.	For the HTML File, we implemented the “DocumentObserver” which gave us the necessary observer functionality
Requirements being covered:	● Sets save state of file to help the user check if file must be saved.	● Helps see changes to set a new memento for undoing/redoing

In Release Two:

GoF pattern:	Strategy	
Participants	Links, LinkStrategy, AlphaStrategy, ConsecutiveOrderStrategy	
Class	**Role in pattern**	**Participant's contribution in the context of the application**
Links	Context	This is the data (links) that we want to display using two different strategies
LinkStrategy	Strategy Interface	Interface for the two strategies so that they can be treated the same regardless of the type of strategy
AlphaStrategy	Concrete Strategy	This sorts the links alphabetically and returns the list for them to be displayed
ConsecutiveOrderStrategy	Concrete Strategy	This sorts the links by consecutive order that they appear in the file and returns the list for them to be displayed
Requirements being covered:	The different layouts for displaying the links (either alphabetical or in order)	

GoF pattern:	Mediator	
Participants	CommandMediator, Commands	
Class	**Role in pattern**	**Participant's contribution in the context of the application**
CommandMediator	ConcreteMediator	We notify the CommandMediator whenever a command is executed so that it can then do any other commands that are necessary.
Command	ColleagueBase	This is the Command interface.
Commands (many classes; see Command Pattern above for more specifics)	ConcreteColleague	All the commands are used as colleagues.
Deviations from the standard pattern:	We only have one mediator, so we didn’t make an interface	
Requirements being covered:	● Validating after save/open	● Refreshing links after insertion of A tag and after saving/opening too

GoF pattern:	Memento	
Participants	CareTaker, Memento, HTMLFile	
Class	**Role in pattern**	**Participant's contribution in the context of the application**
CareTaker	CareTaker	Responsible for creating a stack for each HTML File to save the previous states of the contents.
Memento	Memento	Holds the state of the HTMLFile
HTMLFile	Originator	Creates the Memento objects
Deviations from the standard pattern:		Originally, the caretaker only holds a reference. To tailor our needs, the care taker is not only a reference holder, but the data structure for our undo/redo capabilities. Stores more than one memento.
Requirements being covered:	• Undo/Redo save states	

Sequence Diagrams

Open a file:

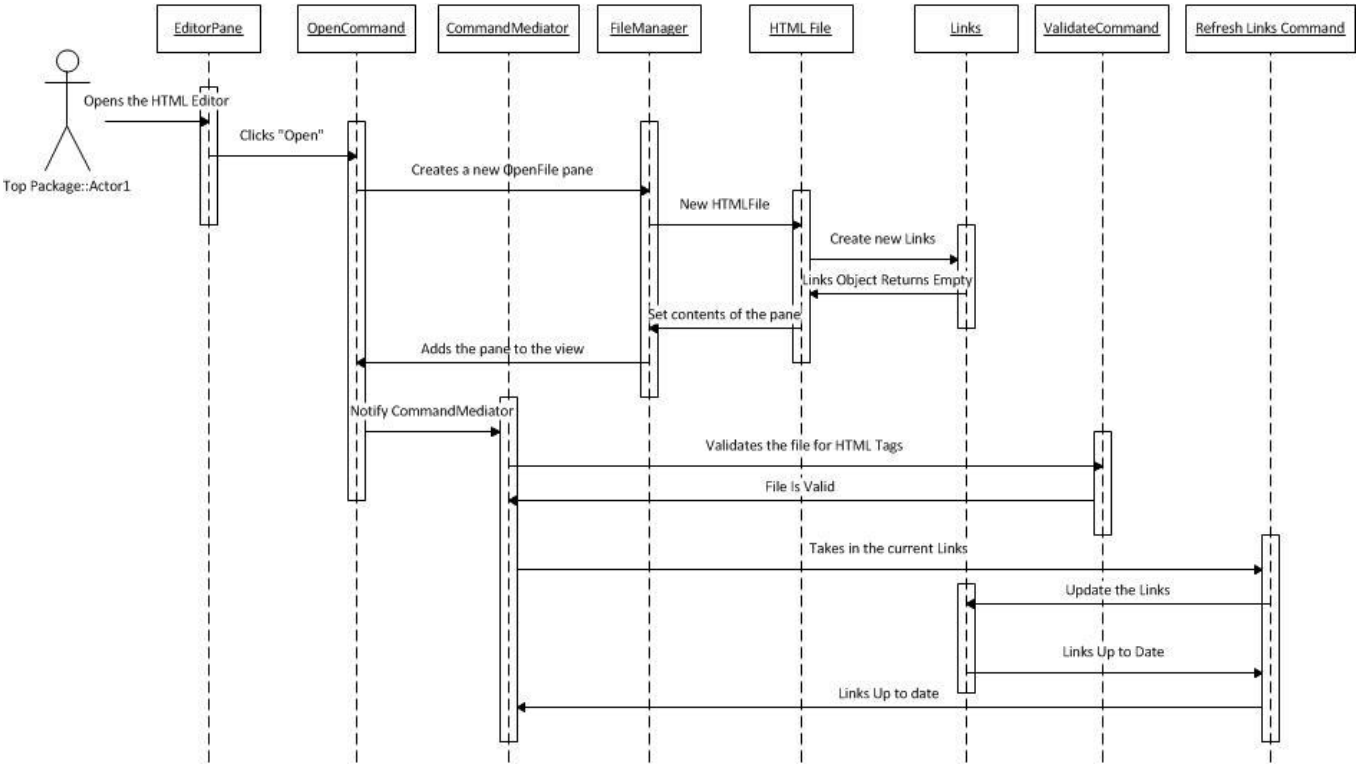


Figure 4 - OpenFile Sequence Diagram

When the User opens our HTML editor, the Editor Pane (our GUI) is the first thing that is accessed. Next, the user will click the Open File menu item under the File menu. When this happens, the OpenCommand object is accessed. This creates a new open file pane, and accesses the FileManager to create a new HTML File object. From here the HTML File creates its own links object to be updated throughout the editing session. Then the contents of the pane are set with the opened file's contents, and the pane is added to the main Editor Pane view. Once the OpenCommand actions are complete, notification is sent to the ComandMediator. In the Command Mediator class, any other commands that need to be triggered are told that Open Command has just been completed. For OpenCommand, this includes validating and updating the links. If the format is valid, there will be no notification to the user. If the file HTML tag format is not valid, then an error message will be displayed to the user notifying them of the problem that has occurred. The RefreshLinksCommand parses through the file and finds all the links in href tags. This is then updated in the view.

Insert an HREF tag:

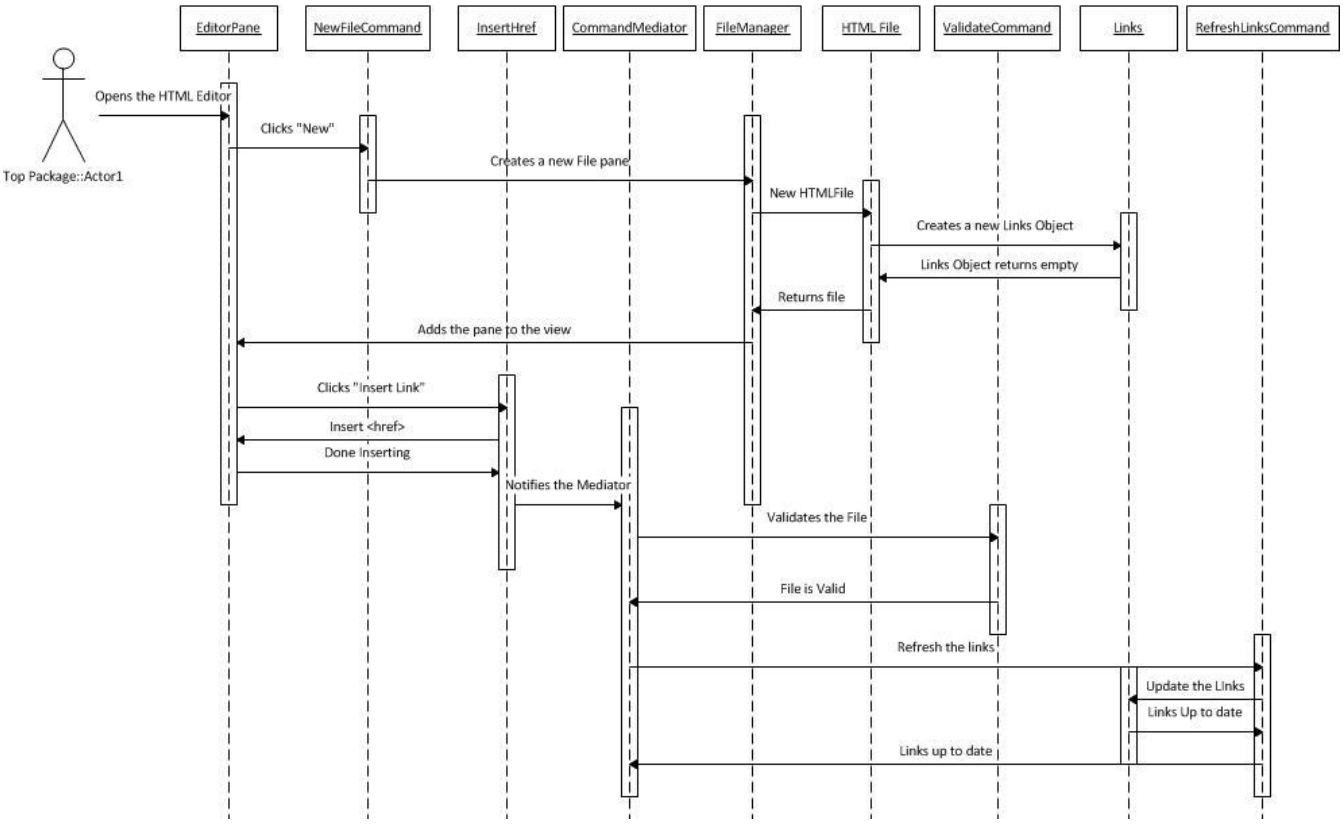


Figure 5 - Insert HREF Sequence Diagram

When the User opens our HTML editor, the Editor Pane (our GUI) is the first thing that is accessed. Next, the user will click the New File menu item under the File menu. When this happens, the NewFileCommand object is accessed. This creates a new file pane, and accesses the FileManager to create a new HTML File object. From here the HTML File creates its own links object to be updated throughout the editing session. Then the correctly formatted file is returned, and the pane is added to the main Editor Pane view. Once this is complete, the user can then type whatever they like. Later, the user will try to insert an a-tag and will do so by clicking the Insert Link menu item, accessing the InsertHrefCommand. From here the user will be provided a dialog box to choose their link and link name, and insert that text into the active pane. Next, the InsertHrefCommand notifies the CommandMediator that it's actions are completed. From here, CommandMediator requests that the file be validated. This command is sent to the ValidateCommand object. If the format is valid, a pop up box will notify the user that their file is valid. If the file's HTML tag format is not valid, then an error message will be displayed to the user notifying them of the problem that has occurred. Next, the RefreshLinksCommand parses through the file and finds all the links in href tags. This is then updated in the view.

Strengths and Weaknesses

Our overall design, like any design has both strengths and weaknesses.

For R2 it was proven that our design was solid to begin with. Additional features that were required could just take on a command class and have it work. For example, inserting more tags like images, was similar to inserting a table, but with different input. The low coupling of the components allowed us to further improve the program in such simple ways.

However, that isn't the forte of R2. In the second release the biggest design strength is probably the use of a Context and a Mediator. The context makes it even easier for the commands to demand whatever information they need in a safe way. The context object itself is only there to provide information and that is a highly decoupled way to pass arguments so that commands can do their thing. On the other hand, the use of the Mediator highly cleans up the execution of commands. Many commands need other commands to run and having to add that and keep track of that on a command basis was getting messy. So the mediator simply knows what gets called after each specific command. It is the embodiment of making a simple list of what commands need what commands and going to the mediator class to write it in and have it a go.

On top of the previous strengths, the simplification of the EditorFrame class was also a success. In R1, editor frame was very messy and cluttered; it had no sense of encapsulation and was a large GUI class with behaviors. For R2 the EditorFrame has split up into 3 files each of which are implementations of Swing's classes. An example is the TabFrame which holds the visible aspects of the editor; such as the text area and the list for links view. This allows for components that go together be wrapped around a class like such.

Our original design, however, was not all fantastic and had some weaknesses. First off, the back end representation of the file was a simple file with text and some attributes. There was no composite pattern involved or any structured way to hold the tags, since it seemed limiting back for Release 1. Now, when we wanted to implement code folding, all solutions pointed towards a hack solution rather than an effective robust one. As a result, the implementation was an elegant mask with no reusable foundation.

Implementation Status

Table 1 - Requirements R1

Feature	Planned for R1	In R1?	Test passed?
Point-and-click method for browsing directories to open file	Yes	Yes	Yes
Name of HTML file can be specified in command line args	Yes	Yes	Yes
Multiple files for editting	Yes	Yes	Yes
User can select which file is being viewed	Yes	Yes	Yes
Insert standard HTML constructs	Yes	Yes	Yes
Validation when loading file	Yes	Yes	Yes
Validation when saving file	Yes	Yes	Yes
Validation menu option	Yes	Yes	Yes
Able to abort save or save even if there are mistakes	Yes	Yes	Yes
Warns if unsaved changes are present when exiting	Yes	Yes	Yes
Supports auto-wrap	Yes	Yes	Yes
Toggle option for auto-wrap	Yes	Yes	Yes
Auto-indent of new HTML constructs	Yes	Yes	Yes
Able to specify indent spacing	Yes	Yes	Yes
Able to indent current line	Yes	Yes	Yes
Auto-wrap brings texts to same level of indentation	Yes	No	No

Table 2 - Requirements R2

Feature	Planned for R2	In R2?	Test passed?
Undo Operation (More than 3)	Yes	Yes	Yes
Undo to Last Operation	Yes	Yes	Yes
Undo to at least 2 Last Operations	Yes	Yes	Yes
Prior Operation: Insertion of HTML tag	Yes	Yes	Yes
Prior Operation: Cut / Copy	Yes	Yes	Yes
Prior Operation: Paste	Yes	Yes	Yes
Prior Operation: Text Entry / Deletion	Yes	Yes	Yes
Prior Operation: Cursor Motion	Yes	Yes	Yes
Prior Operation: Element Outlining	Yes	Yes	Yes
Support: A Tags with HREF fields	Yes	Yes	Yes
Support: IMG Tags with SRC fields	Yes	Yes	Yes
HTML Image Viewer	Yes	Yes	Yes
HTML Text Viewer	Yes	Yes	Yes
Link View: URL: Order Appearance	Yes	Yes	Yes
Link View: URL: Initial Protocol Tag	Yes	Yes	Yes
Link View: Update when Insert HREF	Yes	Yes	Yes
Link View: Update Manually with Refresh	Yes	Yes	Yes
Support: Outline Mode: Expand by Hierarchical	Yes	Yes	Yes
Support: Outline Mode: Collapse by Hierarchical	Yes	Yes	Yes

All of the requirements have been satisfied with the exception of:

- The tabulation given change in wrap text
    - The ability for the wrapped text to be tabbed with the beginning of the line.

Aside from the single unmet requirement we also have a list of additional features that we have implemented:

- The ability to close tabs
    - With and without a button
- A help menu item.
- An ‘about the authors’ menu item
- A Readme Document
- Insert various other tags.
    - Header tags etc.
- Not only undo but redo capabilities
- Undo/Redo not limited to 2 times
- Redo only works when you have created a new file, instead of opening a previously created file

As non-functional requirements, there was a large focus on usability. There are popup dialogs that all follow the same format when a value from the user is needed. Shortcuts and accelerators are enabled for fast usage and follow the common practice.

Known Issues

Table 3 - Bugs Found R2 Only

Bug Name	Details	Recreation	Date Found	Fixed? On
Selecting Links view across files	When changing the selection of the type of view for the links list, the actual list only changes for the current file	Open 2 files with links. Change the list view to alphabetical on one. Swap to the other file and will show that alphabetical is selected but not displayed.	4/14	yes 4/15
InsertConstruct command eat tabs (white space)	When inserting header, bold, italics (New ConstructCommand), if there are multiple empty tabs before it, it will eat them and use them between the tags.	New file > tab around 10 times > put in a header command. The tags get shifted although not incorrect, it is unexpected.	4/1	no
Quick save doesn't work	If a file has been saved already, it should not pop up a save file chooser	Open a file, save it, and save it again.	4/10	yes 4/14
Outline view repeats	Outline view shows the repeated	Toggle between outline and normal mode multiple times	4/15	yes 4/15
Blank Outline mode	If validation fails, outline mode shows nothing	Create a non-valid file (i.e. leave off a closing tag)	4/16	no

Table 4 - Testing R1 and R2

Test Case	Alpha Testing	Beta Testing	R1/R2 Final Testing
Editor Application	Yes 3/22/2013	Yes 2/23/2013	Yes 3/27/2013
Editor TextField	Yes 3/22/2013	Yes 2/23/2013	Yes 3/27/2013
HTML Writer Buffer	No 3/22/2013	Yes 3/27/2013	Yes 3/27/2013
Multiple Files	No 3/22/2013	Yes 2/23/2013	Yes 3/27/2013
HTML Tags/Construct	No 3/22/2013	No 3/23/2013	Yes 3/27/2013
HTML Indentation	No 3/22/2013	No 3/23/2013	Yes 3/27/2013
Menu Items Bar	Yes 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: File: New	Yes 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: File: Open	No 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: File: Save	No 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: File: Save As	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: File: Closed Tab	No 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: File: Quit	No 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Menu Items: Edit: Copy	No 4/9/2013	No	Yes 4/16/2013
Menu Items: Edit: Paste	No 4/9/2013	No	Yes 4/16/2013

Menu Items: Edit: Undo	No 4/9/2013	Yes 4/16/2013	Yes 4/17/2013
Menu Items: Edit: Redo	No 4/9/2013	No 4/16/2013	Partial 4/17/2013
Test Case	**Alpha Testing**	**Beta Testing**	**R1/R2 Final Testing**
Menu Items: Insert: Header	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: Insert: Table	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: Insert: Font	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: Insert: List	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: Insert: Image	No 4/9/2013	Yes 4/16/2013	Yes
Menu Items: Insert: Link	No 4/9/2013	Yes 4/16/2013	Yes
Menu Items: Options: Tab Width	No 4/9/2013	Yes 3/24/2013	Yes 3/27/2013
Menu Items: Options: Wrap Text	No 4/9/2013	Yes 3/24/2013	Yes 3/27/2013
Menu Items: Options: Auto Indents	No 4/9/2013	Yes 3/24/2013	Yes 3/27/2013
Menu Items: Options: View Links	No 4/9/2013	Yes 3/24/2013	Yes 3/27/2013
Menu Items: Options: Links View Options	No 4/9/2013	Yes 4/16/2013	Yes 4/17/2013
Menu Items: HTML: Preview	No 4/9/2013	Yes 4/16/2013	Yes 4/17/2013
Menu Items: HTML: Refresh Links	No 4/9/2013	Yes 4/16/2013	Yes 4/17/2013
Menu Items: About: Authors	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Menu Items: About: Help	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Shortcut: New	No 3/22/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Open	Yes 3/23/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Save	Yes 3/23/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Save As	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Shortcut: Validate	No 4/9/2013	Yes 4/16/2013	Yes 4/17/2013
Shortcut: Close Tab	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Shortcut: Quit	No 3/22/2013	No 3/24/2013	Yes 3/27/2013
Shortcut: Copy	Yes 3/23/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Paste	Yes 3/23/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Undo	No 4/9/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Redo	No 4/9/2013	Yes 3/23/2013	Yes 3/27/2013
Shortcut: Preview	No 4/9/2013	Yes 4/16/2013	Yes 3/27/2013
Shortcut: Refresh Links	No 4/9/2013	Yes 4/16/2013	Yes 3/27/2013

After one of our final iterations, there were bugs that were fixed. However a few have been noticed but not addressed due to time constraints. Those involve:

- No matter what the user sets as tab spaces, whenever re-launching the editor, the tab spaces will reset to default = 4.
- When highlighting text and tabulating, all the selected text gets tabulated but then is un-highlighted

Refactoring Command Survey Summary

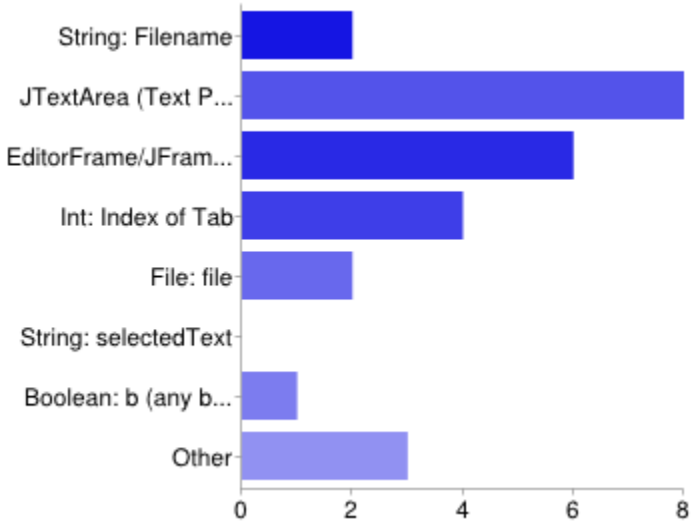
After R1, the team quickly realized that refactoring had to be done. In order to start, we put together a survey that would be answered by each command class. We asked them what parameters they needed and so it allowed us to see what was most important.

For example, the NewFileCommand needed the filename, the file, etc.

This ultimately showed us that these parameters had no real value and cluttered the code. As a result, we implemented a Context Class (Could be interpreted as context pattern). This class was a limited copy of the EditorFrame. From there, most of the overly used parameters could be extracted in a safe manner and reducing the clutter of code. This also simplified the worries of changing multiple areas of the command execute method.

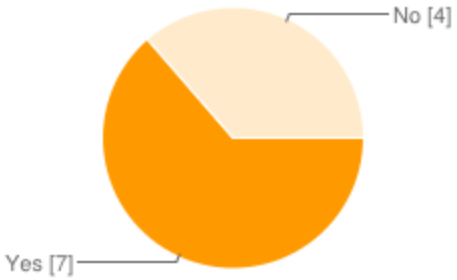
Below is graphic of the data we collected. Notice that many command required the same things over and over again, it only made sense to give them a superset of those parameters.

Check the required parameters for the given Command:



String: Filename	2	8%
JTextArea (Text Pane)	8	31%
EditorFrame/JFrame (The main frame)	6	23%
Int: Index of Tab	4	15%
File: file	2	8%
String: selectedText	0	0%
Boolean: b (any boolean)	1	4%
Other	3	12

Does the Command actually use the getFileManager() method?



Yes	7	64%
No	4	36%