LATEX EDITOR VIEW

OVERALL REPORT

VERSION <2.8>

Tucaliuc Agnes Monalisa, 336

Joanis Prifti, 3321

TABLE OF CONTENTS

INTRODUCTION

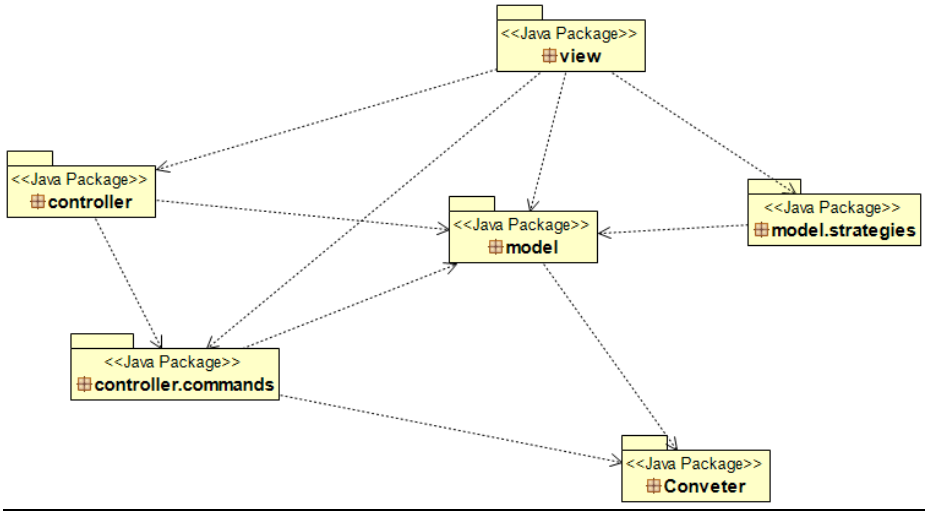
Extensibility is a design paradigm in software engineering where the implementation takes future growth into account. Versatility in this project refers to the Latex templates, commands, and version tracking systems that are provided. The application is particularly designed to make adding new templates and tactics as simple as possible while adhering to the well-known open-closed philosophy. Furthermore, usability in software engineering refers to the simplicity of use and learnability. The application enables pattern specification using a simple and user-intuitive interface in the context of this project. The program provides user instructions in the form of user guides that explain the application's key features and the contents of the various pattern templates.

To be more specific, this project's purpose is to reengineer and enhance a Java application that already exists. For beginner Latex users, the application provides a straightforward Latex editor. Latex is a well-known markup language for creating high-quality documents. It comes with a plethora of styles and commands for advanced document formatting. Formatting papers with Latex is akin to programming because it necessitates the correct use of Latex commands embedded in the document's text. The Latex editor's objective is to make it easier to use Latex commands to create Latex documents. One of the features that sets the LatexEditor apart from other similar apps is its multi-strategy version tracking capabilities, which allow you to undo and redo actions.

REFACTORED DESIGN

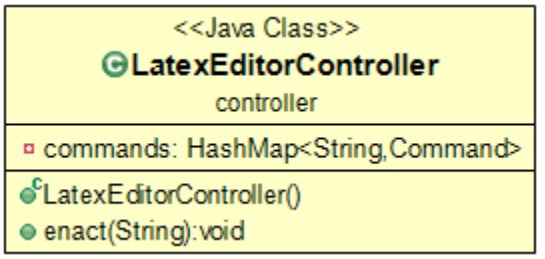
ARCHITECTURE

Package UML:

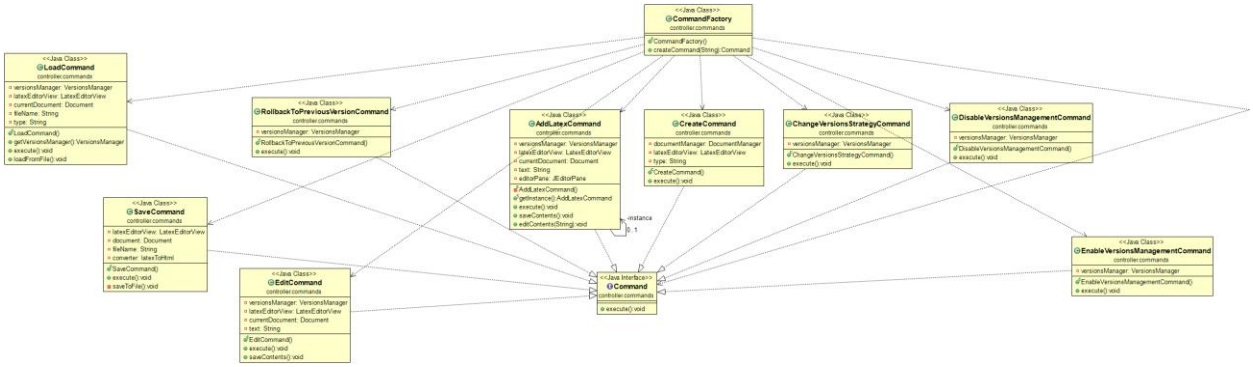


DETAILED DESIGN

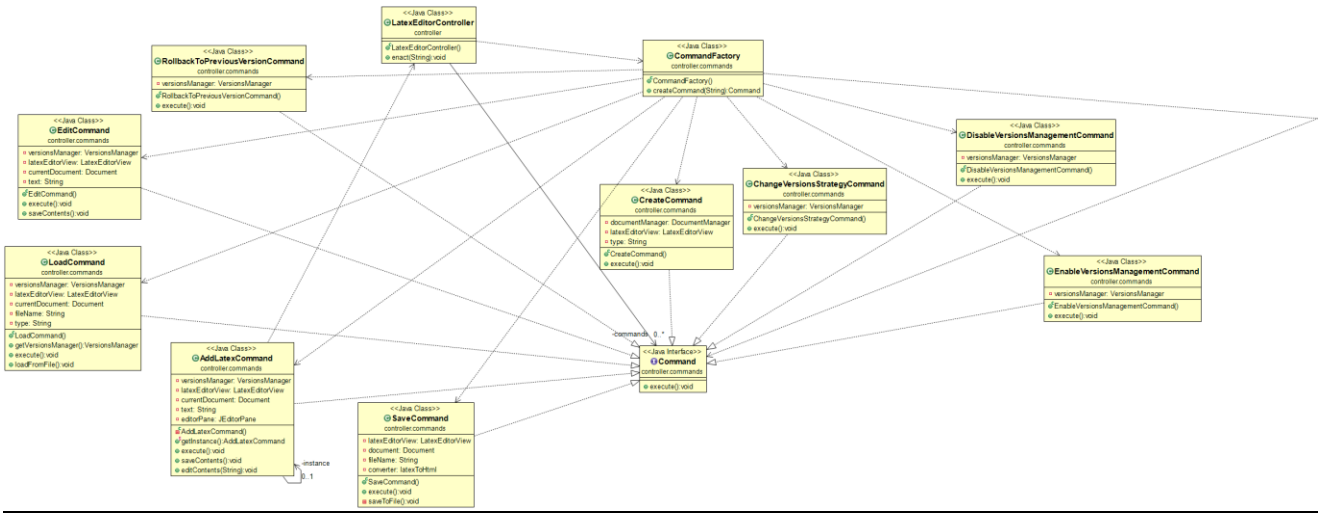
- Package controller UML:



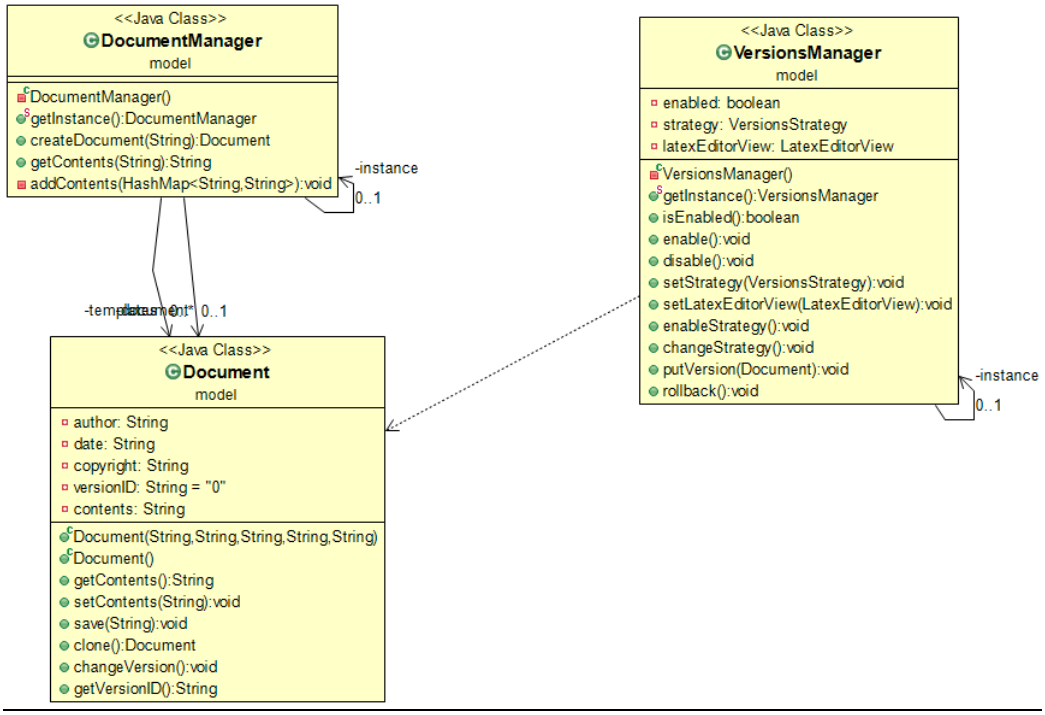
- Package controller.commands UML:



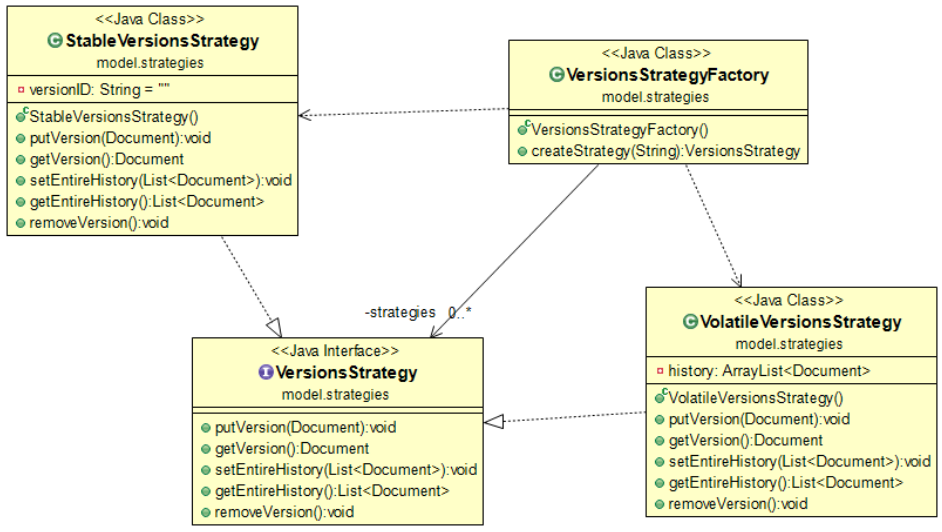
- Package controller and controller.commands UML:



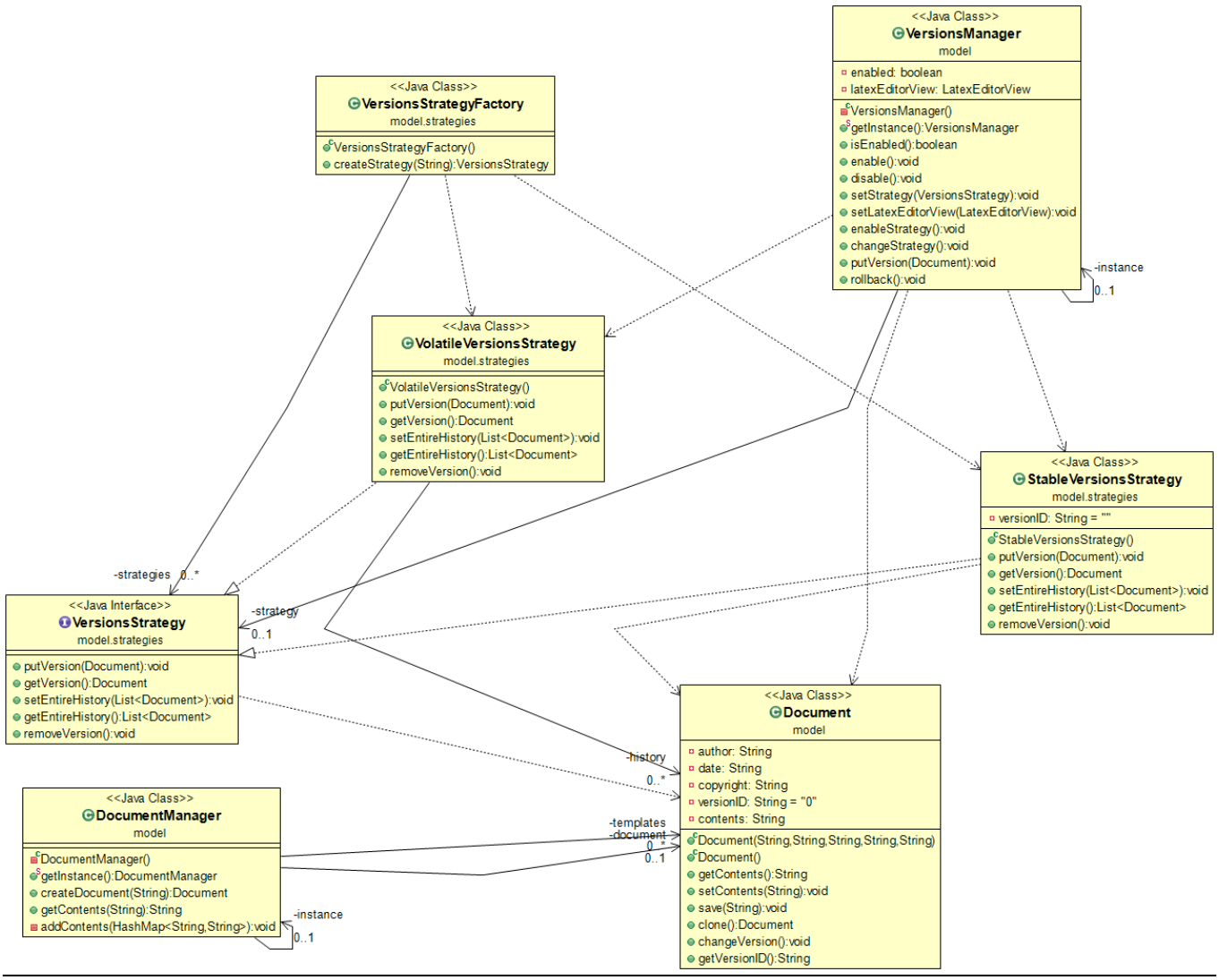
- Package model UML:



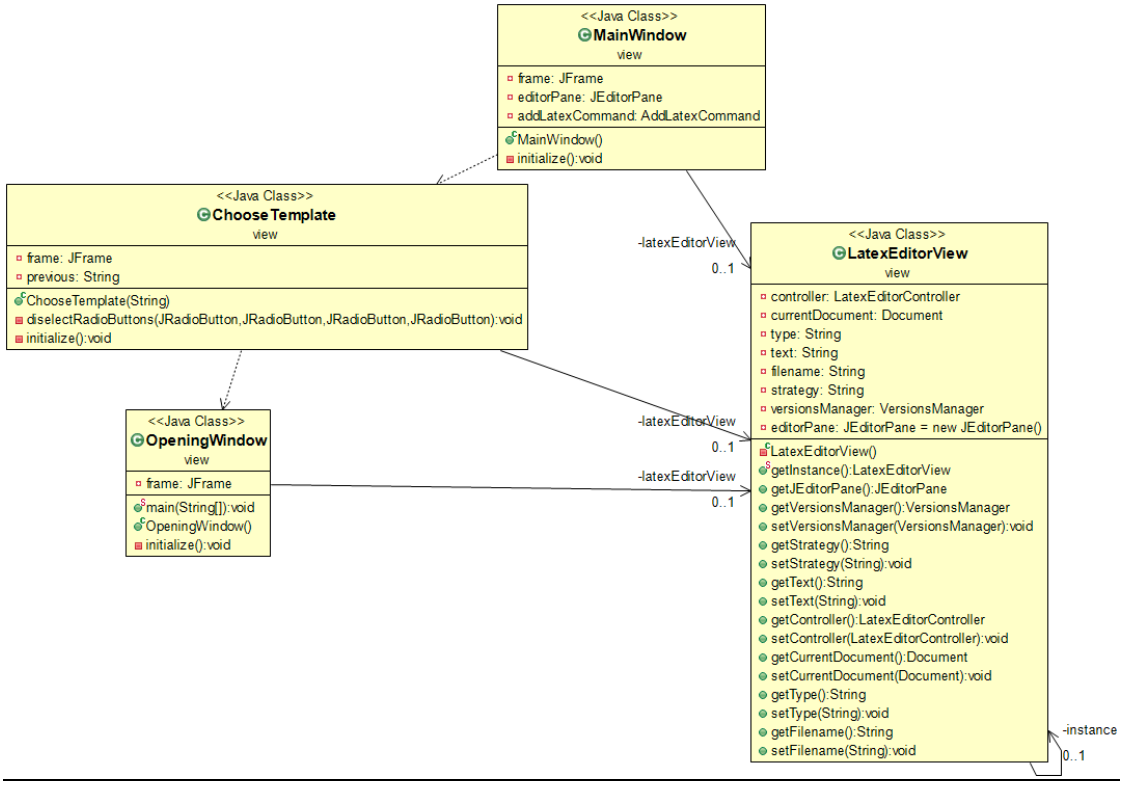
- Package model.strategies UML:



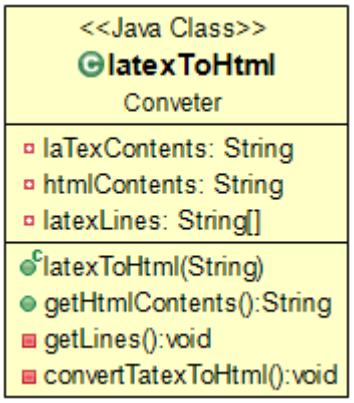
- Package model and model.strategies UML:



- Package view UML:



- Package converter UML:



First, we examined and analyzed the project to find the points in the code that needed improvement and aggregation. Specifically in the first phase we took a cursory look over the paperwork, learned more about the application's design and use cases by studying the documentation. Then we performed a mock installation, set up the project's operating version, and attempted to use some of its basic

features. After we understanded the legacy architecture, the role/responsibilities of each class we came to these alternations in phase 2.

In phase 2 we started refactoring these specific classes. For your information, refactoring is the process of improving the internal structure of current source code while maintaining the program's exterior behavior. Refactoring is usually motivated by noticing a code smell. For example, the method at hand may be very long, or it may be a near duplicate of another nearby method. Once recognized, such problems can be addressed by refactoring the source code or transforming it into a new form that behaves the same as before but that no longer "smells". Bellow we will describe our changes.

We started with LatexEditorController class where the constructor of the class has a lot of duplicate code, which is an issue. To populate a HashMap, the same instructions are used again and again. The Substitute Algorithm refactoring is one solution to this problem. We stored the options in a string array and looped over the file contents to create the required Command objects with the help of the HashMap.

Secondly, in the controller. commands package we spotted duplicated code that exists in several classes that implement the Command interface. We searched the classes thoroughly to see what improvements we can make. What we noticed is that in some circumstances, this manifests itself as the classes sharing the same fields. There are also similar methods between the classes in some circumstances. Using the Singleton pattern, we solved the problem of common fields is to give single points of access to the key objects that are utilized by the instructions. One idea to work around the common fields problem is to provide unique access points for key objects used by directives using the Singleton pattern. We noticed that almost every class carried in their constructor the VersionsManager class, so we converted it in a Singleton class and we called an instance of it in the controller classes that needed it. Because we blended refactoring patterns, we made additional changes that we will detail later. Please read through all of the modifications to gain a better understanding of the enhancements we made.

Furthermore, we found duplication code in the constructor of the DocumentManager class. Another issue is the long method getContents(), in particular, is a large method with a clear duplication code issue. So we used the Substitute Algorithm refactoring as a solution to these problems To build the requisite Document objects, we placed the latex template names and contents in a map class field and looped over the elements of the map, we got rid of a lot of repetition and the code is more clear and understandable. Lastly, we transformed it into a Singleton class because need it to get the current document which is used by the whole project. So we made it easier to refer at the same document everywhere in the code.

Next in VersionsManager class we detected a number of issues. We deleted dead code, which is code that does nothing and isn't used anywhere. Despite the fact that the class is little in terms of lines of code, it might be considered a Large Class because it has a lot of unrelated responsibilities. It's evident from the class's name that version management is its primary function. Specifically VersionsManager class acts as a Middle Man with several methods, as getType(), loadFromFile(), saveToFile(), saveContents() that already exist in LatexEditorView class. We removed them and called them directly from their source, that’s why we made it a Singleton class to get access to the same object to access the current document version.

To continue with LatexEditorView class which is responsible for a lot of changes in the project. Its title and responsibilities are incompatible. The view package comprises classes that deal with the graphical user interface and data visualization and it has nothing to do with any of these. Nevertheless, it provides application functionality, including methods for saving a document to a file, loading a document from a file, and utilizing the VersionsManager to create a new version of the current document. To overcome this problem, we used the Move Method and Move Field refactorings to disperse the class's duties to the appropriate classes. The SaveCommand class now contains the code that saves a document to a file (saveToFile()), whereas the LoadCommand class now contains the code that loads a document from a file (loadFromFile()). The logic for saving versions (saveContents) has been transferred to the EditCommand class. Besides the code, fields of LatexEditorView were altered and relocated to other classes. So we did the appropriate alternations to the classes that we moved these methods.

Another class we changed is MainWindow, which is the main GUI for the project. This class, however, provides a Large Method that should be included in the application logic. The editContents() function, in particular, is a misplaced responsibility that can be refactored to the AddLatexCommand class using the Move Method refactoring. In order to move it to the AddLatexCommand we made the LatexEditorCommand a singleton class because connect both MainWindow and AddLatexCommand and we need in some way to have access to the JEditorPane to supply both of the connected classes. That's why we initialized it in LatexEditorCommand and we added a get JEditorPane method there. To finish with the refactorings because the editContents() function was too big we used the Substitute Algorithm refactoring to make it smaller and remove the duplicated code.

As for the extensibility of the project we extended the application by adding some new functionalities. Firstly, we added a new file extension type the .html int the MainWindow class, in order to be able to save the latex file as an html file. To be more specific, we wanted the user to be able to save a latex document as an Html document to convert the file into a web page. We started by creating a new package, the converter package, which includes two classes, the HtmlToLatex class and the latexToHtml class. Let's start by explaining the latexToHtml class. This class is responsible for the converting role, what we mean by this is that we read the contents of the template that the user chose (Book, Report, Article, Letter, or blank document) and we replace the key words of the latex structure with the html key words. This happens by reading line by line and finding the key word, once we find it we check it match in html, we noticed that the Letter Template needed special treatment but we fount it's way around. Particularly different sections and subsections become corresponded to latex headings, latex figures, lists and highlighting text also have been mapped to corresponding html elements. Below are the matches:

Sectioning commands:

HTML	**LATEX**
<h1>	\chapter
<h2>	\section
<h3>	\subsection
<h4>	\subsubsection
<h5>	\subparagraph

<p>	\par

Lists:

HTML	**LATEX**
Numbered list:	\begin{enumerate}
Unnumbered list:	\end{itemize}
List element:	\item
Description list: <dl>	\begin{description}
Term: <dt>	\item
Definition: <dd>	text

Highlighting text:

HTML	**LATEX**
Emphasis: text	\emph{text}
Italic: text	\textif{text}
Bold: text>	\textbf{text}
Fixed with: <tt>text<tt>	\texttt{text}

To verify our implementation we got help from this link: https://pandoc.org/try/?text=%5Cdocumentclass%5B11pt%2Ca4paper%5D%7Breport%7D%0A%0A%5Cbegin%7Bdocument%7D%0A%5Ctitle%7BReport+Template%3A+How+to+Structure+a+LaTeX+Document%7D%0A%5Cauthor%7BAuthor1+%5Cand+Author2+%5Cand+...%7D%0A%5Cdate%7B%5Ctoday%7D%0A%5Cmaketitle%0A%0A%5Cbegin%7Babstract%7D%0AYour+abstract+goes+here...%0A...%0A%5Cend%7Babstract%7D%0A%0A%5Cchapter%7BIntroduction%7D%0A%5Csection%7BSection+Title+1%7D%0A%5Csection%7BSection+Title+2%7D%0A%5Csection%7BSection+Title.....%7D%0A%0A%5Cchapter%7B....%7D%0A%0A%5Cchapter%7BConclusion%7D%0A%0A%0A%5Cchapter*%7BReferences%7D%0A%0A%5Cend%7Bdocument%7D%0A&from=latex&to=html5&standalone=0

So we added a conditional statement in SaveCommand class , method saveToFile() to check if the user is saving the file as an Html file or not. If user does then we automatically call the latexToHtml class with its convert method, we read the latex contents from the JEditorPane and we convert it to html structure type and save the file as an .html file, otherwise as default the file is saved as an latex file.

Bonus part

While checking the project we detected some malfunctions. One of them was the "Open Existing Document" JButton in Opening Window class. We removed it but in case the user wants to open an existing file, the user can open it from the "New File" JMenuItem in the MainWindow class. Another alternation was the "frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);" we deleted this command in order to enable the exit (right up X button) choice in the OpeningWindow class and in the ChooseTemplate class.

IMPLEMENTATION

- For each class give a brief description in terms of a CRC card (see the format below)

Class Name: LatexEditorController	
Responsibilities	**Collaborations**
▪ Fills HashMap with command names and actions ▪ Executes commands	▪ CommandFactory

Class Name: AddLatexCommand	
Responsibilities	**Collaborations**
▪ Creates JEditorPane ▪ Gets current document	▪ VersionsManager ▪ LatexEditorView

▪ Adds contents in JEditorPane	▪ Document ▪ Implements Command Interface

Class Name: ChangeVersionsStrategyCommand	
Responsibilities	**Collaborations**
▪ Executes strategies(stable,volatile)	▪ VersionsManager ▪ Implements Command Interface

Class Name: Command	
Responsibilities	**Collaborations**
▪ Interface	▪ AddLatexCommand ▪ ChangeVersionsStrategyCommand ▪ CreateCommand ▪ DisableVersionsManagementCommand ▪ EditCommand ▪ EnableVersionsMnagementCommand

	▪ LoadCommand ▪ RollbackToPreviousVersionCommand ▪ SaveCommand

Class Name: CommandFactory	
Responsibilities	**Collaborations**
▪ Creates commands based on type	▪ AddLatexCommand ▪ ChangeVersionsStrategy ▪ CreateCommand ▪ DisableVersionsManagementCommand ▪ EditCommand ▪ EnableVersionsManagementCommand ▪ LoadCommand ▪ RollbackToPreviousVersionCommand ▪ SaveCommand

Class Name: CreateCommand	
Responsibilities	**Collaborations**
▪ Gets type ▪ Creates document type ▪ Sets the current document	▪ DocumentManager ▪ LatexEditorView ▪ Implements Command Interface

Class Name: DisableVersionsManagementCommand	
Responsibilities	**Collaborations**
▪ Enables the disable	▪ VersionsManager ▪ Implements Command Interface

Class Name: EditCommand	
Responsibilities	**Collaborations**
▪ Gets current document ▪ Gets text ▪ Checks if is enabled and changes	▪ Implements Command Interface ▪ VersionsManager

version ▪ Sets contents in current document	▪ LatexEditorView ▪ Document

Class Name: EnableVersionsMnagementCommand	
Responsibilities	**Collaborations**
▪ Enables the strategies(stable, volatile)	▪ Implements Command Interface ▪ VersionsManager

Class Name: LoadCommand	
Responsibilities	**Collaborations**
▪ Gets version manager ▪ Gets current document ▪ Gets filename ▪ Scans lines of the file contents ▪ Sets contents in current document ▪ Sets template type	▪ Implements Command Interface ▪ VersionsManager ▪ LatexEditorView ▪ Document

Class Name: RollbackToPreviousVersionCommand	
Responsibilities	**Collaborations**
▪ Executes rollback	▪ Implements Command Interface ▪ VersionsManager

Class Name: SaveCommand	
Responsibilities	**Collaborations**
▪ Saves .tex file ▪ Saves converted latex file as an html file	▪ Implements Command Interface ▪ LatexEditorView ▪ Document ▪ latexToHtml

Class Name: Document	
Responsibilities	**Collaborations**
▪ Initializes author, date, copyright, versionID, contents ▪ Gets and sets contents	

▪ Saves filename ▪ Clones document ▪ Changes version ▪ Gets version	

Class Name: DocumentManager	
Responsibilities	**Collaborations**
▪ Selects template ▪ Creates new documents based on template ▪ Creates document and clones it ▪ Add content based on template type	▪ Document

Class Name: VersionsManager	
Responsibilities	**Collaborations**
▪ Enables ▪ Disables	▪ VersionsStrategy ▪ LatexEditorView

▪ Sets strategy ▪ Sets LatexEditorView ▪ Creates enableStrategy(), changeStrategy(), rollback(), putVersion()	

Class Name: StableVersionsStrategy	
Responsibilities	**Collaborations**
▪ Puts version on document ▪ Gets version of document ▪ Sets entire document history ▪ Gets entire document history ▪ Removes version documents	▪ Implements VersionsStrategy Interface ▪ Document

Class Name: VersionsStrategy	
Responsibilities	**Collaborations**
▪ Interface	▪ StableVersionsStrategy

	▪ VolatileVersionsStrategy

Class Name: VersionsStrategyFactory	
Responsibilities	**Collaborations**
▪ Creates HashMap and fills it with strategies	▪ VersionsStrategy ▪ VolatileVersionsStrategy ▪ StableVersionsStrategy

Class Name: VolatileVersionsStrategy	
Responsibilities	**Collaborations**
▪ Puts version on document ▪ Gets version of document ▪ Sets entire history of document ▪ Removes version	▪ Implements VersionsStrategy Interface ▪ Document

Class Name: ChooseTemplate	
Responsibilities	**Collaborations**

▪ Initializes the JFrame ▪ Initializes the JRadioButton ▪ Adds action listeners to the buttons ▪ Creates JLabels ▪ Creates JButtons ▪ Adds action listeners to the JButtons ▪ Adds elements to the frame	▪ LatexEditorView

Class Name: LatexEditorView	
Responsibilities	**Collaborations**
▪ Initializes the JEditorPane for the template window ▪ Gets JEditorPane, VersionsManager, Strategy, Text, Controller, CurrentDocument, Type, Filename ▪ Sets VersionsManager, Strategy, Text, Controller, CurrentDocument, Type, Filename	▪ LatexEditorController ▪ Document ▪ VersionsManager

Class Name: MainWindow	
Responsibilities	**Collaborations**
▪ Initializes new JFrame ▪ Creates a JMenuBar ▪ Creates JMenu and JMenuItem ▪ Adds action listeners to JMenuItems	▪ LatexEditorView ▪ AddLatexCommand

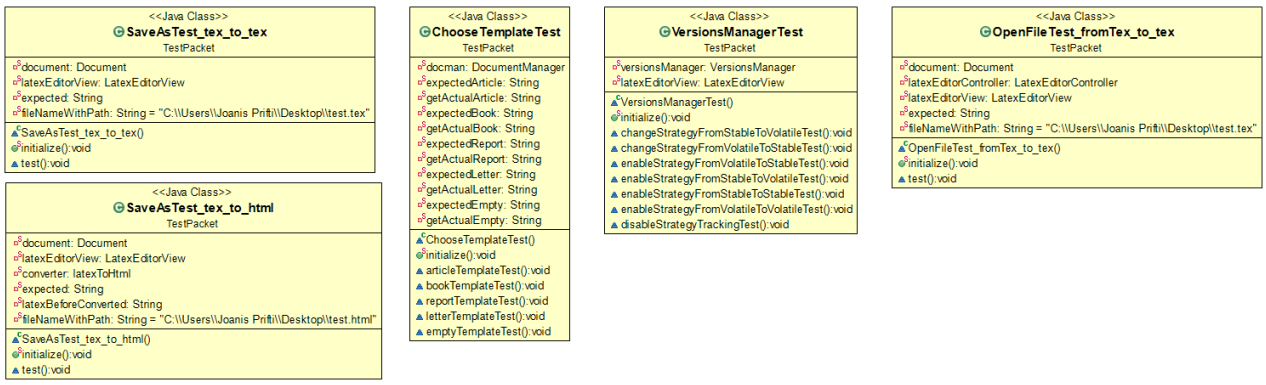
Class Name: OpeningWindow	
Responsibilities	**Collaborations**
▪ Has the main ▪ Opens first window/JFrame ▪ Creates JButons ▪ Adds actions listeners	▪ LatexEditorView ▪ VersionsStrategy ▪ VersionsManager ▪ LatexEditorController

Class Name: latexToHtml	
Responsibilities	**Collaborations**

▪ Gets html contents ▪ Splits latex contents ▪ Converts latex contents to html contents	

ACCEPTANCE TESTS

The architecture for the acceptance test is in packet TestPacket. The Uml is below:



[US-1] SaveAsTest_tex_to_tex

Description: Saves a latex document on disk storage

Methods To Test: setContents(String contents), save(String filename) which are in Document class and setCurrentDocument(Document currentDocument), setFilename(String filename) which are in LatexEditorView class.

Assertion: Checks if the saved document contains indeed what we saved inside it.

[US-2] SaveAsTest_tex_to_html

Description: Converts a latex document and saves it as an html document

Methods To Test: converterLatexToHtml() which is in latexToHtml class and save(String filename) which is in Document class

Assertion: Checks if the content in the saved is converted right.

[US-3] OpenFileTest_fromTex_to_tex

Description: Opens a latex file.

Methods To Test: enact(String Command) which is in LatexEditorController class

Assertion: Checks if the contents of the file are the expected ones.

[US-4] ChooseTemplateTest

Description: Gets contents of the template the user chose.

Methods To Test: getContents(String type) which is in DocumentManager class

Assertion: Compares if the template the user chooses contains the right latex lines.

[US-5] VersionsManagerTest

Description: Change and enable strategies (volatile and stable)

Methods To Test: changeStrategy() , isEnabled(), enableStrategy() which are in VersionsManagers class and execute() which is in DisableVersionsManagementCommand class.

Assertion: Checks if the strategy is indeed changed and if the right strategy is indeed enabled.