LATEX EDITOR PROJECT

REENGINEERING THE LEGACY CODE

1.

	Date 10/11/2021

GOAL OF THE PROJECT

The goal of this project is to reengineer and extend an existing Java application. The application is a simple Latex editor for inexperienced Latex users. Latex is a well known high quality document preparation markup language. It provides a large variety of styles and commands that enable advanced document formatting. Typically, a Latex document is compiled with a tool like MikTex, Lyx, etc. to produce a respective formatted document in pdf, ps, etc. Formatting documents with Latex is like a programming process as it involves the proper usage of Latex commands which are embedded in the document contents. The goal of the Latex editor is to facilitate the usage of Latex commands for the preparation of Latex documents. One of the prominent features that distinguishes the LatexEditor from other similar applications is its multi-strategy version tracking functionalities that enable undo and redo actions. More information about Latex:

[1] General information about the Latex project https://www.latex-project.org/
[2] Download MikTex - Latex distribution - https://miktex.org/
[3] LaTeX/Document Structure https://en.wikibooks.org/wiki/LaTeX/Document_Structure

PHASE 1 TASK LIST

2. Skim the documentation: The legacy application has been developed based on a more detailed requirements specification that is available along with the application source code (LatexEditorRequirementsDefinition.pdf). In a first step, study the documentation to get more information concerning the application's architecture and use cases.

3. Do a mock installation: The application source code is provided as an eclipse project (LatexEditorProject-2022 folder). Setup a running version of the project and try to use its basic functionalities.

4. Build confidence: Read all the source code once and try to understand the legacy architecture, the role/responsibilities of each class, and so on.

5. Prepare test cases for the user stories supported by the application.

6. Capture the design: Specify the legacy architecture in terms of a UML package diagram. Specify the detailed design in terms of UML class diagrams. Prepare CRC cards that describe the responsibilities and collaborations of each class.

PHASE 2 TASK LIST

	Date 10/11/2021

REFACTORING TASKS

1. **LatexEditorController class**: A problem here is in the constructor of the class, which has a lot of **Duplicate Code.** The same commands are repeated to populate a HashMap. An idea to deal with this issue is to use the Substitute Algorithm refactoring. Specifically, you can store the string command names in a map class field and loop over the elements of the map to create the required command objects. Another possibility is to have the command names in a properties file and loop over the file contents to create the required Command objects.

2. **Command classes:** A problem that is easy to spot here is that the different classes that implement the Command interface have **Duplicate Code**. In some cases, this appears in the form of same fields between the classes. In other cases there are also common methods between the classes. An idea to solve the problem of common fields is to provide single points of access to the key objects that are used by the commands using the Singleton pattern. More information about the singleton pattern can be found at:

   - https://refactoring.guru/design-patterns/singleton/java/example
   - https://www.cs.uoi.gr/~zarras/se-notes/slides-new-versions-20-4-20.zip (αρχειο 4-SoftEng-Design.pdf)

3. **DocumentManager class**: In this class we have **Duplicate Code** in the constructor. Another problem is **Long Method**. In particular, getContents() is a huge method that also has a clear **Duplicate Code** problem**.** An idea to deal with these issues is to use the Substitute Algorithm refactoring. Specifically, you can store the latex template names and their contents in a map class field and loop over the elements of the map to create the required Document objects. Another possibility is to have the templates as files and loop over the file contents to create the required Document objects.

4. **VersionsManager class:** This class has several problems. The simplest problem is **Dead Code**, methods that do nothing and are not used anywhere. Remove the useless methods. Although the class is not very big in terms of lines of code, it can be seen as a **Large Class** because it has many unrelated responsibilities. From the name of the class it is clear that its main responsibility is version management. However, it also has methods for saving and loading documents to/from files. Another problem that is evident is **Message Chains**. In particular, VersionsManager is a Middle Man with several methods (getType, saveToFile(), loadFromFile(), saveContents()....) that simply delegate invocations to corresponding methods of the LatexEditorView class. A typical way to get rid of these problems is to remove the delegating methods using the Remove Middle Man refactoring.

5. **LatexEditorView class**: This class is strange. Its name and responsibilities do not match. The view package contains classes that have to do with the GUI and the data visualization. LatexEditorView does not have anything to do with these. Instead, it contains application logic and specifically it has methods that realize some basic commands like saving a document to a file, loading a document from a file, creating a new version

of the current document using the VersionsManager. An idea to solve this problem is to redistribute the responsibilities of the class to the right classes using the Move Method and the Move Field refactorings. In particular, the code that saves a document to a file (saveToFile()) can be moved to the SaveCommand class, the code that loads a document from a file (loadFromFile()) can be moved to the LoadCommand class. The code that creates versions (saveContents) can be moved to the EditCommand class or to the AddLatexCommand class. Along with the code you may need to change/move fields of LatexEditorView to other classes. Other methods and fields that possibly remain can be moved to the LatexEditorController class which is supposed to serve as middle layer that decouples the GUI from the model classes.

6. **MainWindow class:** This is the main GUI class. However, this class also contains a **Large Method** that should be part of the application logic. Specifically, the editContents() method is a misplaced responsibility that can be moved to the AddLatexCommand class using the Move Method refactoring. The editContents() method also has a lot of **Duplicate Code** that can be removed.

EXTENSION TASKS

The second goal of this phase is to extend the application by adding new functionalities and respective tests. In particular, we want to realize the following two user stories:

7. **As a user I want to be able to save a latex document as an HTML document so as to make the document available as a web page.** Specifically, different sections and subsections should become correspond latex headings. Latex figures, tables, lists and enumerations should also be mapped to corresponding html elements.

8. **As a user I want to be able to load an html document and transform it to a latex document so at to be able to edit the document with the latex editor.**

PREPARE DELIVERABLES

7. A **report** based on the given template (Project-Deliverable-Template.doc). For the delivery of the report include a pdf of the report in the project folder.

8. A **DEMO video**, about 15' minutes, using a screen capture tool like ActivePresenter for example. In the demo you should illustrate that the user stories of the application are still working after the refactoring. During the demo further explain in the code how you dealt with the reengineering problems of the application. Moreover you should state which of the problems you DID NOT handle. Finally, demonstrate the execution of the JUnit tests that you prepared for the project.

9. For the delivery of the demo make an account - if you do not have one - on GitHub (or Google drive, youtube, etc). Upload the demo video. In the project folder that you turnin include a txt file named **DEMO-LINK.txt** that contains the link that points to the video.

10. Turn in the **refactored project** and the other deliverables using **turnin deliverables@mye004 <your-project>.zip**, where your-project is a zip file of your Eclipse refactored project.