LATEX EDITOR PROJECT

PHASE 2 - REENGINEERING THE CODE

	Date 9/30/2019

GOAL OF THE PROJECT

The goal of this project is to reengineer and extend an existing Java application. At a glance, the objective of this project is to develop 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.

REFACTORING/REENGINEERING TASKS

The first goal of this phase is to refactor the application so as to fix the following problems:

controller package:

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 in the disk and loop over the file contents to create the required Command objects.

controller.commands package:

***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 is to use refactorings like Extract Super Class or Inline Class.

model package:

1. **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 in the disk and loop over the file contents to create the required Document objects.

2. **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 if 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.

view package:

1. **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.

2. **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:

- [XUS-1] As a user I want to save an encrypted form of the latest version of the Latex document on disk storage. The application should let me select a particular encryption strategy that is going to be used before saving an encrypted form of the Latex document. In particular, the application should support at least the following strategies:
    - Atbash: The Atbash cipher is formed by taking the alphabet and mapping it to its reverse, so that the first letter becomes the last letter, the second letter becomes the second to last letter, and so on.
    - Rot-13: Rot-13 is a letter substitution cipher that replaces a letter with the 13th letter after it, in the alphabet. Rot-13 is a special case of the Caesar cipher, which was developed in ancient Rome.
- [XUS-2] As a user I want to load an encrypted form the latest version of the Latex document from stable storage. The application should decrypt the document to enable further document editing actions.

HINT - Ideally the new functionalities should be implemented with respect to the current design of the application. Therefore, the new functionalities should be implemented as new implementations of the Command interface.

[PREPARE REPORT]

Extend the report of Phase 1 with the new design of the legacy application, along with a discussion that explains how you dealt with the all the problems (Project-Deliverable-Phase2.doc).