LATEX EDITOR

DESIGN RECOVERY AND QUALITY ASSESSMENT REPORT

VERSION <1.0>

Βασιλειάδης Μιλτιάδης 2944

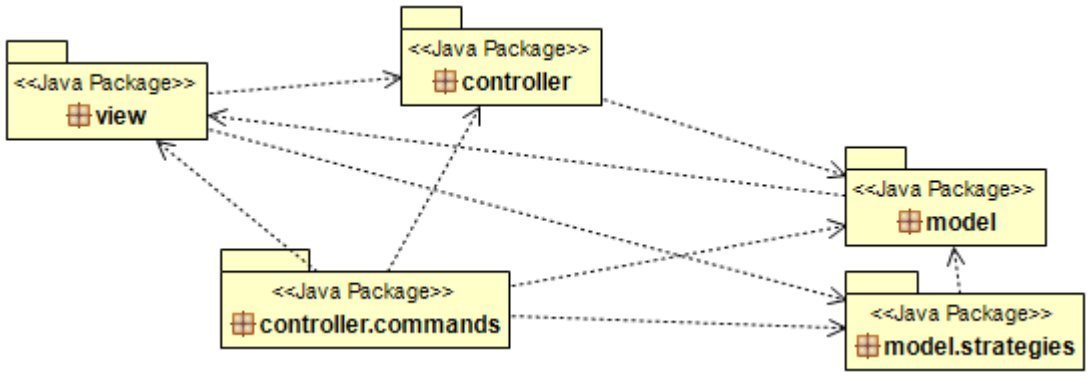
Θωμά Αθανάσιος 2979

INTRODUCTION

The goal of this project is to reengineer a 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.

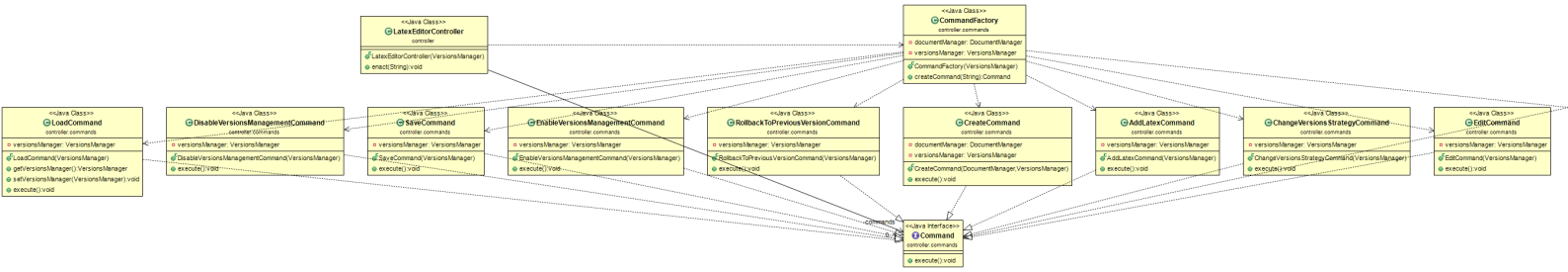
DESIGN RECOVERY

ARCHITECTURE

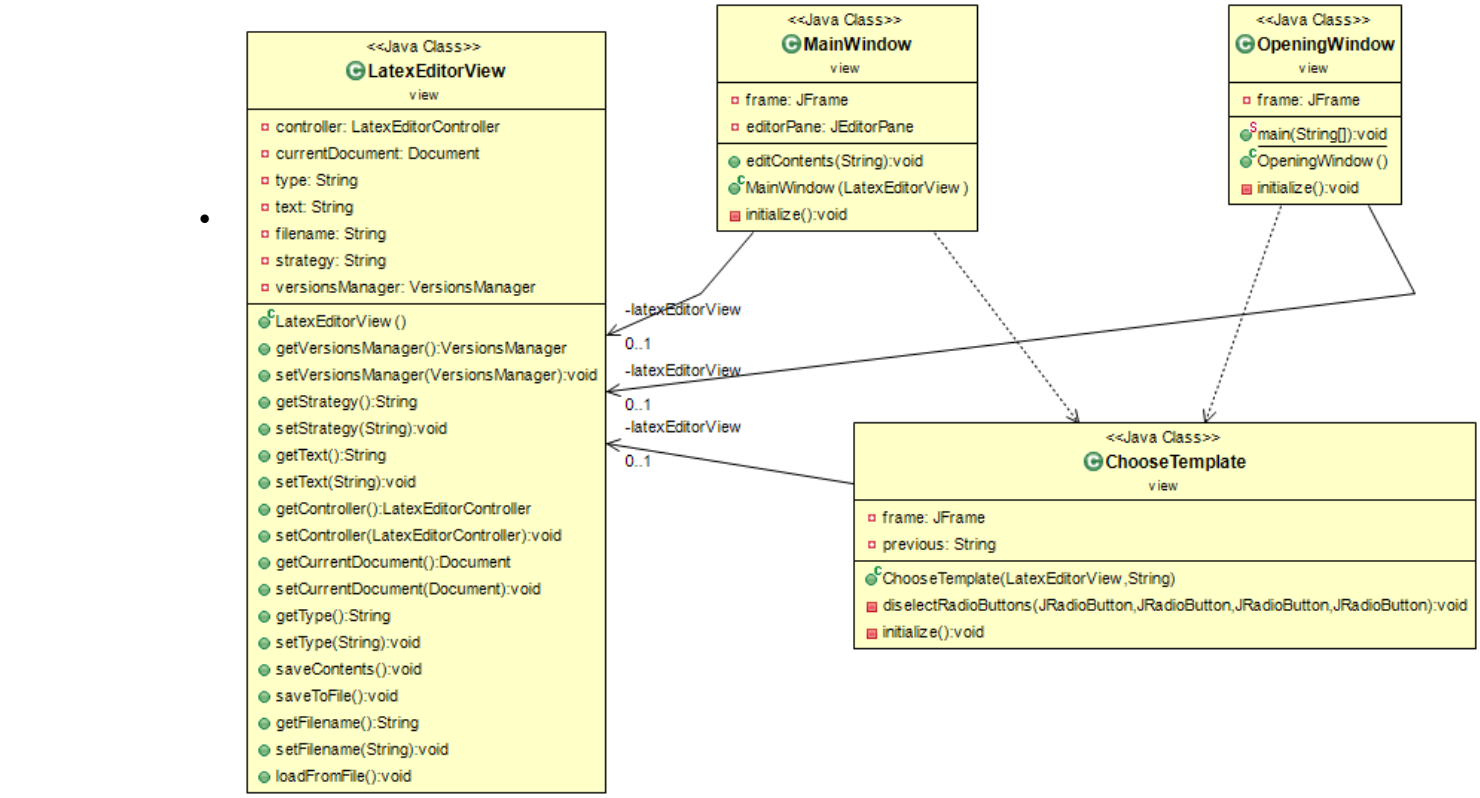


DETAILED DESIGN

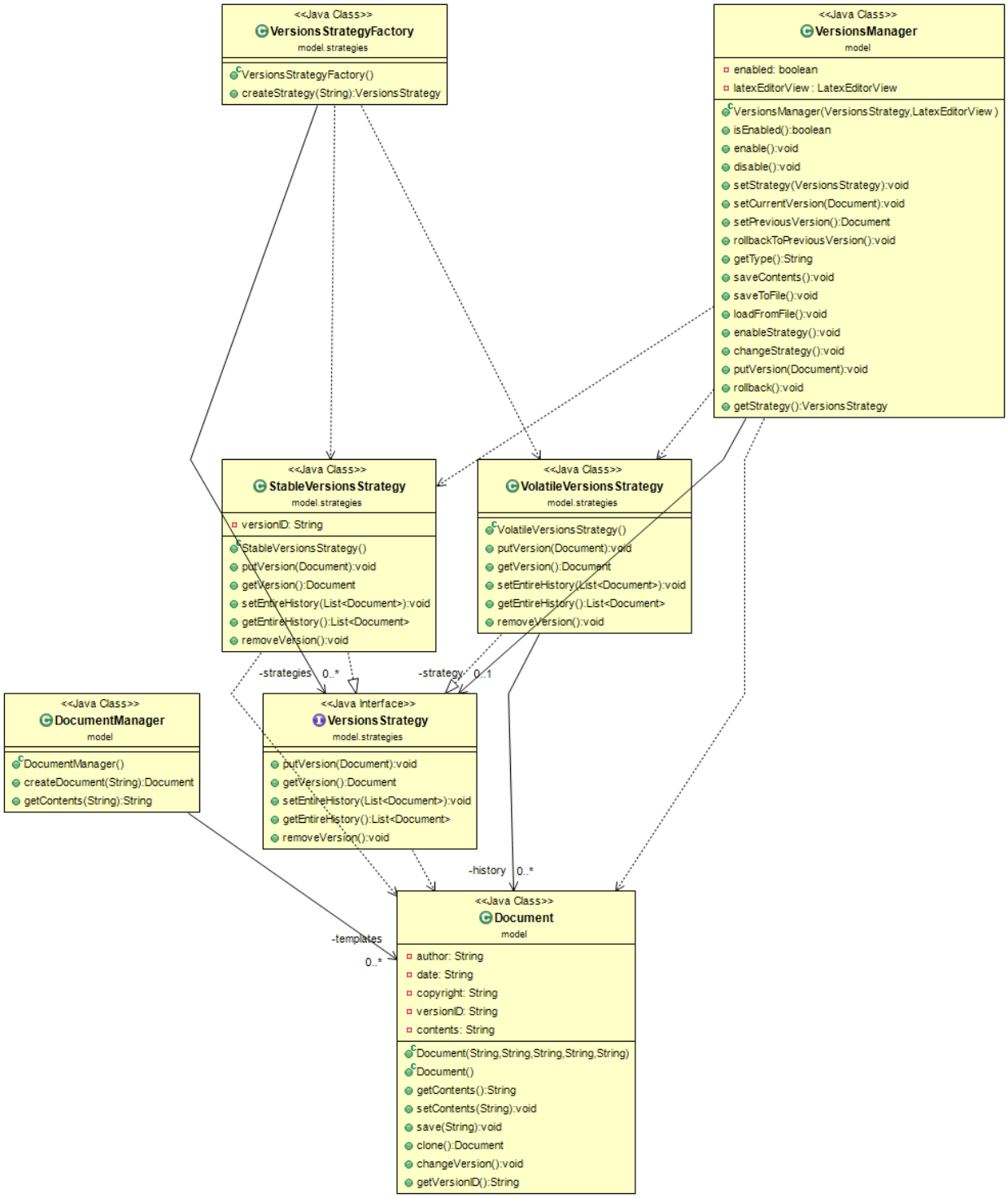
- controller package



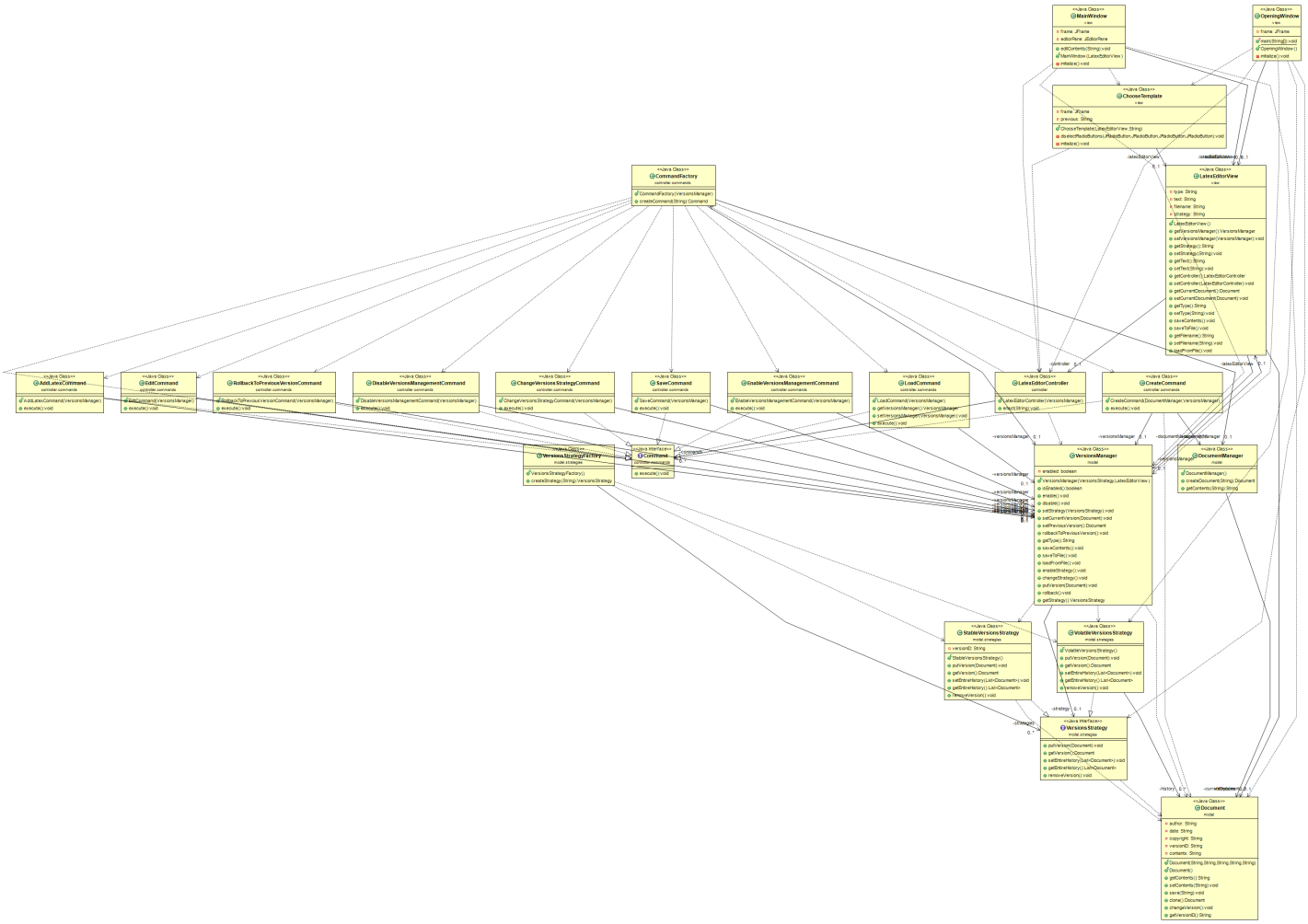
- view package



- model package


<<Java Class>>
VersionsStrategyFactory
model.strategies
VersionsStrategyFactory()
createStrategy(String):VersionsStrategy
<<Java Class>>
VersionsManager
model
enabled: boolean
latexEditorView : LatexEditorView
VersionsManager(VersionsStrategy,LatexEditorView)
isEnabled():boolean
enable():void
disable():void
setStrategy(VersionsStrategy):void
setCurrentVersion(Document):void
setPreviousVersion():Document
rollbackToPreviousVersion():void
getType():String
saveContents():void
saveToFile():void
loadFromFile():void
enableStrategy():void
changeStrategy():void
putVersion(Document):void
rollback():void
getStrategy():VersionsStrategy
<<Java Class>>
StableVersionsStrategy
model.strategies
versionID: String
StableVersionsStrategy()
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
<<Java Class>>
VolatileVersionsStrategy
model.strategies
VolatileVersionsStrategy()
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
-strategies
0..*
-strategy
0..1
<<Java Class>>
DocumentManager
model
DocumentManager()
createDocument(String):Document
getContents(String):String
<<Java Interface>>
VersionsStrategy
model.strategies
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
-history
0..*
-templates
0..*
<<Java Class>>
Document
model
author: String
date: String
copyright: String
versionID: String
contents: String
Document(String,String,String,String,String)
Document()
getContents():String
setContents(String):void
save(String):void
clone():Document
changeVersion():void
getVersionID():String

- overall graph



IMPLEMENTATION

QUALITY ASSESSMENT

Problematic Methods:

- Document.clone() ~ creates shallow instead of deep copy.
- DocumentManager.constructor() ~ Document constructor is not used correctly, instead the contents of each document template is filled through setter method.
- VersionsManager.rollbackToPreviousVersion() ~ does nothing.
- StableVersionsStrategy.putVersion() ~ calls Document.save() method instead of implementing its own mechanism to write to the disk.
- LoadCommand ~ has setter and getter methods. Dead code.
- ChooseTemplate.initialize() ~ has duplicate code inside.
- LatexEditorView ~ has setter and getter methods. Dead code.

Classes with many responsibilities:

~Model Package

- Document: also has save() method.
- DocumentManager: getContents() method should be in Document class.
- VersionsManager: is a God class. It is not only responsible for the versioning system. This is the class that all the command classes send their job to. It also handles some of the communication with the GUI.

~View Package

- MainWindow: method editContents() should be in AddLatexCommand class.
- LatexEditorView: this is the “controller” class of the GUI package. Responsibilities such as loadFromFile() and saveContents() are shoved in here. Has some duplicate code.

Classes with very few responsibilities:

~Controller Package

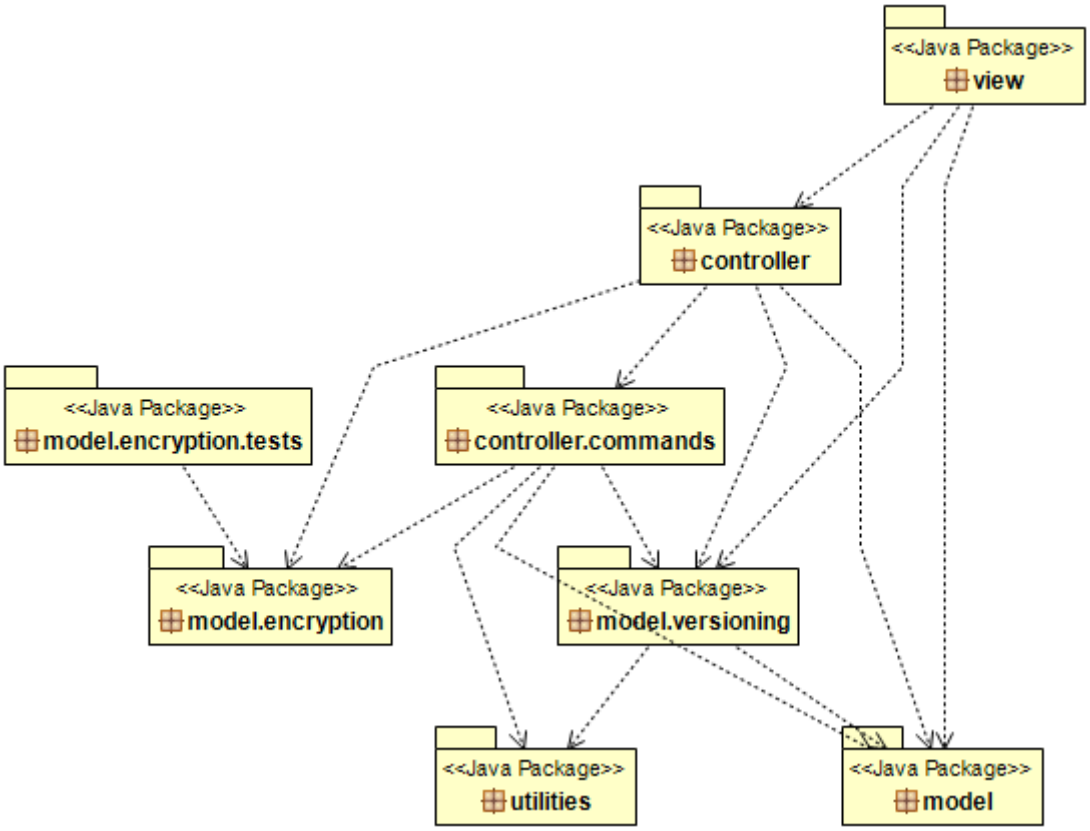
- LatexEditorController: this class should be the one communicating with the GUI.

~Controller.Commands Package

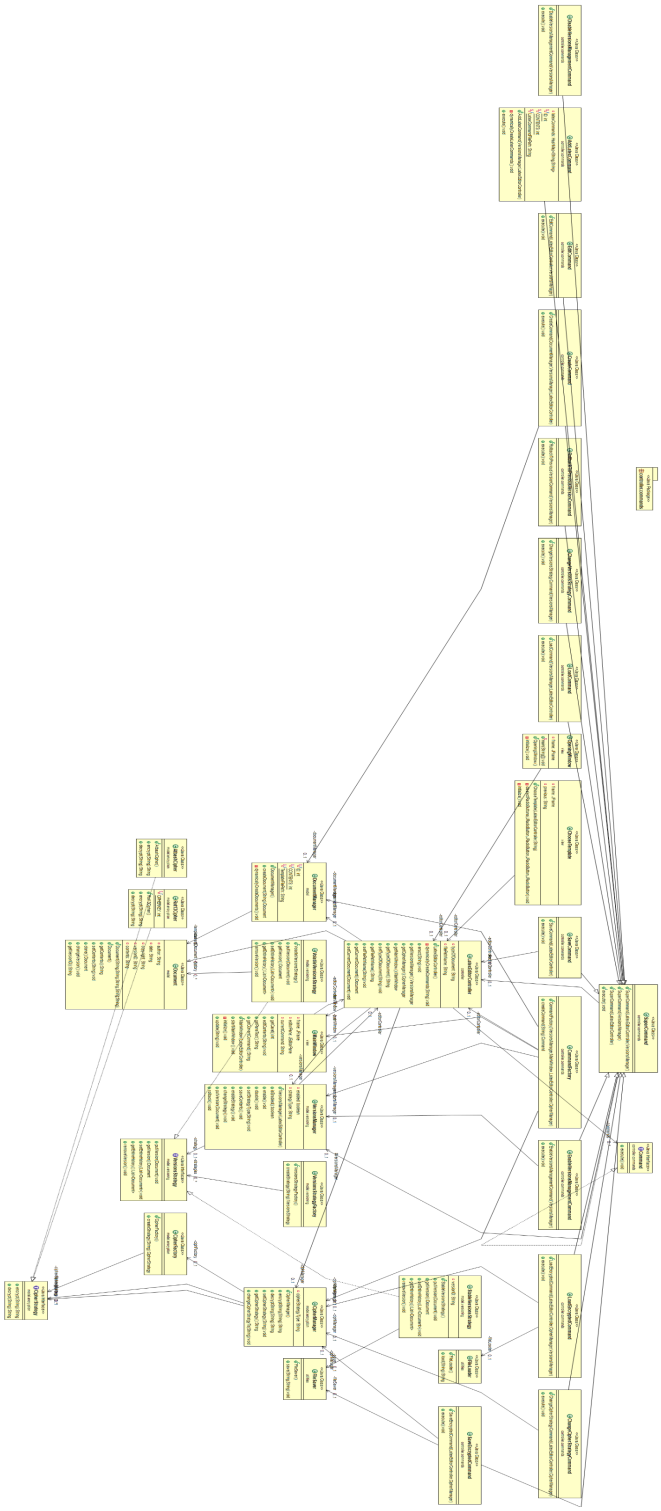
- AddLatexCommand: the job of this class is implemented elsewhere. The whole class is a duplicate of EditCommand.
- EditCommand: doesn't do anything, just calls the VersionsManager.
- ChangeVersionsStrategyCommand: doesn't do anything, just passes calls VersionsManager.
- LoadCommand: doesn't do anything, just calls the VersionsManager.
- RollbackToPreviousVersionCommand: doesn't do anything, just calls the VersionsManager.
- SaveCommand: doesn't do anything, just calls the VersionsManager.

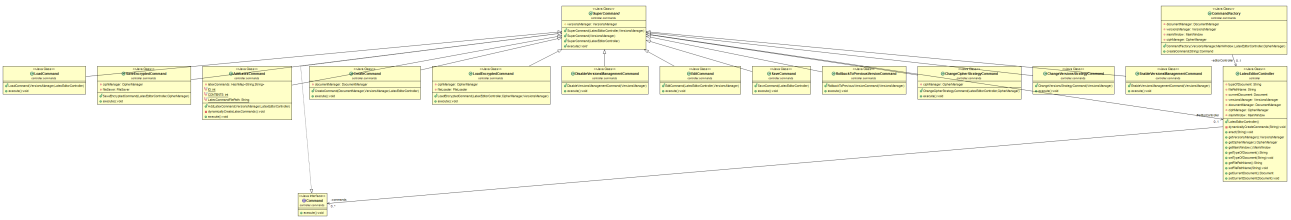
Re-Engineered Design

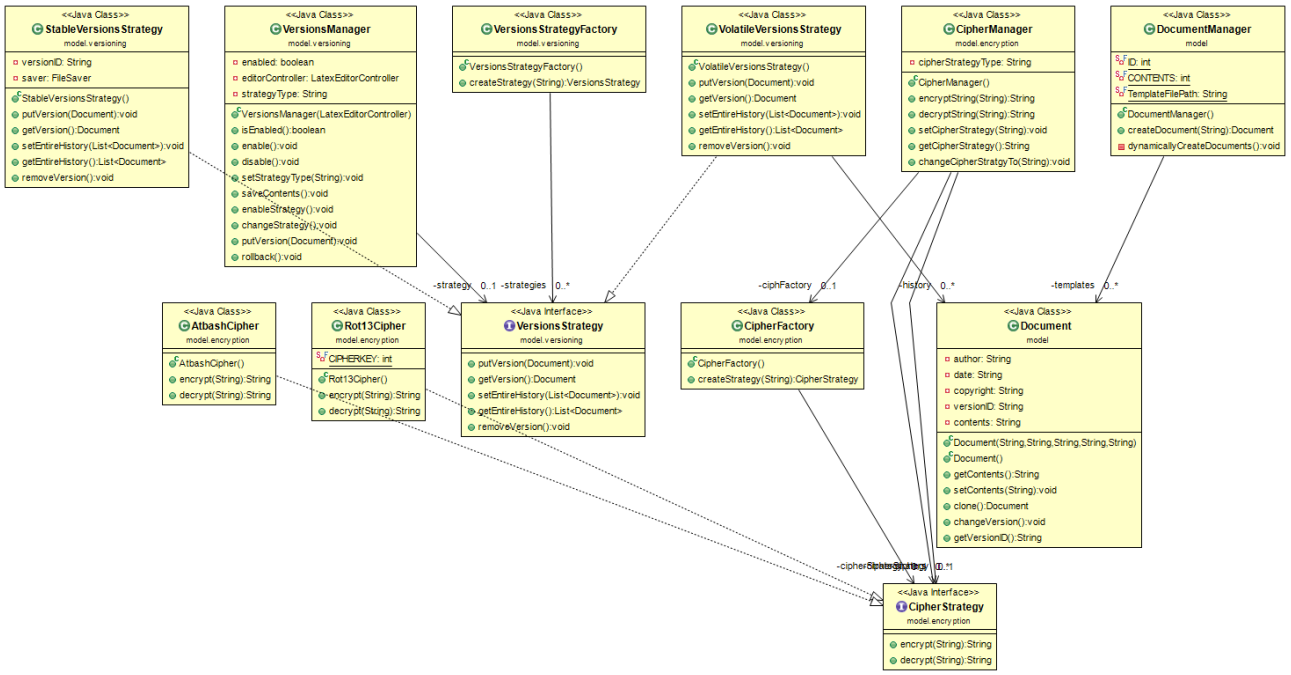
ARCHITECTURE

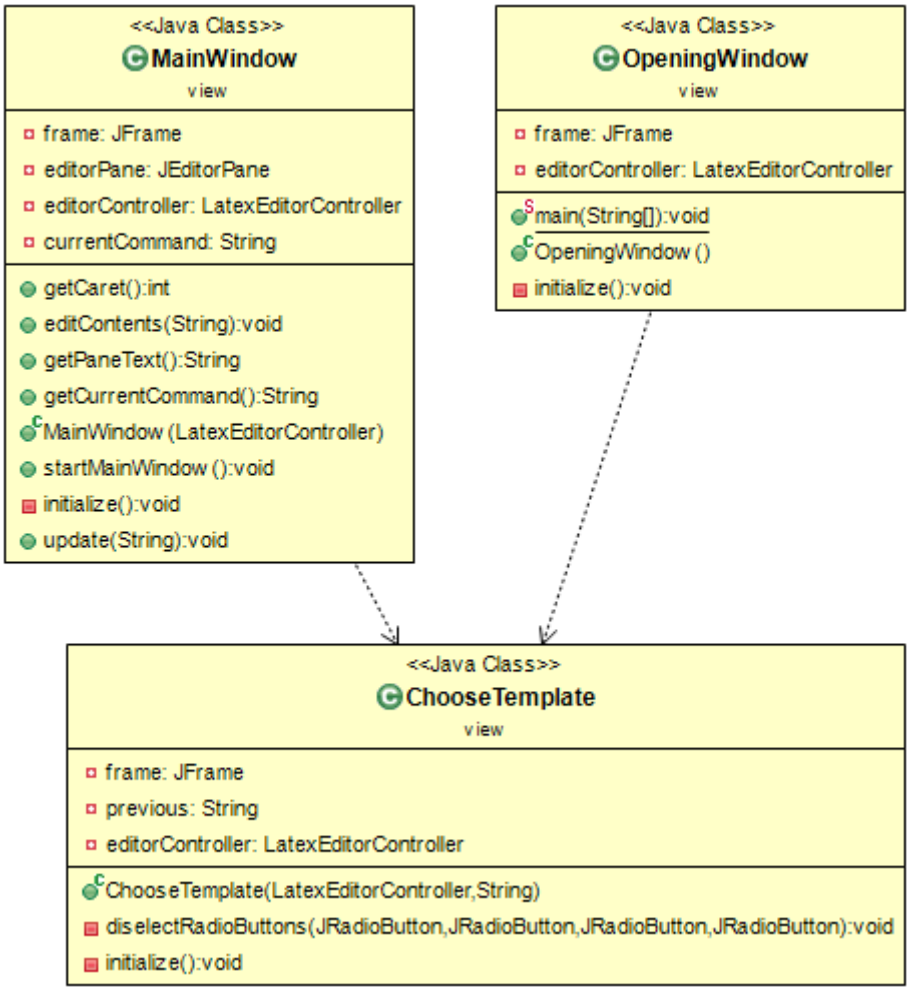


DETAILED DESIGN






<<Java Class>>
StableVersionsStrategy
model.versioning
versionID: String
saver: FileSaver
StableVersionsStrategy()
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
<<Java Class>>
VersionsManager
model.versioning
enabled: boolean
editorController: LatexEditorController
strategyType: String
VersionsManager(LatexEditorController)
isEnabled():boolean
enable():void
disable():void
setStrategyType(String):void
saveContents():void
enableStrategy():void
changeStrategy():void
putVersion(Document):void
rollback():void
<<Java Class>>
VersionsStrategyFactory
model.versioning
VersionsStrategyFactory()
createStrategy(String):VersionsStrategy
<<Java Class>>
VolatileVersionsStrategy
model.versioning
VolatileVersionsStrategy()
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
<<Java Class>>
CipherManager
model.encryption
cipherStrategyType: String
CipherManager()
encryptString(String):String
decryptString(String):String
setCipherStrategy(String):void
getCipherStrategy():String
changeCipherStratgyTo(String):void
<<Java Class>>
DocumentManager
model
ID: int
CONTENTS: int
TemplateFilePath: String
DocumentManager()
createDocument(String):Document
dynamicallyCreateDocuments():void
-strategy 0..1
-strategies 0..*
-ciphFactory 0..1
-history 0..*
-templates 0..*
<<Java Class>>
AtbashCipher
model.encryption
AtbashCipher()
encrypt(String):String
decrypt(String):String
<<Java Class>>
Rot13Cipher
model.encryption
CIPHERKEY: int
Rot13Cipher()
encrypt(String):String
decrypt(String):String
<<Java Interface>>
VersionsStrategy
model.versioning
putVersion(Document):void
getVersion():Document
setEntireHistory(List<Document>):void
getEntireHistory():List<Document>
removeVersion():void
<<Java Class>>
CipherFactory
model.encryption
CipherFactory()
createStrategy(String):CipherStrategy
<<Java Class>>
Document
model
author: String
date: String
copyright: String
versionID: String
contents: String
Document(String,String,String,String,String)
Document()
getContents():String
setContents(String):void
clone():Document
changeVersion():void
getVersionID():String
<<Java Interface>>
CipherStrategy
model.encryption
encrypt(String):String
decrypt(String):String



To address the problem of Duplicate Code in the Document Manager and LatexEditorController classes we implemented the alternative algorithm method, now the document templates and the types of commands are read from properties files from the disk that are in separate folders in the project. This also helps with the extending the project without needing to touch multiple classes, in order to add a new Latex Document template or a new Command.

In the controller.commands package we extracted a SuperCommand abstract class from all classes that implement the Command interface. Now the SuperCommand class contains the fields that all commands required. SuperCommand has protected fields. The Commands can call methods from the fields of SuperCommand.

In addLatexCommand we moved code from the MainWindow class that added the commands in the document. Delegation of responsibilities was wrong so we fixed that.

EditCommand is now responsible for setting the document contents taking input from the main window. Moved the responsibility from the mainWindow and latexEditorView classes.

Moved the save to disk of a .tex file from DocumentClass to the SaveCommand class.

Moved responsibility of loading files from VersionsManager to LoadCommand class.

Regarding the model.versioning subsystem. (renamed the package from strategy):

VersionManager had some methods that were dead code, not called anywhere in the project we removed those methods from the project completely.

VersionManager besides having responsibility for the VersioningSubsystem also served the role of a middle man that was called from most of the classes from the back end to communicate with the front end namely the LatexEditorView class. We started by removing middle man methods and passing reference to the LatexEditorView to every class that needed access to it's methods/fields.

View Package:

Latex EditorView this class contained data and methods that had nothing to do with the GUI, we decided that we should remove this class from the project entirely. This class was the interface between the back end and the front end, so after removing this class most of re responsibilities moved to LatexEditorController except those that had nothing to do with interfacing like load and save functionalities that were moved to their respective command classes.
We set the LatexEditorController to have the currentVersion of the Document and fields with references to the other classes of the project as well as get methods for these fields so commands can use them.

In the mainWindow class we mended the "long method" problem that actually was the addlatexCommand we moved that method to it's respective command class.

Other tweaks to the code were minor and were to ensure that the program compiled and ran after the refactoring.

Added a utilities package to ensure we have no duplicate code.
Two classes handle loading and saving from and to the disk.

Extension

The Encryption package consists of :
A Manager class that handles the switch from an encryption method to another and also provides a simple interface to the main system logic.
A Factory that creates instances of encryption algorithms classes.
An Interface for the different encryption algorithms.
An implementation of the Atbash algorithm and
an implementation of the Rot13 algorithm.

This package follows the Strategy Pattern for simple addition of new ways to encrypt documents and to switch from one to another.

IMPLEMENTATION