LatexEditor

Release Report

OneDayCase
Androutsopoulos Georgios, AM: 2933
Xristoforos Karvelis, AM: 2989

VERSIONS HISTORY

Date	Versio n	Description	Author
18/4/2019	1	Release 1	Both team members

1 Introduction

This document provides information concerning the **first** release of the project.

1.1 Purpose

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 a programming like process as it involves the proper usage of Latex commands which are embedded in the document contents. The objective of this project is to develop a simple Latex editor for inexperienced Latex users. The goal of the 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.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 specifies the acceptance tests that have been employed for this release of the project. Section 3 specifies the main design concepts for this release of the project.

2 Tests

2.1 Tests for User Story **US1**

Test ID	*C_E1*
User Story	*US1*
Test Class	`CreateCommandTest`
Description	*We test the right creation of an Empty Document by creating a CreateCommand object and setting the currentDocumentType to “Empty”. Then we execute the createCommand so we expect the Type of the currentDocument to be “Empty” and its Contents to be empty string.*

Test ID	*C_R1*
User Story	*US1*
Test Class	`CreateCommandTest`
Description	We test the right creation of a Report Document by creating a CreateCommand object and setting the currentDocumentType to “Report”. Then we execute the createCommand so we expect the Type of the currentDocument to be “Report” and its Contents to be the same as the Report template.

Test ID	*C_A1*
User Story	*US1*
Test Class	`CreateCommandTest`
Description	*We test the right creation of an Article Document by creating a CreateCommand object and setting the currentDocumentType to “Article”. Then we execute the createCommand so we expect the Type of the currentDocument to be “Article” and its Contents to be the same as the Article template.*

Test ID	*C_B1*
User Story	*US1*
Test Class	`CreateCommandTest`
Description	*We test the right creation of a Book Document by creating a CreateCommand object and setting the currentDocumentType to “Book”. Then we execute the createCommand so we expect the Type of the currentDocument to be “Book” and its Contents to be the same as the Book template.*

Test ID	*C_L1*
User Story	*US1*
Test Class	`CreateCommandTest`
Description	*We test the right creation of a Letter Document by creating a CreateCommand object and setting the currentDocumentType to “Letter”. Then we execute the createCommand so we expect the Type of the currentDocument to be “Letter” and its Contents to be the same as the Letter template.*

2.2 Tests for User Story **US3**

Test ID	*A_C3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Chapter command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Chapter. Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Chapter command's.*

Test ID	*A_S3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Section command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Section". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Section command's.*

Test ID	*A_SS3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Subsection command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Subsection". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Subsection command's.*

Test ID	*A_SSS3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a* `Subsubsection` *command by creating an AddLatexCommand object, an Empty document and setting the* `currentCommandName` *to "*`Subsubsection`*". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the* `Subsubsection` *command's.*

Test ID	*A_IL3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of an Item List command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Item List". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Item List command's.*

Test ID	*A_EL3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Enumeration List command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Enumeration List". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Enumeration List command's.*

Test ID	*A_T3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Table command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Table". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Table command's.*

Test ID	*A_F3*
User Story	*US3*
Test Class	`AddLatexCommandTest`
Description	*We test the right addition of a Figure command by creating an AddLatexCommand object, an Empty Document and setting the* `currentCommandName` *to "Figure". Then we execute the AddLatexCommand so we expect the contents of the currentDocument to be the same as the Figure command's.*

2.3 Tests for User Story **US8**

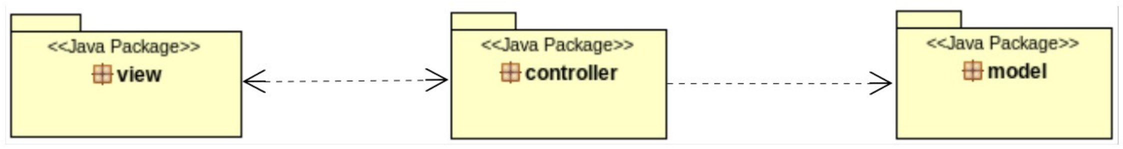
Test ID	*SV8*
User Story	*US8*
Test Class	`SaveCommandTest`
Description	*We test the right save of a Document to disk by creating a SaveCommand object, an Empty document and setting the the contents of it to a specific string. Then we execute the SaveCommand so we expect the contents of the saved file to be the same as the contents of the currentDocument.*

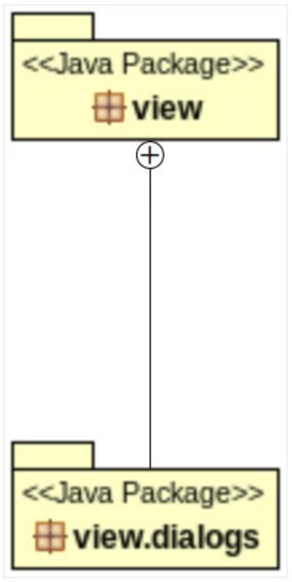
2.4 Tests for User Story **US9**

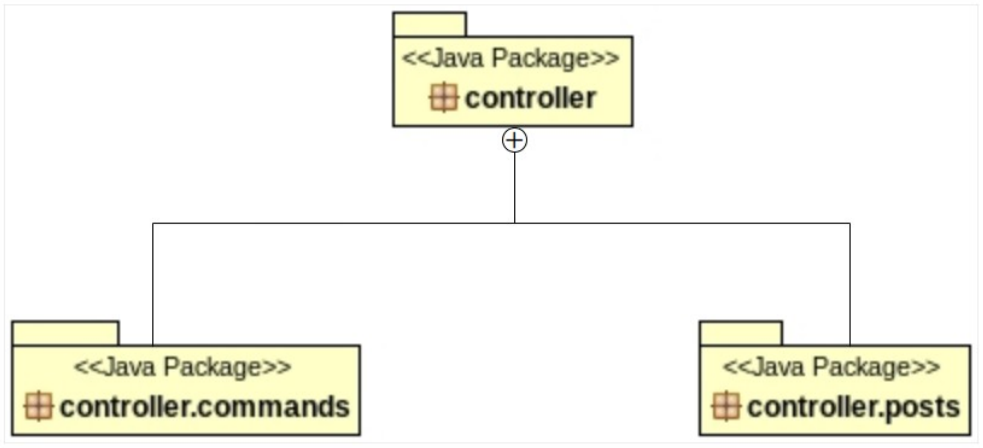
Test ID	*OPN9*
User Story	*US9*
Test Class	`OpenCommandTest`
Description	*We test the right Load of a file from disk by writing a LateX file with a specific string and creating an OpenCommand object. Then we execute the OpenCommand with a new Document object so we except the contents of the Document to be the same as the specific string.*

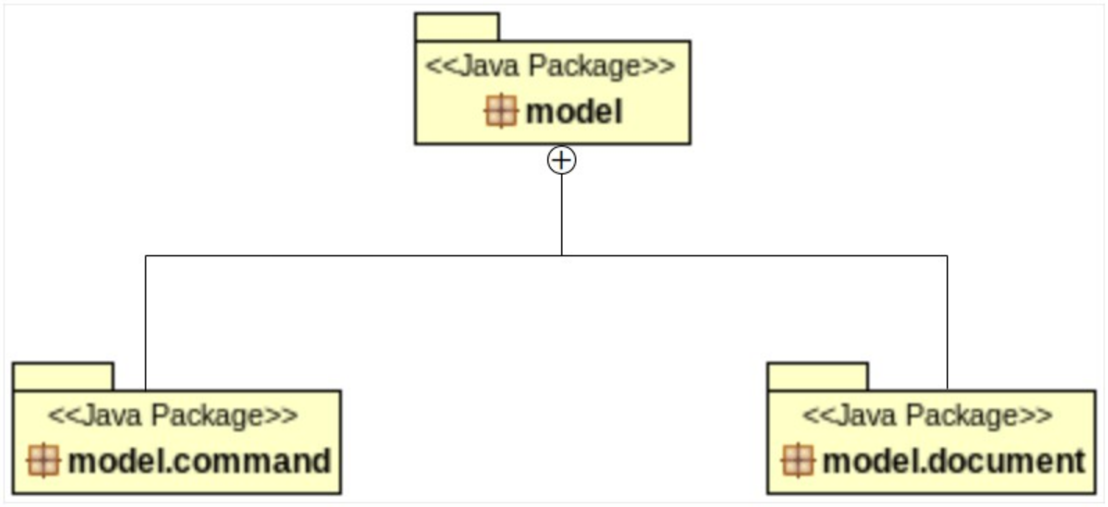
3 Design

3.1 Architecture

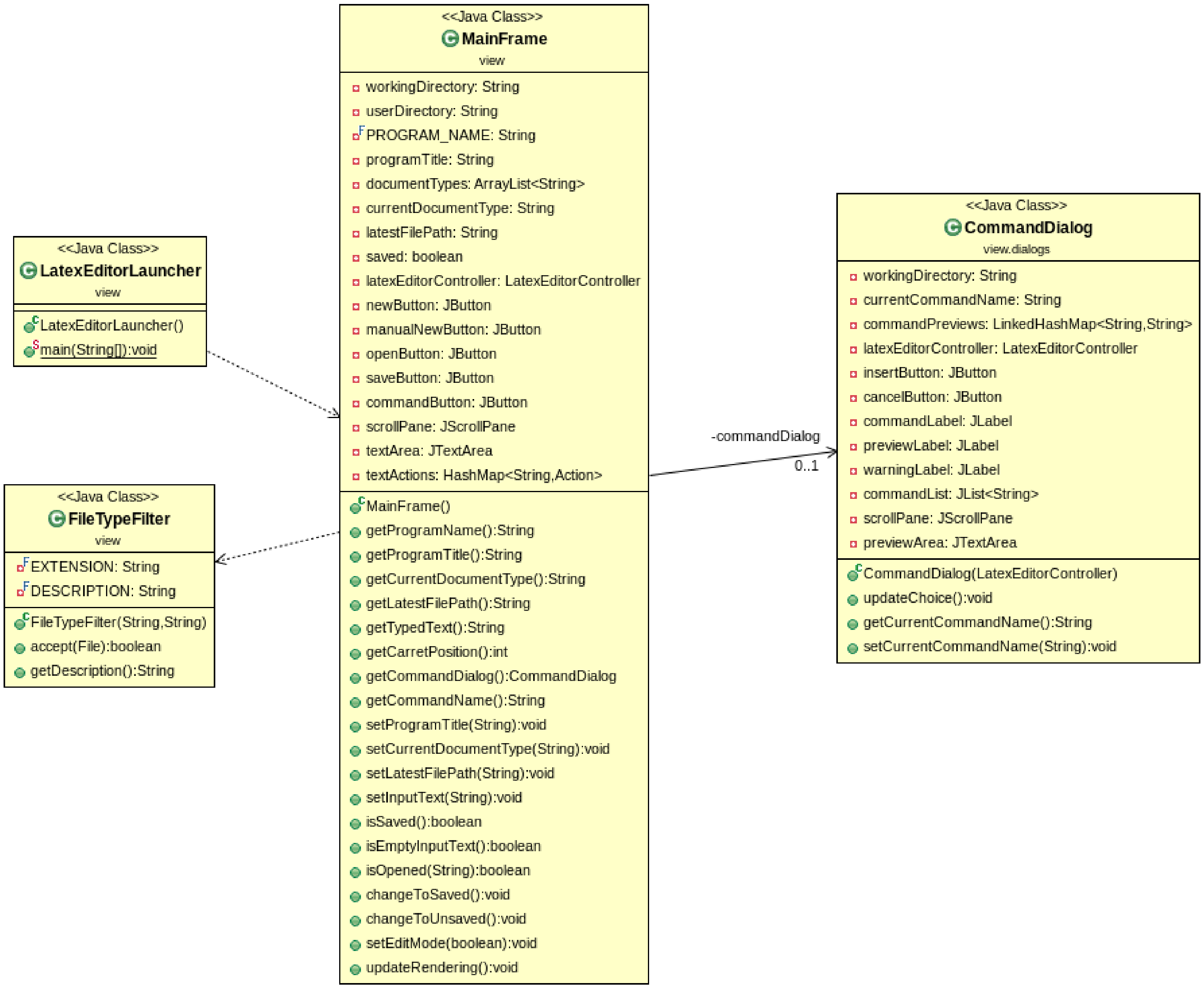




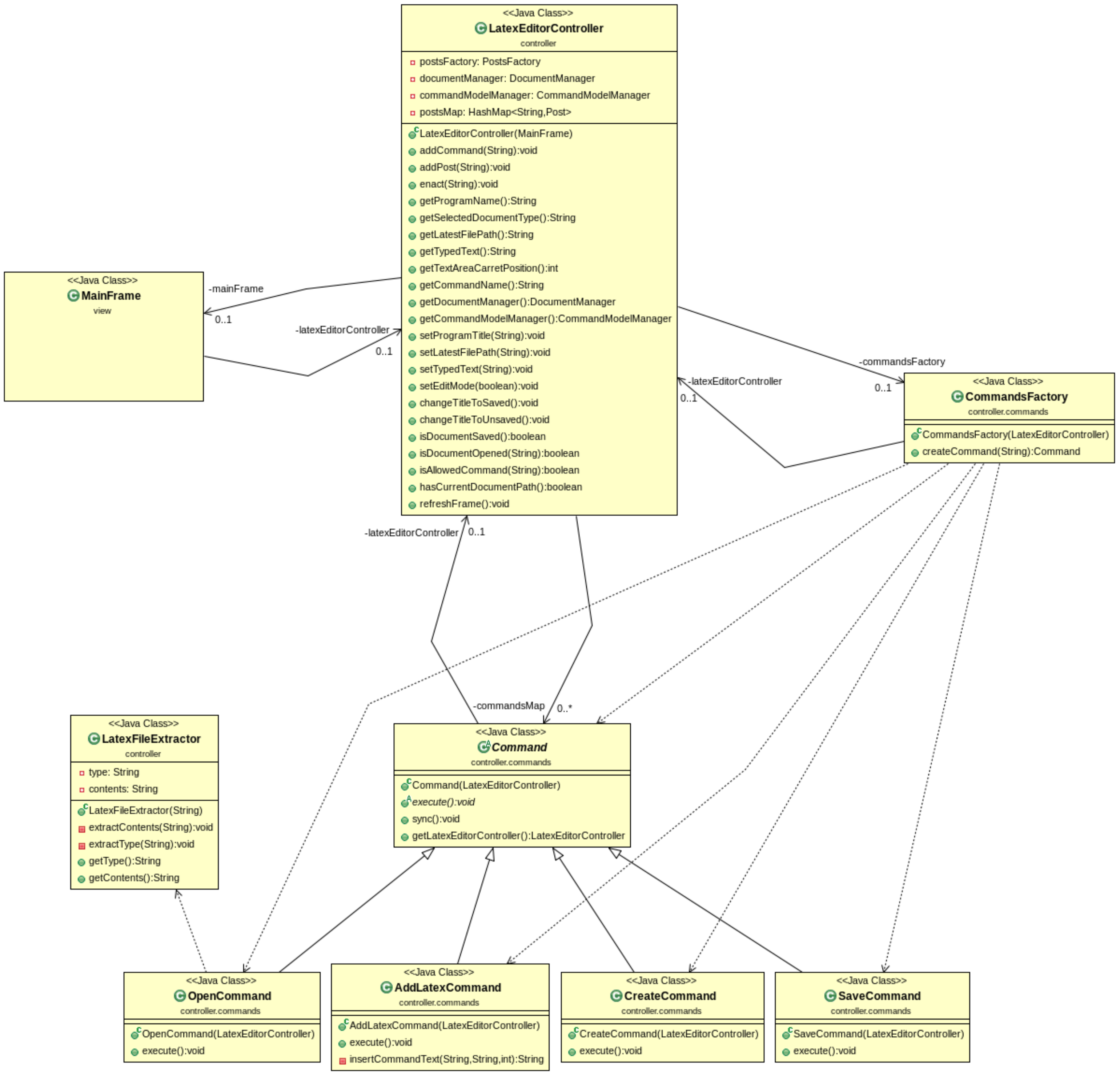

<<Java Package>>
controller
<<Java Package>>
controller.commands
<<Java Package>>
controller.posts


<<Java Package>>
model
<<Java Package>>
model.command
<<Java Package>>
model.document

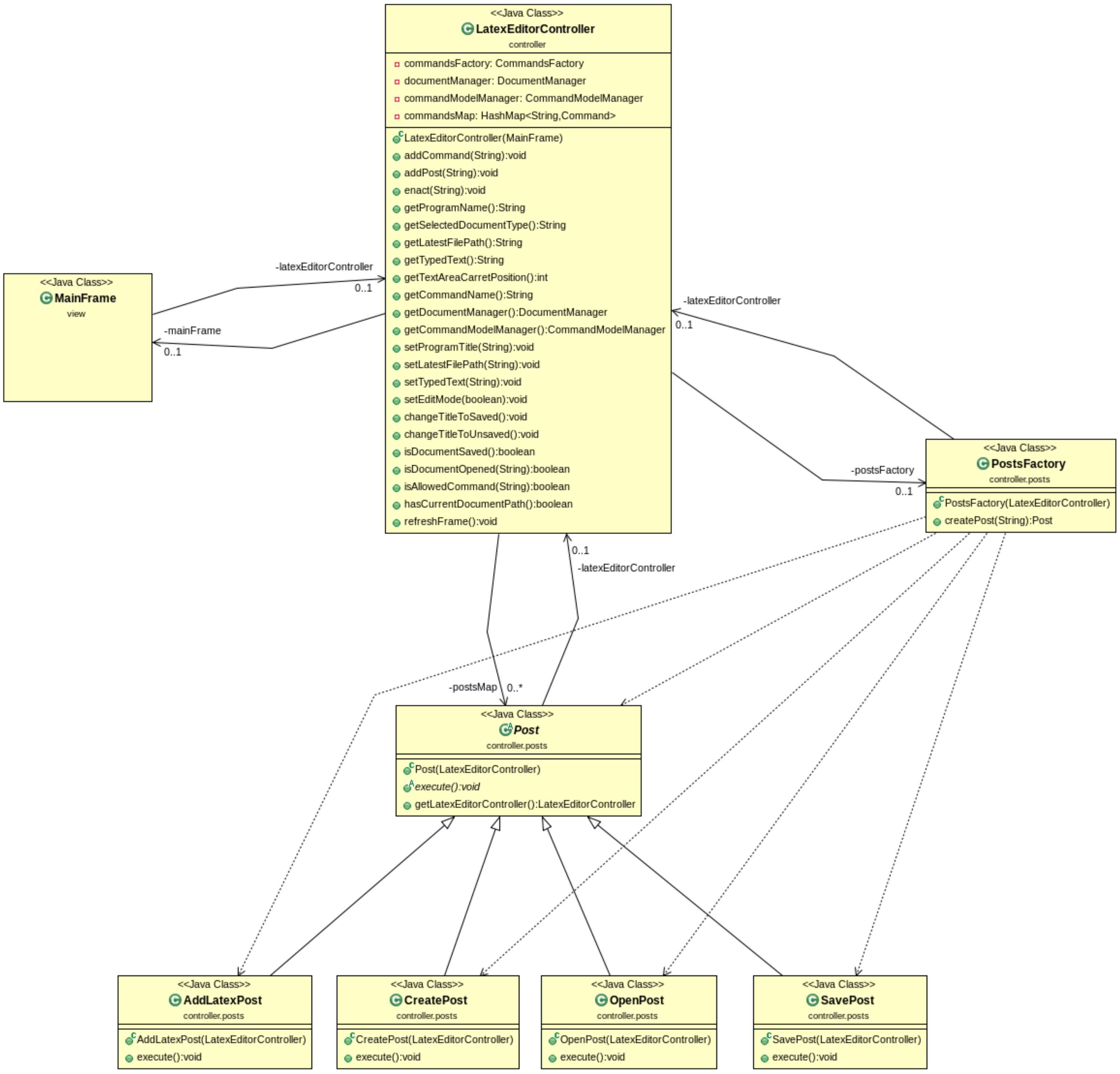
3.2 Design



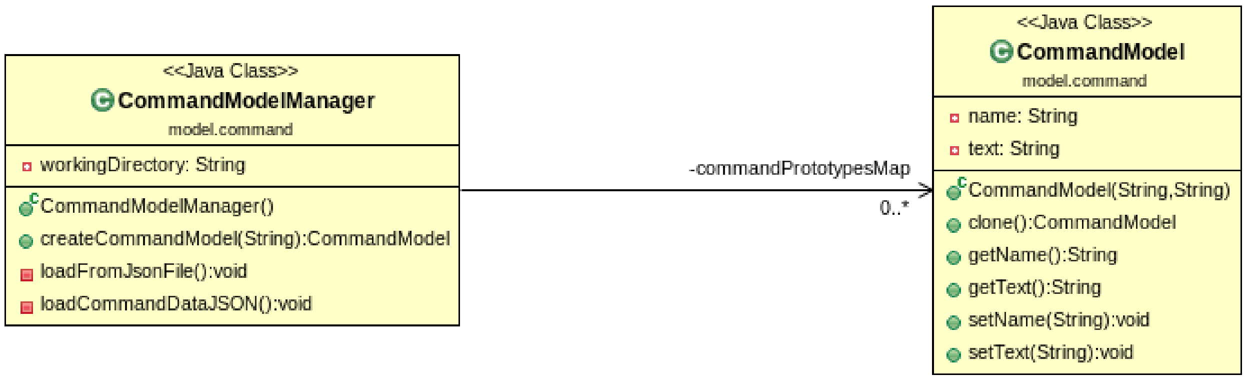
<<Java Class>>
LatexEditorLauncher
view
LatexEditorLauncher()
main(String[]):void
<<Java Class>>
FileTypeFilter
view
EXTENSION: String
DESCRIPTION: String
FileTypeFilter(String,String)
accept(File):boolean
getDescription():String
<<Java Class>>
MainFrame
view
workingDirectory: String
userDirectory: String
PROGRAM_NAME: String
programTitle: String
documentTypes: ArrayList<String>
currentDocumentType: String
latestFilePath: String
saved: boolean
latexEditorController: LatexEditorController
newButton: JButton
manualNewButton: JButton
openButton: JButton
saveButton: JButton
commandButton: JButton
scrollPane: JScrollPane
textArea: JTextArea
textActions: HashMap<String,Action>
MainFrame()
getProgramName():String
getProgramTitle():String
getCurrentDocumentType():String
getLatestFilePath():String
getTypedText():String
getCarretPosition():int
getCommandDialog():CommandDialog
getCommandName():String
setProgramTitle(String):void
setCurrentDocumentType(String):void
setLatestFilePath(String):void
setInputText(String):void
isSaved():boolean
isEmptyInputText():boolean
isOpened(String):boolean
changeToSaved():void
changeToUnsaved():void
setEditMode(boolean):void
updateRendering():void
-commandDialog
0..1
<<Java Class>>
CommandDialog
view.dialogs
workingDirectory: String
currentCommandName: String
commandPreviews: LinkedHashMap<String,String>
latexEditorController: LatexEditorController
insertButton: JButton
cancelButton: JButton
commandLabel: JLabel
previewLabel: JLabel
warningLabel: JLabel
commandList: JList<String>
scrollPane: JScrollPane
previewArea: JTextArea
CommandDialog(LatexEditorController)
updateChoice():void
getCurrentCommandName():String
setCurrentCommandName(String):void

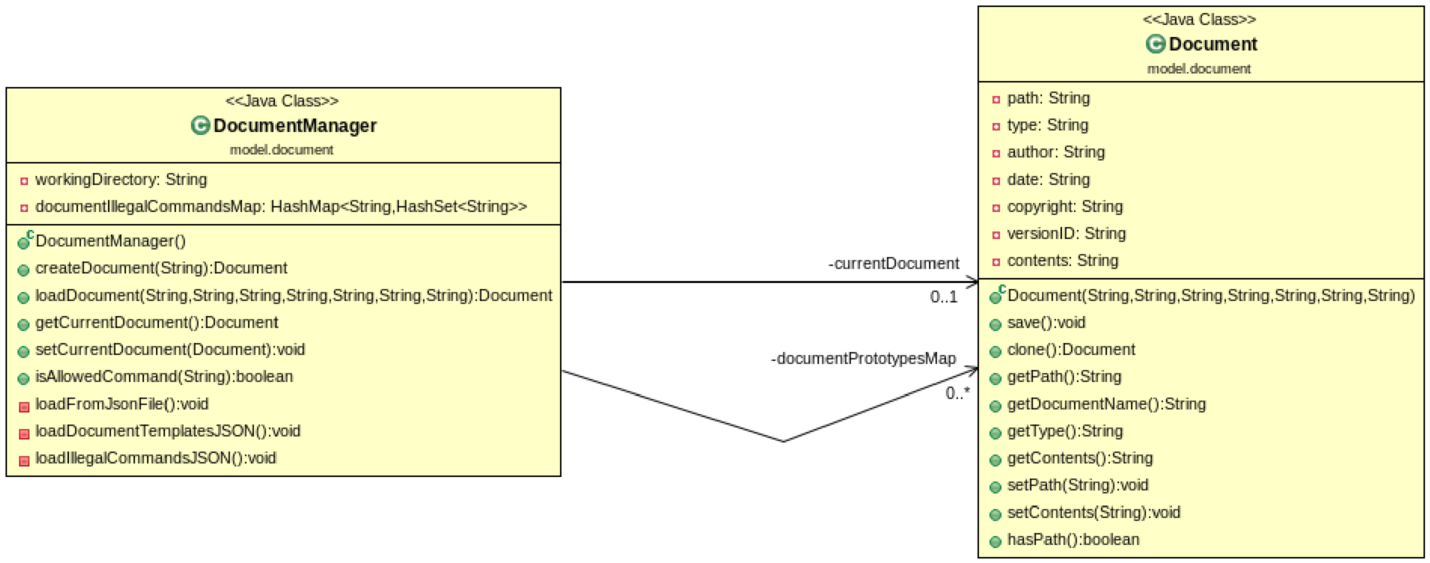


<<Java Class>>
LatexEditorController
controller
postsFactory: PostsFactory
documentManager: DocumentManager
commandModelManager: CommandModelManager
postsMap: HashMap<String,Post>
LatexEditorController(MainFrame)
addCommand(String):void
addPost(String):void
enact(String):void
getProgramName():String
getSelectedDocumentType():String
getLatestFilePath():String
getTypedText():String
getTextAreaCarretPosition():int
getCommandName():String
getDocumentManager():DocumentManager
getCommandModelManager():CommandModelManager
setProgramTitle(String):void
setLatestFilePath(String):void
setTypedText(String):void
setEditMode(boolean):void
changeTitleToSaved():void
changeTitleToUnsaved():void
isDocumentSaved():boolean
isDocumentOpened(String):boolean
isAllowedCommand(String):boolean
hasCurrentDocumentPath():boolean
refreshFrame():void
<<Java Class>>
MainFrame
view
-mainFrame
0..1
-latexEditorController
0..1
-commandsFactory
0..1
-latexEditorController
0..1
<<Java Class>>
CommandsFactory
controller.commands
CommandsFactory(LatexEditorController)
createCommand(String):Command
-latexEditorController
0..1
-commandsMap
0..*
<<Java Class>>
LatexFileExtractor
controller
type: String
contents: String
LatexFileExtractor(String)
extractContents(String):void
extractType(String):void
getType():String
getContents():String
<<Java Class>>
Command
controller.commands
Command(LatexEditorController)
execute():void
sync():void
getLatexEditorController():LatexEditorController
<<Java Class>>
OpenCommand
controller.commands
OpenCommand(LatexEditorController)
execute():void
<<Java Class>>
AddLatexCommand
controller.commands
AddLatexCommand(LatexEditorController)
execute():void
insertCommandText(String,String,int):String
<<Java Class>>
CreateCommand
controller.commands
CreateCommand(LatexEditorController)
execute():void
<<Java Class>>
SaveCommand
controller.commands
SaveCommand(LatexEditorController)
execute():void



<<Java Class>>
LatexEditorController
controller
commandsFactory: CommandsFactory
documentManager: DocumentManager
commandModelManager: CommandModelManager
commandsMap: HashMap<String,Command>
LatexEditorController(MainFrame)
addCommand(String):void
addPost(String):void
enact(String):void
getProgramName():String
getSelectedDocumentType():String
getLatestFilePath():String
getTypedText():String
getTextAreaCarretPosition():int
getCommandName():String
getDocumentManager():DocumentManager
getCommandModelManager():CommandModelManager
setProgramTitle(String):void
setLatestFilePath(String):void
setTypedText(String):void
setEditMode(boolean):void
changeTitleToSaved():void
changeTitleToUnsaved():void
isDocumentSaved():boolean
isDocumentOpened(String):boolean
isAllowedCommand(String):boolean
hasCurrentDocumentPath():boolean
refreshFrame():void
<<Java Class>>
MainFrame
view
-latexEditorController
0..1
-mainFrame
0..1
-latexEditorController
0..1
-postsFactory
0..1
<<Java Class>>
PostsFactory
controller.posts
PostsFactory(LatexEditorController)
createPost(String):Post
0..1
-latexEditorController
-postsMap
0..*
<<Java Class>>
Post
controller.posts
Post(LatexEditorController)
execute():void
getLatexEditorController():LatexEditorController
<<Java Class>>
AddLatexPost
controller.posts
AddLatexPost(LatexEditorController)
execute():void
<<Java Class>>
CreatePost
controller.posts
CreatePost(LatexEditorController)
execute():void
<<Java Class>>
OpenPost
controller.posts
OpenPost(LatexEditorController)
execute():void
<<Java Class>>
SavePost
controller.posts
SavePost(LatexEditorController)
execute():void


<<Java Class>>
CommandModelManager
model.command
workingDirectory: String
CommandModelManager()
createCommandModel(String):CommandModel
loadFromJsonFile():void
loadCommandDataJSON():void
-commandPrototypesMap
0..*
<<Java Class>>
CommandModel
model.command
name: String
text: String
CommandModel(String,String)
clone():CommandModel
getName():String
getText():String
setName(String):void
setText(String):void


<<Java Class>>
DocumentManager
model.document
workingDirectory: String
documentIllegalCommandsMap: HashMap<String,HashSet<String>>
DocumentManager()
createDocument(String):Document
loadDocument(String,String,String,String,String,String,String):Document
getCurrentDocument():Document
setCurrentDocument(Document):void
isAllowedCommand(String):boolean
loadFromJsonFile():void
loadDocumentTemplatesJSON():void
loadIllegalCommandsJSON():void
-currentDocument
0..1
-documentPrototypesMap
0..*
<<Java Class>>
Document
model.document
path: String
type: String
author: String
date: String
copyright: String
versionID: String
contents: String
Document(String,String,String,String,String,String,String)
save():void
clone():Document
getPath():String
getDocumentName():String
getType():String
getContents():String
setPath(String):void
setContents(String):void
hasPath():boolean

view

Class Name: MainFrame	
Responsibilities:	**Collaborations:**
▪ This class is responsible for creating the main window of the editor and all the elements that are inside it(e.g. buttons, panels,...) . ▪ This class creates the listeners of the interactive elements of the main window. ▪ This class is also responsible for creating the latexEditorController and initializing both its maps.	▪ `LatexEditorController` ▪ `CommandDialog` ▪ `FileTypeFilter`

Class Name: LatexEditorLauncher	
Responsibilities:	**Collaborations:**
▪ This is the main class of the view package that creates the main frame.	▪ `MainFrame`

Class Name: FileTypeFilter	
Responsibilities:	**Collaborations:**
▪ This class is responsible to create a certain file type filter for the openFileChooser so the user can't load other files except from .tex ones.	▪ None

view.dialogs

Class Name: `CommandDialog`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to create a pop-up window that implements the AddLatexCommand option. ▪ This class is also responsible for creating the listeners of the interactive elements of the pop-up window.	▪ None

controller

Class Name: `LatexEditorController`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to implement the communication between the view package and the model package.	▪ `MainFrame` ▪ `CommandsFactory` ▪ `PostsFactory` ▪ `DocumentManager` ▪ `CommandModelManager` ▪ `Command` ▪ `Post`

Class Name: `LatexFileExtractor`	
Responsibilities:	**Collaborations:**
▪ This class performs the reading of a .tex file and the extraction of the Document type and its contents from it.	▪ `None`

controller.commands

Class Name: `Command`	
Responsibilities: ▪ This class is an abstract class and serves as the main body of of every Command class. ▪ This class is responsible to give to all the classes that inherit its properties the ability to sync any changes the user does from the current open window with the current Document before doing any further action. ▪ This class also provides to the classes that inherit its properties with the LatexEditorController object.	**Collaborations:** ▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `CommandsFactory`	
Responsibilities: ▪ This class is a parameterized factory that is responsible to create the right Command object according to a parameter.	**Collaborations:** ▪ `LatexEditorController` ▪ `Command` ▪ `CreateCommand` ▪ `OpenCommand` ▪ `SaveCommand` ▪ `AddLatexCommand`

Class Name: `CreateCommand`	
Responsibilities:	**Collaborations:**
▪ This class extends the abstract Command class and is responsible to create a Document of a specific type and set the DocumentManager's currentDocument to it.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `OpenCommand`	
Responsibilities:	**Collaborations:**
▪ This class extends the abstract Command class and is responsible to create a new Document that will receive the data read from a .tex file and then use it to update the current Document through the DocumentManager.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document` ▪ `LatexFileExtractor`

Class Name: `SaveCommand`	
Responsibilities:	**Collaborations:**
▪ This class extends the abstract Command class and is responsible to sync any changes made from the user and then call the save method on the current Document to save any changes made to file.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `AddLatexCommand`	
Responsibilities:	**Collaborations:**
▪ This class extends the abstract Command class and is responsible to get the current text area and insert the right command text in it according to the command selection of the user. Then update the contents of the current Document.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `CommandModelManager` ▪ `Document` ▪ `CommandModel`

controller.posts

Class Name: `Post`	
Responsibilities:	**Collaborations:**
▪ This class is an abstract class and serves as the main body of of every Post class. ▪ This class provides to the classes that inherit its properties with the LatexEditorController object.	▪ `LatexEditorController`

Class Name: `PostsFactory`	
Responsibilities:	**Collaborations:**
▪ This class is a parameterized factory that is responsible to create the right Post object according to a parameter	▪ `LatexEditorController` ▪ `CreatePost` ▪ `OpenPost` ▪ `SavePost` ▪ `AddLatexPost`

Class Name: `CreatePost`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to get the elements of the current Document, update the view window and change the title of the editor to indicate that it is unsaved.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `OpenPost`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to get the elements of the current Document and update the view window ▪ This class is also responsible change the title of the editor to indicate the name of the file	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `SavePost`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to change the title of the editor to indicate that the document is saved.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

Class Name: `AddLatexPost`	
Responsibilities:	**Collaborations:**
▪ This class is responsible to get the elements of the current Document, update the view window and change the title of the editor to indicate that the document is unsaved.	▪ `LatexEditorController` ▪ `DocumentManager` ▪ `Document`

model.command

Class Name: `CommandModel`	
Responsibilities:	**Collaborations:**
▪ This class implements the Prototype pattern and is responsible to return clones of CommandModel objects.	▪ `None`

Class Name: `CommandModelManager`	
Responsibilities:	**Collaborations:**
▪ This class implements the manager of the Prorotype pattern and is responsible to create a Map. The Map's key-set contains names of the available Latex Commands and its values are `CommandModel objects containing the` corresponding text for each command.	▪ `CommandModel`

model.document

Class Name: `Document`	
Responsibilities:	**Collaborations:**
▪ This class implements the Prototype pattern and is responsible to return clones of Document objects.	▪ `None`

Class Name: `DocumentManager`	
Responsibilities:	**Collaborations:**
▪ This class implements the manager of the Prototype pattern and is responsible to create a Map. The Map’s key-set contains names of the available Latex Documents and its values are Document objects containing the corresponding template for each document	▪ `Document`