` AdvancedText2SpeechEditor

Sprint Report

Prodigies

Μέλη Ομάδας:

- Ηλιάδης Ευάγγελος ,3117
- Αργυροκαστρίτη Δημητρία, 4029
- Καπαρινού Χρυσή Μαρία, 4069

VERSIONS HISTORY

Date	Version	Description	Author
25/3/2021 – 10/4/2021	<1.0>	MyFrame class, Commands Pattern ,Document Class	Ηλιάδης Ευάγγελος Αργυροκαστρίτη Δημητρία Καπαρινού Χρυσή Μαρία
11/4/2021 – 29/4/2021	<2.0>	Strategy Pattern	Ηλιάδης Ευάγγελος Αργυροκαστρίτη Δημητρία Καπαρινού Χρυσή Μαρία
4/5/2021 – 9/5/2021	<3.0>	Façade Pattern	Ηλιάδης Ευάγγελος Αργυροκαστρίτη Δημητρία Καπαρινού Χρυσή Μαρία
10/5/2021 – 20/5/2021	<4.0>	Decorator Pattern	Ηλιάδης Ευάγγελος Αργυροκαστρίτη Δημητρία Καπαρινού Χρυσή Μαρία
20/5/2021 – 28/5/2021	<5.0>	Διορθώσεις ,report , demo	Ηλιάδης Ευάγγελος Αργυροκαστρίτη Δημητρία Καπαρινού Χρυσή Μαρία

1 Introduction

The objective of this project is to develop an application that allows to transform documents to audio, so people who have reading problems or visually impaired people can have a useful tool .More specifically , it allows to open, edit and transform to audio different file formats , like Microsoft Word(.docx) , Excel(.xlsx) and Text Editor(.txt) files/documents. The application ,also support the decoding and encoding of those file formats (Rot13, AtBash) .

1.1 Purpose

The purpose of this project is to develop an application that convert different formats of files to speech.

1.2 Document Structure

The rest of this document is structured as follows. Section 2 describes out Scrum team and specifies the this Sprint's backlog. Section 3 specifies the main design concepts for this release of the project.

2 Scrum team and Sprint Backlog

2.1 Scrum team

Product Owner	Apostolos Zarras
Scrum Master	-
Development Team	Ευάγγελος Ηλιάδης, Δημητρία Αργυροκαστρίτη , Χρυσή Μαρία Καπαρινού

2.2 Sprints

Sprint No	Begin Date	End Date	Number of weeks	User stories
1	25/3/2021	10/4/2021	**2**	**1,2,3,4,5,6**
2	4/5/2021	5/5/2021	**3 days**	**7,8**
3	6/5/2021	9/5/2021	**4 days**	**9,10,11**

3 Use Cases

3.1 Use Case 1:Open file

Use case ID	UC1
Actors	User
Pre conditions	The application should be up and running and the file exists.
Main flow of events	1. The use case starts when the user chooses the “Open” command from the GUI: 2. The “open file” dialog appears. 3. The user specifies a file name. 4. The user confirms the opening of the selected file. 5. The “open file” dialog disappears.
Post conditions	The file contents are available to the user for further editing.

3.2 Use Case 2:Open file by typing name

Use case ID	UC2
Actors	User
Related use	Specialization of: Open file

cases	
Pre conditions	The application should be up and running and the file exists.
Main flow of events	1. The use case starts when the user chooses the “Open” command from the GUI: 2. The “open file” dialog appears. 3. The user specifies a file name: 1. The user selects the text field. 2. The user types the file name. 4. The user confirms the opening of the selected file. 5. The “open file” dialog disappears.
Alternative flow	1. The alternative flow starts after step 4: 2. The application indicates that the file specified by the user does not exist. 3. The user enters a valid file name. 4. The use case continues from step 4 of the main flow.
Post conditions	The file contents are available to the user for further editing.

3.3 Use Case 3:Open file by browsing

Use case ID	UC3
Actors	User
Related use cases	Specialization of: Open file
Pre conditions	The application should be up and running and the file exists.
Main flow of events	1. The use case starts when the user chooses the “Open” command from the GUI: 2. The “open file” dialog appears. 3. The user specifies a file name: 1. While the desired file for opening is not displayed among the files of the list 1. The user selects another directory from the “files” list.

	2. The application displays the directory contents in the “files” list. 2. The user selects the desired file. 4. The user confirms the opening of the selected file. 5. The “open file” dialog disappears.
Post conditions	The file contents are available to the user for further editing.

3.4 Use Case 4: File editing

Use case ID	UC4
Actors	User
Pre conditions	The application should be up and running and the file exists.
Main flow of events	1. The use case starts when the selected file has been opened: 2. A file editor with the file loaded appears. 3. The user edits the file contents.
Post conditions	A new version of the file is ready for saving.

3.5 Use Case 5: File saving

Use case ID	UC5
Actors	User
Pre conditions	The application should be up and running and the file exists.
Main flow of events	1. The use case starts when then file contents have been edited and the user chooses the “Save” command from the GUI: 2. The user “saves “the file.
Post conditions	The file updates are saved on the disk.

3.6 Use Case 6: Text-to-Speech Transformation

Use case ID	UC6
Actors	User
Related use cases	Includes: Text-to-Speech from line-to line
Pre conditions	The application should be up and running. The selected file should contain some form of text.
Main flow of events	1. The use case starts when the selected file has been opened and the user chooses the “Speech” from the GUI: 1. The “Speech” drop-down window appears. 1. The user specifies the desired transformation option: 1. Play Whole. 2. Play Line. 2.The drop-down window disappears. 2. The user listens to the audio product of the transformation.
Post conditions	The audio product of the transformation is ready to be listened to by the user.

3.7 Use Case 7: Text-to-Speech Transformation from line- to line

Use case ID	UC7
Actors	User
Related use cases	Specialization of: Text-to-Speech Transformation
Pre conditions	The application should be up and running. The selected file should contain some form of text.

Main flow of events	1. The use case starts at step 1.1 of UC6. 2. A pop-up window appears. 3. The user types the number corresponding to the desired line. 4. The user listens to the audio
Post conditions	The audio product of the part specifically selected is ready to be listened to by the user.

3.8 Use Case 8:Audio parameter tuning

Use case ID	UC8
Actors	User
Pre conditions	The application should be up and running and the audio product of the transformation should exist.
Main flow of events	1. The use cases starts when the audio product is finished and the user chooses the “Sound” from the GUI: 2. The tuning dialog appears. 3. The user specifies the audio parameters to his needs. 1. The user types the desired volume in the “Volume” pop-up window. 2. The user types the desired volume in the “Speech Rate” pop-up window. 3. The user types the desired volume in the “Pitch” pop-up window. 4. The tuning confirms the parameter changes. 5. The tuning dialog disappears.
Alternative flow	-
Post conditions	The file contents are available to the user for further editing.

3.9 Use Case 9: Recording operation

Use case ID	UC9

Actors	User
Related use cases	Includes: Replay Recording, Stop recording
Pre conditions	The application should be up and running. The selected file should contain some form of text.
Main flow of events	1. The use cases starts when the user selects the “Recording” command from the GUI: 2. A drop-down window appears 3. The user selects “Start Recording” 5. The application starts recording the actions/commands. 6. The recording stops when the “Stop Recording” is selected from the drop-down window . 7. The “Replay” sequence starts with the pressing of the “Replay” button from the drop-down window.
Post conditions	The “text-to speech transformation” sequence is ready to be used multiple times.

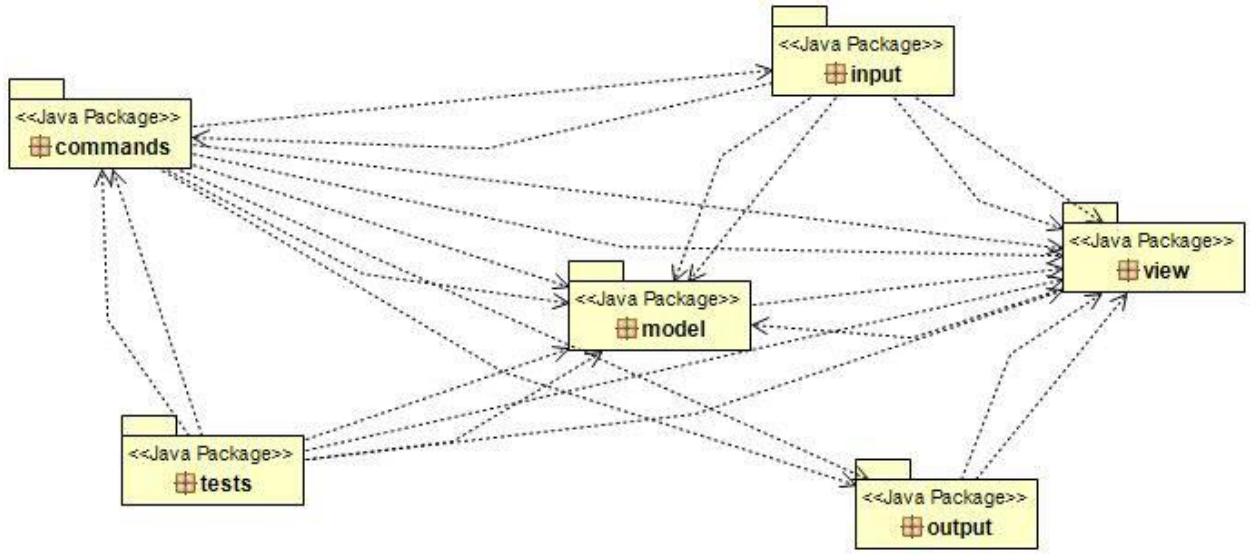
3.10 Use Case 11: Recording Replay

Use case ID	UC11
Actors	User
Related use cases	Included by: Recording operation
Pre conditions	The application should be up and running. The recording operation in progress and available for re-execution.
Main flow of events	1. This use case starts after step 7 of UC10: 2. The recorded actions/commands are being “replayed”.

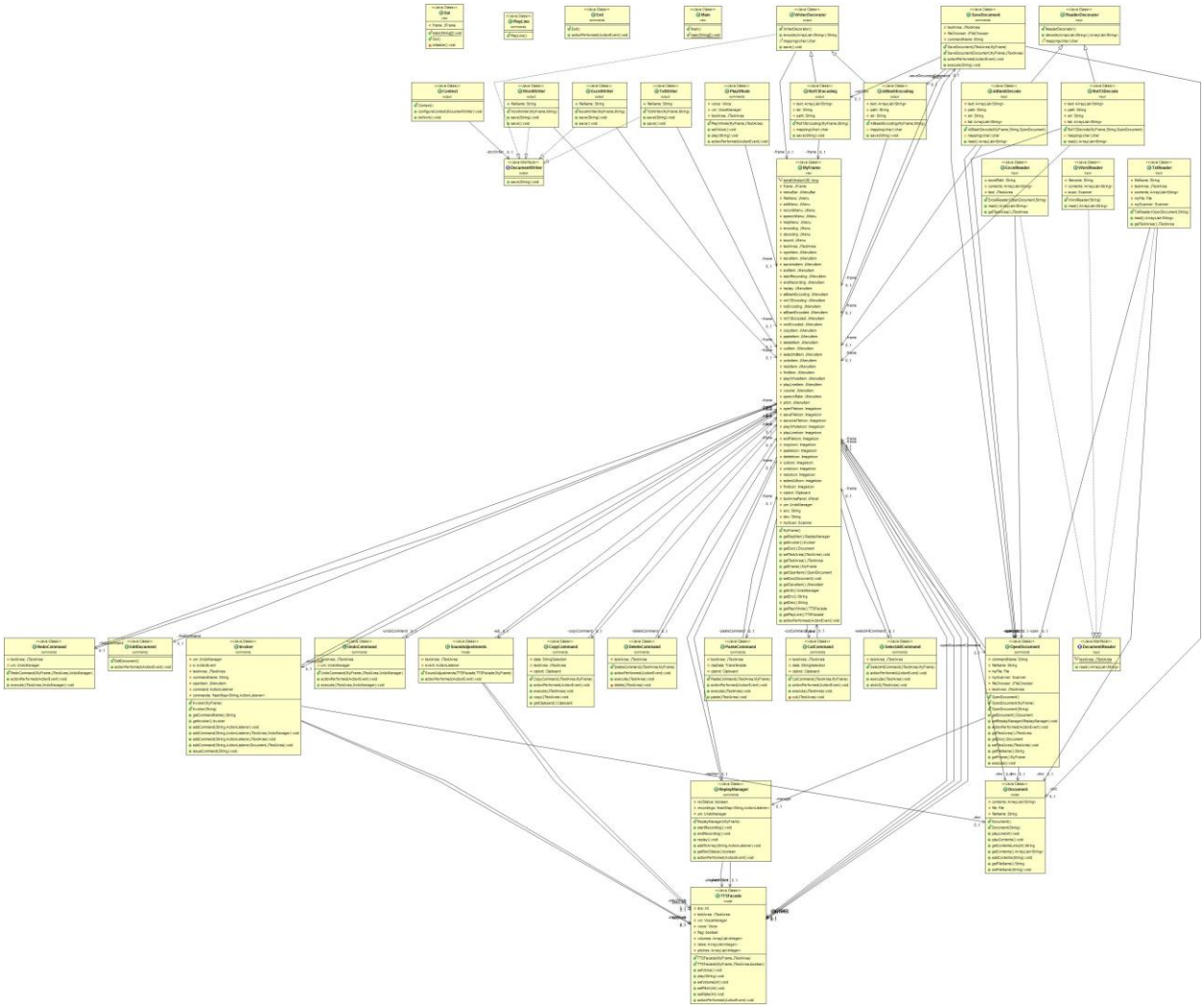
4 Design

4.1 Architecture

Package Diagram:

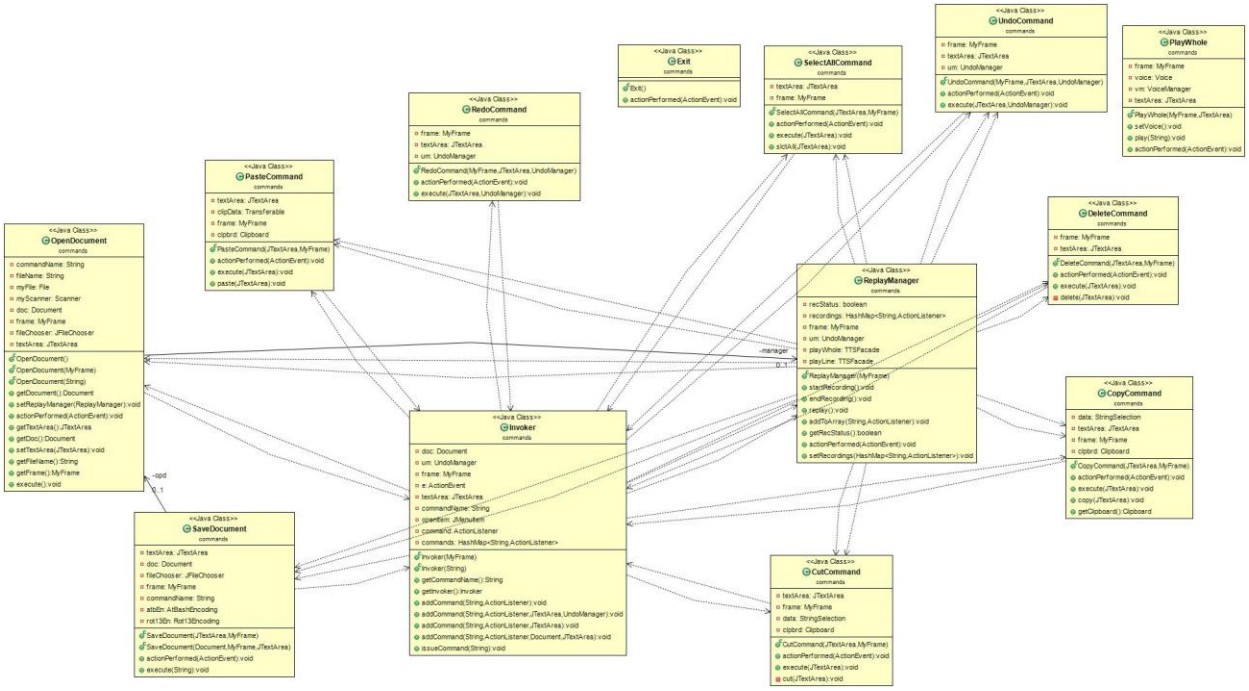


All Classes Diagram:

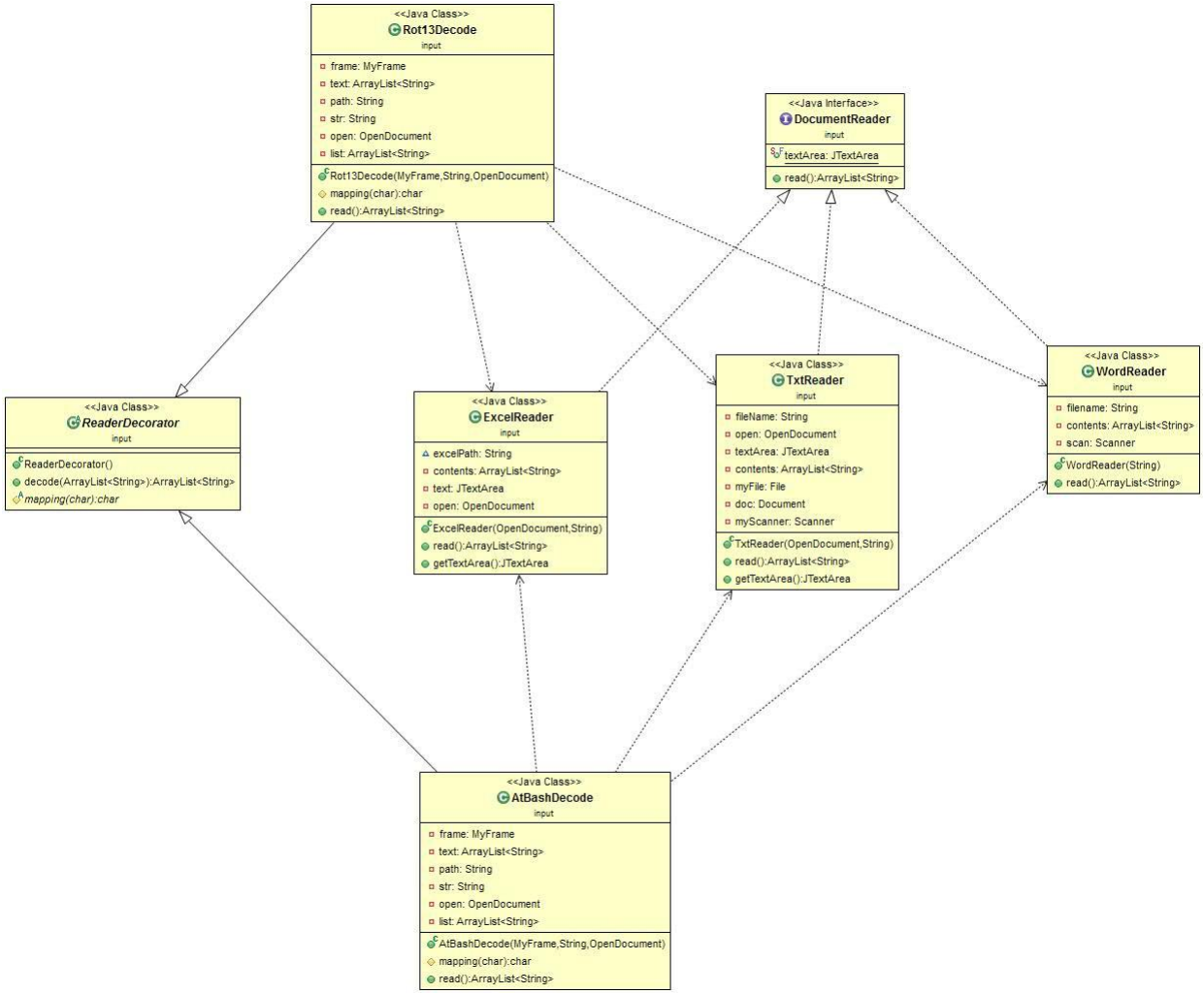


4.2 Design

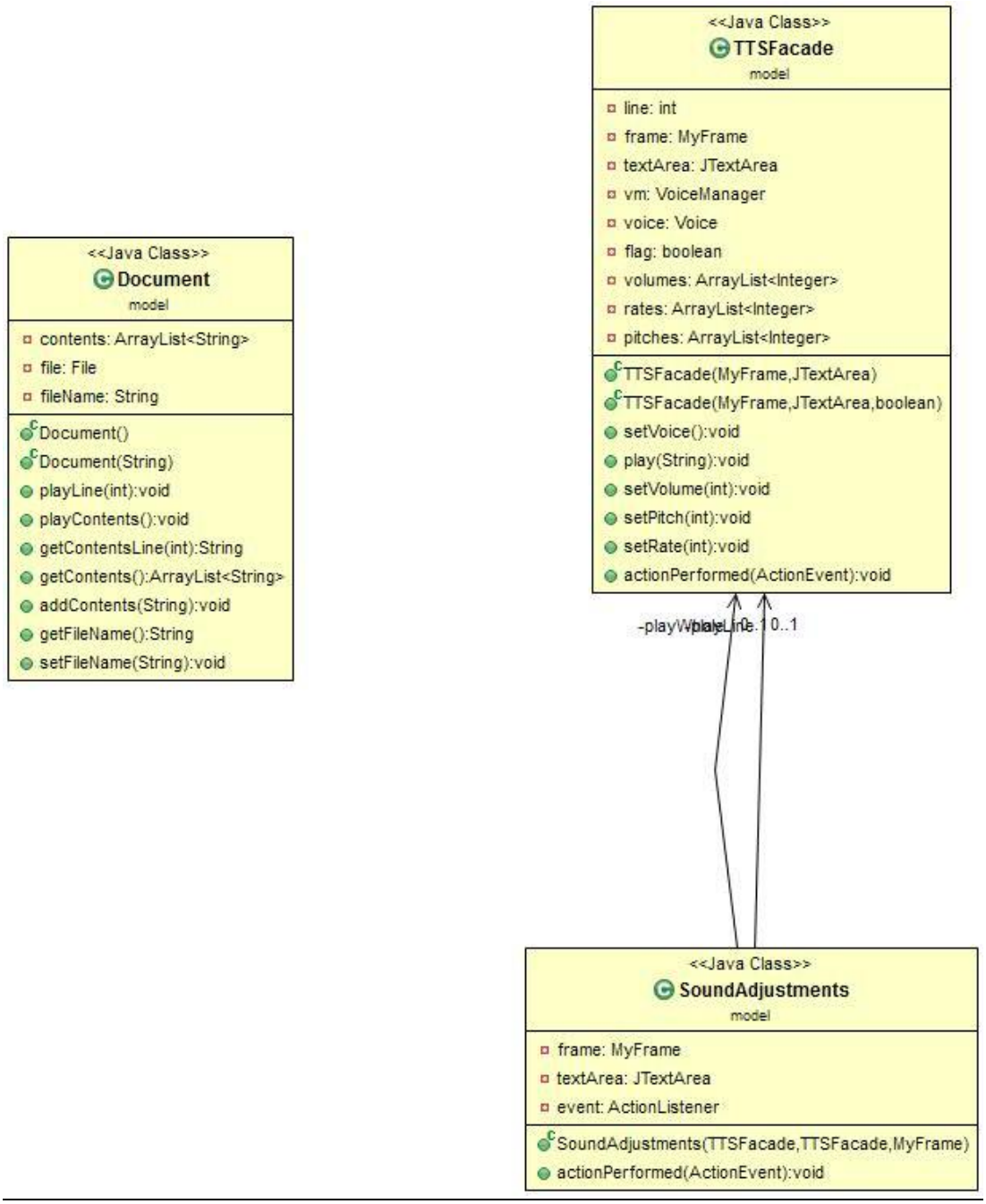
Commands Package Diagram:



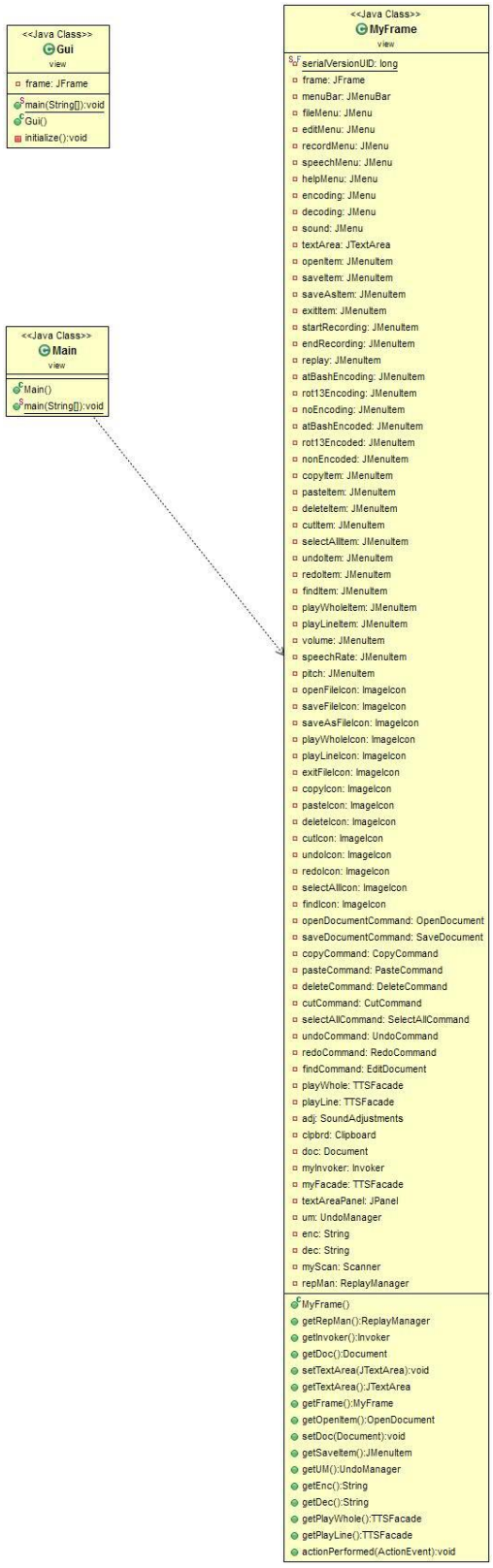
Input Package Diagram:



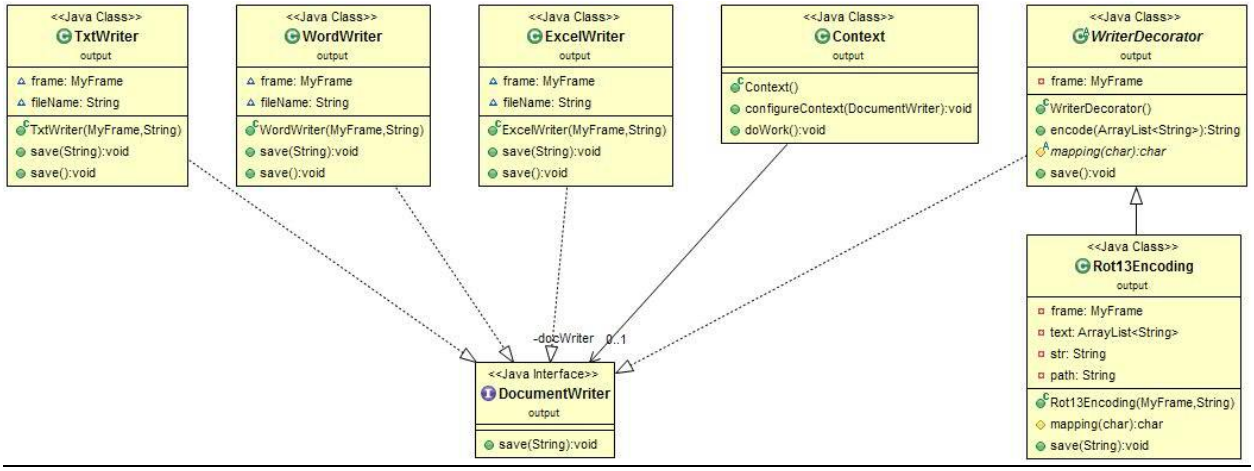
Model Package Diagram:



View Package Diagram:



Output Package Diagram:



<Document the classes that are included in this release in terms of CRC cards according to the template that is given below.>

Class Name: OpenDocument	
Responsibilities: ▪ Responsible for opening different types files including encoded files	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ Document ▪ MyFrame ▪ ReplayManager

Class Name: Save Document	
Responsibilities: ▪ The main class responsible for saving different types of file including encoded files	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ Document ▪ MyFrame ▪ ReplayManager ▪ OpenDocument ▪ Rot13Encoding

	▪ AtBashEncoding

Class Name: Exit	
Responsibilities: ▪ Responsible for closing the application	**Collaborations:** ▪ Implements ActionListener

Class Name: PasteCommand	
Responsibilities: ▪ Responsible for pasting the chosen text	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame

Class Name: CopyCommand	
Responsibilities: ▪ Responsible for coping the chosen text	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame

Class Name: CutCommand	
Responsibilities: ▪ Responsible for cutting the chosen text	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame

Class Name: DeleteCommand	
Responsibilities: ▪ Responsible for deleting the chosen text	**Collaborations:** ▪ Implements ActionListener ▪ Invoker

	▪ MyFrame

Class Name: UndoCommand	
Responsibilities: ▪ Responsible for undoing the previous action	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame ▪ UndoManager

Class Name: RedoCommand	
Responsibilities: ▪ Responsible for redoing the previous action	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame

Class Name: Invoker	
Responsibilities: ▪ Responsible for linking the “action” with the appropriate class “execution” ▪ Responsible for keeping a hashmap of <command_name, ActionListener>	**Collaborations:** ▪ All classes of the “commands” package, except “Exit”,”PlayWhole”.

Class Name: SelectAllCommand	
Responsibilities: ▪ Responsible for selecting all text	**Collaborations:** ▪ Implements ActionListener ▪ Invoker ▪ MyFrame

Class Name: PlayWhole	
Responsibilities: ▪ Responsible for transforming all text to	**Collaborations:** ▪ Implements ActionListener

audio	▪ MyFrame

Class Name: ReplayManager	
Responsibilities: ▪ Responsible for keeping a registry of all actions and replay them	**Collaborations:** ▪ Implements ActionListener ▪ MyFrame ▪ UndoManager

Class Name: DocumentReader	
Responsibilities: ▪ Interface class that creates the read() method	**Collaborations:** ▪ ExcelReader ▪ WordReader ▪ TxtReader

Class Name: ExcelReader	
Responsibilities: ▪ Responsible for reading Excel files	**Collaborations:** ▪ Implements DocumentReader ▪ OpenDocument

Class Name: WordReader	
Responsibilities: ▪ Responsible for reading Word files	**Collaborations:** ▪ Implements DocumentReader

Class Name: TxtReader	
Responsibilities: ▪ Responsible for reading txt files	**Collaborations:** ▪ Implements DocumentReader ▪ OpenDocument ▪ Document

Class Name: ReaderDecorator	
Responsibilities: ▪ Abstract class responsible for decoding encoded files. Create the decode() method and the abstract method mapping()	**Collaborations:** ▪ Rot13Decode ▪ AtBashDecode

Class Name: AtBashDecode	
Responsibilities: ▪ Responsible for decoding atBash encoded files	**Collaborations:** ▪ Extends ReaderDecorator ▪ MyFrame ▪ OpenDocument

Class Name: Rot13Decode	
Responsibilities: ▪ Responsible for decoding rot13 encoded files	**Collaborations:** ▪ Extends ReaderDecorator ▪ MyFrame ▪ OpenDocument

Class Name: Document	
Responsibilities: ▪ Responsible for reading opened file and displays it in text area	**Collaborations:** ▪ MyFrame ▪ SaveDocument ▪ TxtReader ▪ Invoker

Class Name: TTSFacade	
Responsibilities: ▪ Responsible for speech	**Collaborations:** ▪ Implements ActionListener

	▪ MyFrame

Class Name: SoundAdjustments	
Responsibilities: ▪ Responsible for adjusting volume, speech rate and pitch rate	**Collaborations:** ▪ Implements ActionListener ▪ TTSFacade ▪ MyFrame

Class Name: DocumentWriter	
Responsibilities: ▪ Interface class that creates save() method	**Collaborations:** ▪ WordWriter ▪ ExcelWriter ▪ TxtWriter

Class Name: TxtWriter	
Responsibilities: ▪ Responsible for saving a txt file	**Collaborations:** ▪ Implements DocumentWriter ▪ MyFrame ▪ SaveDocument ▪ AtBashEncoding ▪ Rot13Encoding

Class Name: ExcelWriter	
Responsibilities: ▪ Responsible for saving an Excel file	**Collaborations:** ▪ Implements DocumentWriter ▪ MyFrame ▪ SaveDocument ▪ AtBashEncoding ▪ Rot13Encoding

Class Name: WordWriter	
Responsibilities: ▪ Responsible for saving a Word file	**Collaborations:** ▪ Implements DocumentWriter ▪ MyFrame ▪ SaveDocument ▪ AtBashEncoding ▪ Rot13Encoding

Class Name: WriterDecorator	
Responsibilities: ▪ Abstract class responsible for encoding decoded files. Create the encode() method and the abstract method mapping()	**Collaborations:** ▪ Implements DocumentWriter ▪ MyFrame ▪ Rot13Encoding ▪ AtBashEncoding

Class Name: AtBashEncoding	
Responsibilities: ▪ Responsible for encoding the files	**Collaborations:** ▪ Extends WriterDecorator ▪ MyFrame ▪ SaveDocument ▪ ExcelWriter ▪ WordWriter ▪ TxtWriter

Class Name: Rot13Encoding	
Responsibilities: ▪ Responsible for encoding the files	**Collaborations:** ▪ Extends WriterDecorator ▪ MyFrame ▪ SaveDocument ▪ ExcelWriter

	▪ WordWriter ▪ TxtWriter

Class Name: Gui	
Responsibilities: ▪ Responsible for creating the application's window	**Collaborations:** ▪ -

Class Name: Main	
Responsibilities: ▪ The main class of the application	**Collaborations:** ▪ MyFrame

Class Name: MyFrame	
Responsibilities: ▪ Responsible for the graphical interface and the call of all classes	**Collaborations:** ▪ All classes except of gui and main