AdvancedText2SpeechEditor

Sprint Report

Guardians of the Galaxy

Iliopoulos Panagiotis 4060

Katsi Athanasia 4077

Papadopoulos Xenofon-Rafail 4141

VERSIONS HISTORY

Date	Version	Description	Author
10/05/2021	alpha	Created a JFrame, added a JTextArea, JMenu and Open/Save buttons. Implemented a command factory and added the functionality to open a .txt file and display the contents in the JTextArea.	Xenofon, Athanasia
12/05/2021	Pre-release 0.1	Downloaded POI and added support for Opening and Saving word and excel documents. Added a JTextField to display the path of the opened document. Moved the buttons to better positions.	Xenofon, Athanasia
14/05/2021	Pre-release 0.3	Fixed bugs. Downloaded FreeTTS and created a button to Play the contents of the opened document. The JTextArea is disabled by default and is toggling with the Edit button. If no document is opened but JTextArea has text in it, the contents will be stored to temporary document in memory.	Xenofon, Athanasia
15/05/2021	Pre-release 0.5	Fixed bugs. Added Play Selected Text and Play Line functionalities. Implemented the Volume, Pitch, Rate settings. Added labels and moved buttons to better positions.	Xenofon, Panagiotis
16/05/2021	Pre-release 0.8	Added support for Rot13, AtBsash and None encryption. The user can open or save a document with either.	Xenofon, Panagiotis
25/05/2021	1.0	Fixed bugs. Tweaked the code to implement the record/replay commands functionality.	Xenofon
28/05/2021	1.1	Added Dark and Light Theme support. Added project to GitHub with screenshots and a video tutorial.	Xenofon

1 Introduction

1.1 Text2speech is an advanced text to speech application that converts text from a file or the build-in editor to speech. The user can open a word (.docx) or an excel (.xslx) document, edit and save the contents of it with no, rot13 or atbash encryption and convert it to speech. The voice is fully customizable, the volume, pitch and rate can be adjusted with sliders. The user can select to play the whole contents of the document, highlight specific text or select a line by pressing the button corresponding to the action. The app also allows the user to record the button presses or actions and replay them at a later time.

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

<For the user stories included in this release specify below corresponding tests using a typical tabular form.>

2.1 Scrum team

Product Owner	Apostolos Zarras
Scrum Master	Papadopoulos Xenofon - Rafail
Development Team	Papadopoulos Xenofon - Rafail Katsi Athanasia Iliopoulos Panagiotis

2.2 Sprints

<List below the sprints that you performed and the user stories that have been realized in each Sprint>

Sprint No	Begin Date	End Date	Number of days	User stories
1	**10/05/2021**	**12/05/2021**	**3**	**US1, US3**
2	**13/05/2021**	**14/05/2021**	**2**	**US2, US4**
3	**15/05/2021**	**16/05/2021**	**2**	**US5, US6**
4	**22/05/2021**	**25/05/2021**	**4**	**US7, US8, US9**

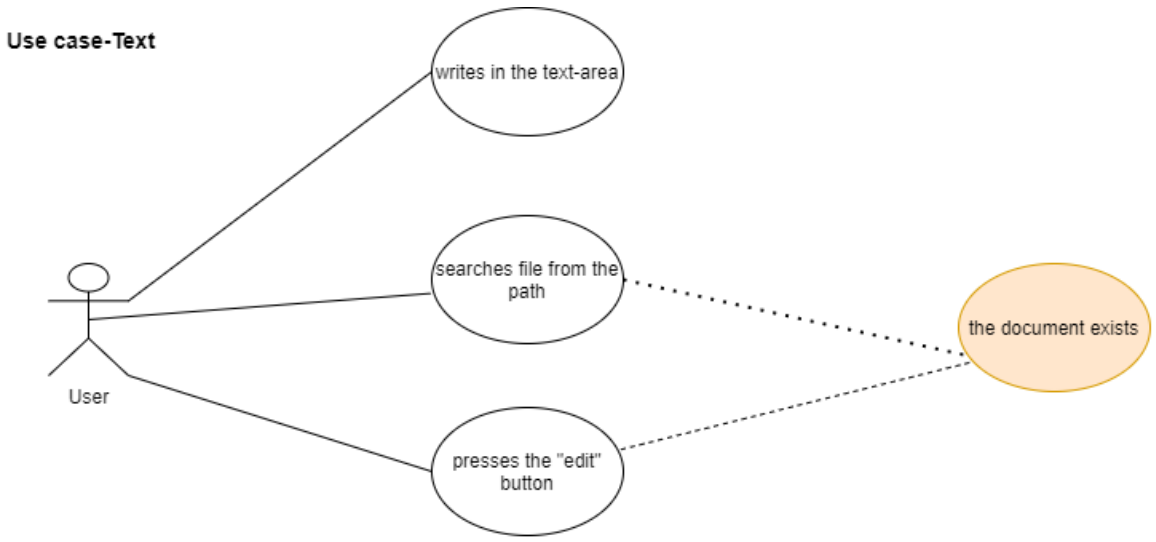
2.3 User Stories Tests

Test No	Description
1	For this test we create in the test code an OpenDocument command, associate it with a particular Document object, execute the command, and then check if the contents of the Document object are equal to the contents of the file that has been opened. We do this for every extension and encryption.
2	For this test we create an EditDocument command, associate it with a particular Document object, execute the command to perform some pre-defined changes to the contents of the Document object and then checks if the contents of the Document object after the command have changed as expected.
3	For this test we create in the test code a SaveDocument command, associate it with a particular Document object, execute it to save the contents of the Document object to a file and then read the contents of the file back to verify that they are equal with the contents of the Document object that has been saved.
4	For this test we create a FakeTTSFacade subclass of the TTSFacade class that has a private field named playedContents that is used for storing the text that is given an input to the play method each time the method is. In the test code, we create FakeTTSFacade object, a Document object that uses the FakeTTSFacade object and a DocumentToSpeech command that is associated with the Document object. After executing the command, we can check if the value of playedContents is equal with the contents of the Document object a that is involved in the execution of the command.
5	To test this story, we proceed as in the case of US4.

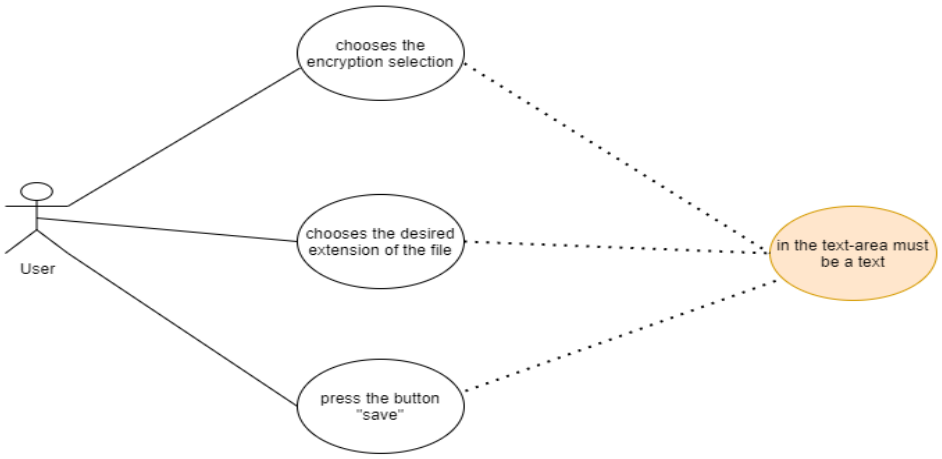
6	To test this story, we can proceed as in the case of US4, using additional fields in the FakeTTSFacade subclass for keeping the values of the audio tuning parameters (pitch, volume, rate).
7	For this test we create a StartRecording command, associate it with a particular ReplayManager, emulate the pressing of start recording button, and then check if the recordingStatus of the ReplayManager object is enabled.
8	To test this story, we create a StartRecording command, associate it with a Document object, a ReplayManager object, execute the command, create and execute a number of commands that transform text to speech, create a Replay command, execute it and finally check if playedContents contains the text that has been played by the replayed commands.
9	For this test we create a StartRecording command, associate it with a particular ReplayManager, emulate the pressing of end recording button, and then check if the recordingStatus of the ReplayManager object is disabled.

3 Use Cases

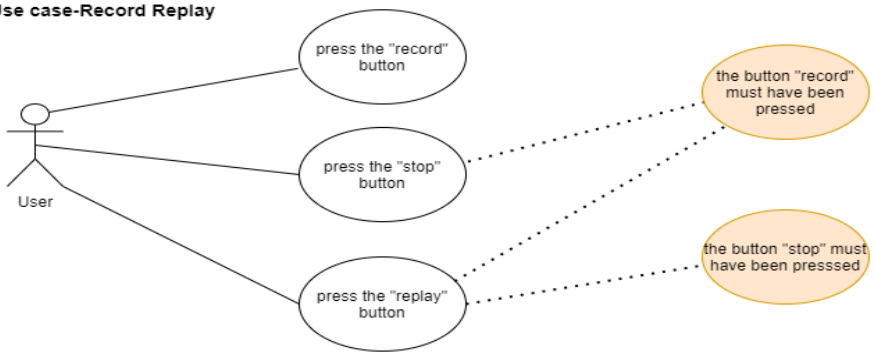
<Specify the concrete Use Cases that describe the interaction of the user with the applications, as derived from the abstract user stories. Give a **UML Use Case diagram** and the **detailed use case descriptions**.>



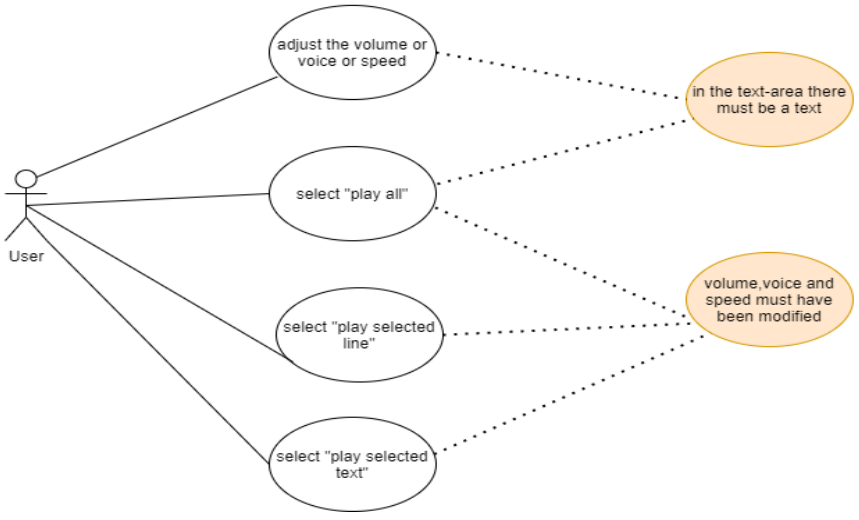
Use case-Read Save Document



Use case-Record Replay



Use case-Play and its operations



3.1 <Use Case Text>

Use case ID	1
Actors	The user
Pre conditions	The user must have opened the application. The document searched must exist.
Main flow of events	1. If the user writes in text-area, the system displays a text in the text-area. 2. If the user searches file from path, this file displayed in the text-area. 3. If the user presses the edit button, the document will be modified.
Alternative flow 1	If the user presses the edit button and no document is opened, a temporary will be created in memory.
Post conditions	1.After the document is opened its contents will be displayed in the TextArea. 2.After editing the document, the contents are updated.

3.2 <Use Case Read-Save document>

Use case ID	2
Actors	The user
Pre conditions	There must be text in the text area (read operation).
Main flow of events	1.If the user chooses the encryption method, if any, of the text, the system displays the codification modified. 2.If the user chooses the desired extension/type of the file to open/save, the file gets the appropriate extension. 3.If the user chooses to save text to file, the system saves it in the path which user selected.
Alternative flow 1	If the user forgets to write an extension of the saved file, one will be added automatically.
Post conditions	After saving the file, the file will be displayed to the correct path the user specified.

3.3 <Use Case Record-Replay>

Use case ID	3
Actors	The user
Pre conditions	User has to press the record button. User has to press the stop button and then the replay.
Main flow of events	1.If the user presses record, the recording of the commands start. 2.If the user presses stop, the system stop the recording. 3. If the user presses replay, the recorded commands are replayed.
Alternative flow 1	If the buttons record and stop haven't been the record button won't do anything.
Post conditions	If everything went right the replay button will replay all the recorded commands.

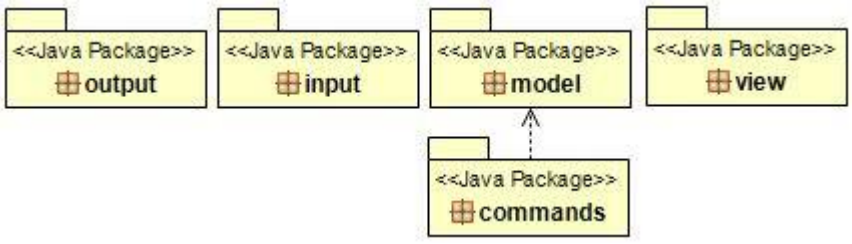
3.4 <Use Case Play and its operations >

Use case ID	4
Actors	The user
Pre conditions	Text area must contain text. The user must adjust the volume, volume or speed for the speaking to start.
Main flow of events	1. The user adjusts the volume or the voice or the speed of the speech by selecting the corresponding keys. 2. If the user selects "play all", the system pronounces all the text. 3. If the user presses the button "play selected line", the system pronounces a specific line. 4.If the user selects "play selected text", the system pronounces specific part of the text.
Alternative flow 1	The user cannot change the volume or speed or tone of voice once a text has been spoken.
Alternative flow 2	If the user selects one of the play buttons without text in the text area, nothing will be spoken.
Post conditions	If all goes well, the text in the text area (or part of it) should be spoken.

4 Design

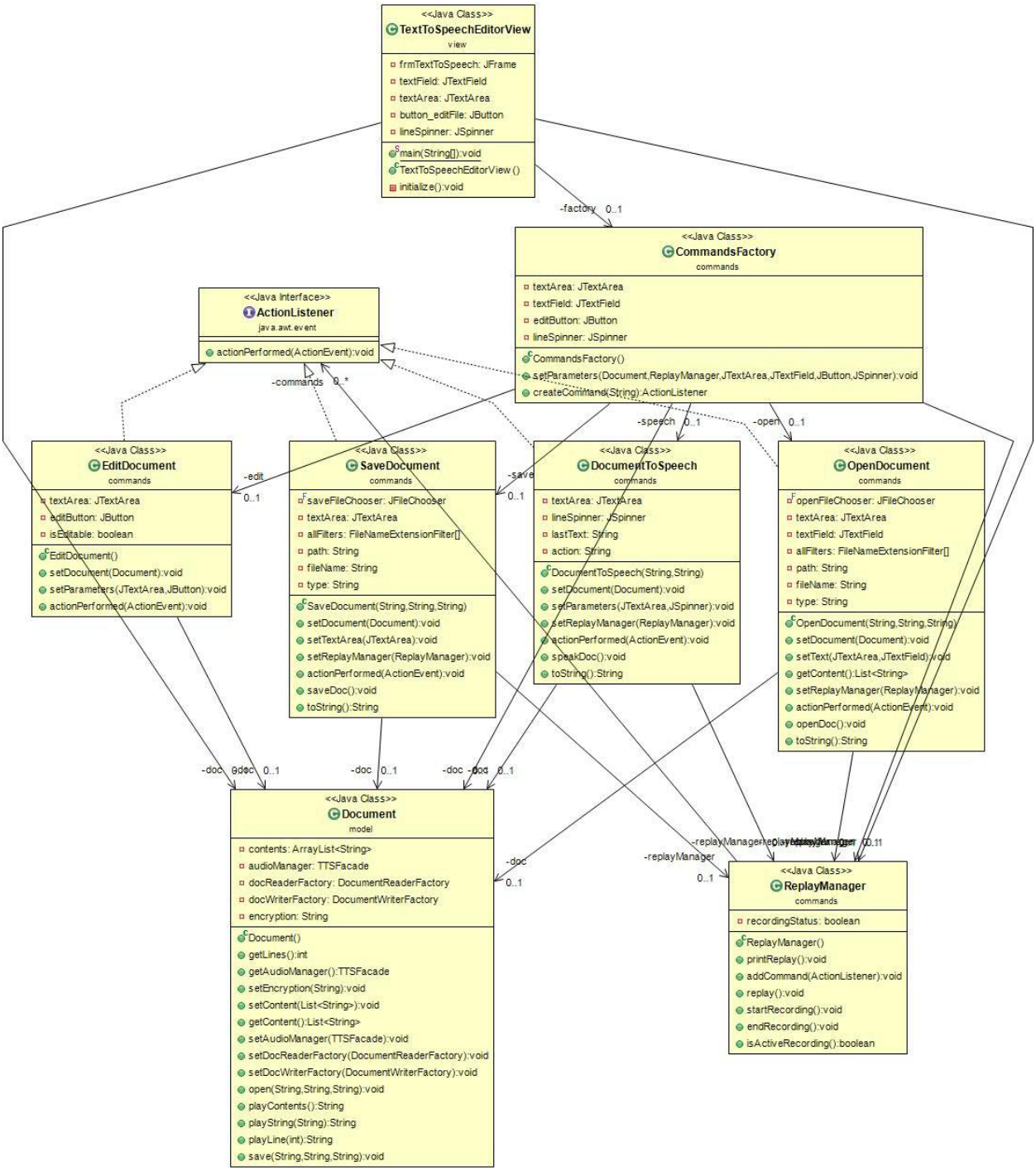
4.1 Architecture

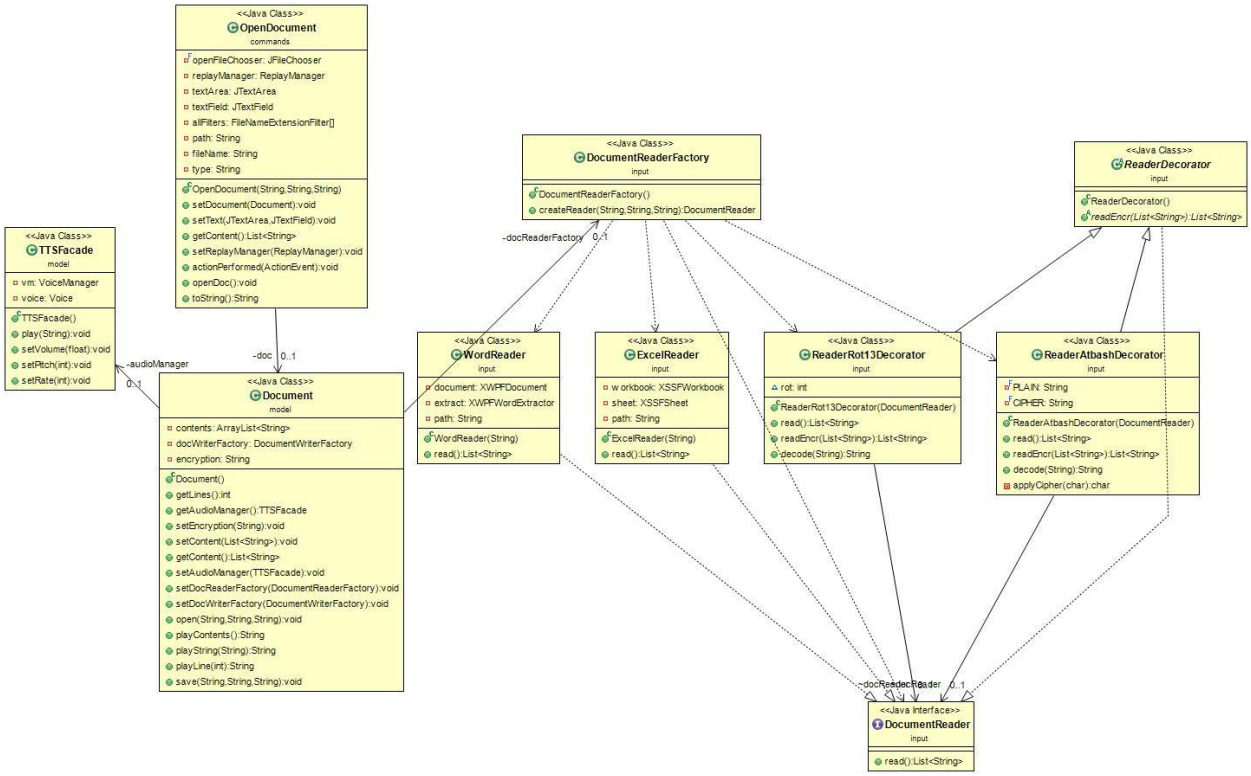
<Specify the overall architecture for this release in terms of a **UML package diagram**.>

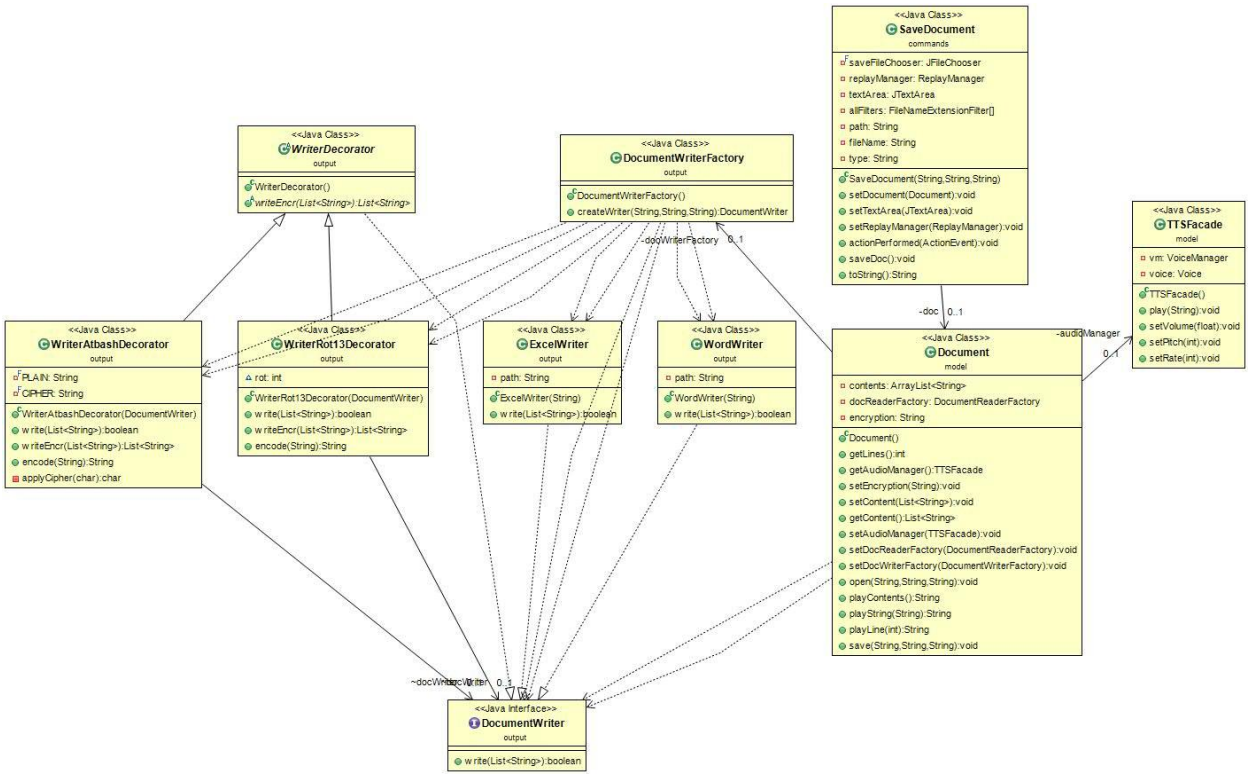


4.2 Design

<Specify the detailed design for this release in terms of **UML class diagrams**.>


<<Java Class>>
TextToSpeechEditorView
view
frmTextToSpeech: JFrame
textField: JTextField
textArea: JTextArea
button_editFile: JButton
lineSpinner: JSpinner
main(String[]):void
TextToSpeechEditorView()
initialize():void
-factory 0..1
<<Java Class>>
CommandsFactory
commands
textArea: JTextArea
textField: JTextField
editButton: JButton
lineSpinner: JSpinner
CommandsFactory()
setParameters(Document,ReplayManager,JTextArea,JTextField,JButton,JSpinner):void
createCommand(String):ActionListener
<<Java Interface>>
ActionListener
java.awt.event
actionPerformed(ActionEvent):void
-commands 0..*
-speech 0..1
-open 0..1
<<Java Class>>
EditDocument
commands
textArea: JTextArea
editButton: JButton
isEditable: boolean
EditDocument()
setDocument(Document):void
setParameters(JTextArea,JButton):void
actionPerformed(ActionEvent):void
-edit 0..1
<<Java Class>>
SaveDocument
commands
saveFileChooser: JFileChooser
textArea: JTextArea
allFilters: FileNameExtensionFilter[]
path: String
fileName: String
type: String
SaveDocument(String,String,String)
setDocument(Document):void
setTextArea(JTextArea):void
setReplayManager(ReplayManager):void
actionPerformed(ActionEvent):void
saveDoc():void
toString():String
-save 0..1
<<Java Class>>
DocumentToSpeech
commands
textArea: JTextArea
lineSpinner: JSpinner
lastText: String
action: String
DocumentToSpeech(String,String)
setDocument(Document):void
setParameters(JTextArea,JSpinner):void
setReplayManager(ReplayManager):void
actionPerformed(ActionEvent):void
speakDoc():void
toString():String
<<Java Class>>
OpenDocument
commands
openFileChooser: JFileChooser
textArea: JTextArea
textField: JTextField
allFilters: FileNameExtensionFilter[]
path: String
fileName: String
type: String
OpenDocument(String,String,String)
setDocument(Document):void
setText(JTextArea,JTextField):void
getContent():List<String>
setReplayManager(ReplayManager):void
actionPerformed(ActionEvent):void
openDoc():void
toString():String
-doc 0..1
-doc 0..1
-doc 0..1
-doc 0..1
<<Java Class>>
Document
model
contents: ArrayList<String>
audioManager: TTSFacade
docReaderFactory: DocumentReaderFactory
docWriterFactory: DocumentWriterFactory
encryption: String
Document()
getLines():int
getAudioManager():TTSFacade
setEncryption(String):void
setContent(List<String>):void
getContent():List<String>
setAudioManager(TTSFacade):void
setDocReaderFactory(DocumentReaderFactory):void
setDocWriterFactory(DocumentWriterFactory):void
open(String,String,String):void
playContents():String
playString(String):String
playLine(int):String
save(String,String,String):void
-doc
0..1
-replayManager
0..1
<<Java Class>>
ReplayManager
commands
recordingStatus: boolean
ReplayManager()
printReplay():void
addCommand(ActionListener):void
replay():void
startRecording():void
endRecording():void
isActiveRecording():boolean


<<Java Class>>
OpenDocument
commands
openFileChooser: JFileChooser
replayManager: ReplayManager
textArea: JTextArea
textField: JTextField
allFilters: FileNameExtensionFilter[]
path: String
fileName: String
type: String
OpenDocument(String,String,String)
setDocument(Document):void
setText(JTextArea,JTextField):void
getContent():List<String>
setReplayManager(ReplayManager):void
actionPerformed(ActionEvent):void
openDoc():void
toString():String
<<Java Class>>
DocumentReaderFactory
input
DocumentReaderFactory()
createReader(String,String,String):DocumentReader
<<Java Class>>
ReaderDecorator
input
ReaderDecorator()
readEncr(List<String>):List<String>
<<Java Class>>
TTSFacade
model
vm: VoiceManager
voice: Voice
TTSFacade()
play(String):void
setVolume(float):void
setPitch(int):void
setRate(int):void
-docReaderFactory 0..1
-doc 0..1
-audioManager
0..1
<<Java Class>>
Document
model
contents: ArrayList<String>
docWriterFactory: DocumentWriterFactory
encryption: String
Document()
getLines():int
getAudioManager():TTSFacade
setEncryption(String):void
setContent(List<String>):void
getContent():List<String>
setAudioManager(TTSFacade):void
setDocReaderFactory(DocumentReaderFactory):void
setDocWriterFactory(DocumentWriterFactory):void
open(String,String,String):void
playContents():String
playString(String):String
playLine(int):String
save(String,String,String):void
<<Java Class>>
WordReader
input
document: XWPFDocument
extract: XWPFWordExtractor
path: String
WordReader(String)
read():List<String>
<<Java Class>>
ExcelReader
input
workbook: XSSFWorkbook
sheet: XSSFSheet
path: String
ExcelReader(String)
read():List<String>
<<Java Class>>
ReaderRot13Decorator
input
rot: int
ReaderRot13Decorator(DocumentReader)
read():List<String>
readEncr(List<String>):List<String>
decode(String):String
<<Java Class>>
ReaderAtbashDecorator
input
PLAIN: String
CIPHER: String
ReaderAtbashDecorator(DocumentReader)
read():List<String>
readEncr(List<String>):List<String>
decode(String):String
applyCipher(char):char
-docReaderReader 0..1
<<Java Interface>>
DocumentReader
input
read():List<String>


<<Java Class>>
SaveDocument
commands
saveFileChooser: JFileChooser
replayManager: ReplayManager
textArea: JTextArea
allFilters: FileNameExtensionFilter[]
path: String
fileName: String
type: String
SaveDocument(String,String,String)
setDocument(Document):void
setTextArea(JTextArea):void
setReplayManager(ReplayManager):void
actionPerformed(ActionEvent):void
saveDoc():void
toString():String
<<Java Class>>
WriterDecorator
output
WriterDecorator()
writeEncr(List<String>):List<String>
<<Java Class>>
DocumentWriterFactory
output
DocumentWriterFactory()
createWriter(String,String,String):DocumentWriter
<<Java Class>>
TTSFacade
model
vm: VoiceManager
voice: Voice
TTSFacade()
play(String):void
setVolume(float):void
setPitch(int):void
setRate(int):void
-docWriterFactory 0..1
-doc 0..1
-audioManager
0..1
<<Java Class>>
WriterAtbashDecorator
output
PLAIN: String
CIPHER: String
WriterAtbashDecorator(DocumentWriter)
write(List<String>):boolean
writeEncr(List<String>):List<String>
encode(String):String
applyCipher(char):char
<<Java Class>>
WriterRot13Decorator
output
rot: int
WriterRot13Decorator(DocumentWriter)
write(List<String>):boolean
writeEncr(List<String>):List<String>
encode(String):String
<<Java Class>>
ExcelWriter
output
path: String
ExcelWriter(String)
write(List<String>):boolean
<<Java Class>>
WordWriter
output
path: String
WordWriter(String)
write(List<String>):boolean
<<Java Class>>
Document
model
contents: ArrayList<String>
docReaderFactory: DocumentReaderFactory
encryption: String
Document()
getLines():int
getAudioManager():TTSFacade
setEncryption(String):void
setContent(List<String>):void
getContent():List<String>
setAudioManager(TTSFacade):void
setDocReaderFactory(DocumentReaderFactory):void
setDocWriterFactory(DocumentWriterFactory):void
open(String,String,String):void
playContents():String
playString(String):String
playLine(int):String
save(String,String,String):void
-docWriterWriter 0..1
<<Java Interface>>
DocumentWriter
output
write(List<String>):boolean

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

Class Name: DocumentReaderFactory	
Responsibilities: ▪ Return the correct Document Reader object.	**Collaborations:** ▪ ReaderAtBashDecorator ▪ ReaderRot13Decorator ▪ WordReader ▪ ExcelReader ▪ Document

Class Name: ExcelReader	
Responsibilities: ▪ Read the contents of an excel (.xlsx) document.	**Collaborations:** ▪ DocumentReaderFactory

Class Name: WordReader	
Responsibilities: ▪ Read the contents of a word (.docx) document.	**Collaborations:** ▪ DocumentReaderFactory

Class Name: ReaderAtbashDecorator	
Responsibilities: ▪ Decode the text given from readers with atBash encryption.	**Collaborations:** ▪ DocumentReader ▪ DocumentReaderFactory

Class Name: ReaderRot13Decorator	
Responsibilities: ▪ Decode the text given from readers with Rot13 encryption.	**Collaborations:** ▪ DocumentReader ▪ DocumentReaderFactory

Class Name: ReaderDecorator	
Responsibilities: ▪ Abstract class.	**Collaborations:** ▪ DocumentReader

Class Name: DocumentWriterFactory	
Responsibilities: ▪ Return the correct Document Writer object.	**Collaborations:** ▪ WriterAtbashDecorator ▪ WriterRot13Decorator ▪ WordWriter ▪ ExcelWriter ▪ Document

Class Name: ExcelWriter	
Responsibilities: ▪ Write the contents to an excel (.xlsx) document.	**Collaborations:** ▪ DocumentWriterFactory

Class Name: WordWriter	
Responsibilities: ▪ Write the contents to a word (.docx) document.	**Collaborations:** ▪ DocumentWriterFactory

Class Name: WriteAtbashDecorator	
Responsibilities: ▪ Encode the text and then give it to writers with atBash encryption.	**Collaborations:** ▪ DocumentWriter ▪ DocumentWriterFactory

Class Name: WriterRot13Decorator	
Responsibilities: ▪ Encode the text and then give it to writers with Rot13 encryption.	**Collaborations:** ▪ DocumentWriter ▪ DocumentWriterFactory

Class Name: WriterDecorator	
Responsibilities: ▪ Abstract class.	**Collaborations:** ▪ DocumentWriter

Class Name: CommandsFactory	
Responsibilities: ▪ Return the correct ActionListener command. ▪ Pass the correct parameters to this ActionListener object.	**Collaborations:** ▪ Document ▪ ReplayManager ▪ OpenDocument ▪ SaveDocument ▪ EditDocument ▪ DocumentToSpeech ▪ Text2SpeechEditorView

Class Name: DocumentToSpeech	
Responsibilities: ▪ Detect which play button was pressed. ▪ Play the correct content according to that button.	**Collaborations:** ▪ Document ▪ ReplayManager ▪ CommandsFactory

Class Name: StartRecording	
Responsibilities: ▪ Detect which record button was pressed. ▪ Start, stop or replay actions accordingly.	**Collaborations:** ▪ ReplayManager ▪ CommandsFactory

Class Name: EditDocument	
Responsibilities: ▪ Toggle the text area as editable and not. ▪ Save the contents of text area to document contents.	**Collaborations:** ▪ CommandsFactory

Class Name: OpenDocument	
Responsibilities: ▪ Make user select a valid document path. ▪ Open a document from the selected path.	**Collaborations:** ▪ ReplayManager ▪ Document ▪ CommandsFactory

Class Name: SaveDocument	
Responsibilities: ▪ Make user select a valid location and name to save document. ▪ Save the document.	**Collaborations:** ▪ ReplayManager ▪ Document ▪ CommandsFactory

Class Name: ReplayManager	
Responsibilities: ▪ Store to an array all actions recorded. ▪ Start / End recording. ▪ Replay recorded commands.	**Collaborations:** ▪ OpenDocument ▪ SaveDocument ▪ DocumentToSpeech ▪ CommandsFactory ▪ Text2SpeechEditorView

Class Name: TTSFacade	
Responsibilities: ▪ Voice allocation. ▪ Specific Text to Speech.	**Collaborations:** ▪ Document

Class Name: Document	
Responsibilities: ▪ Store all contents of a document. ▪ Use the reader / writers to open / save the document.	**Collaborations:** ▪ DocumentReader ▪ DocumentReaderFactory ▪ DocumentWriter ▪ DocumentWriterFactory ▪ TTSFacade ▪ Text2SpeechEditorView ▪ SaveDocument ▪ OpenDocument ▪ DocumentToSpeech ▪ CommandsFactory

Class Name: TextToSpeechEditorView	
Responsibilities: ▪ GUI ▪ Initialize the document, replayManager and JComponets.	**Collaborations:** ▪ CommandsFactory ▪ Document ▪ ReplayManager ▪ CommandsFactory