Text2SpeechEditor

Sprint No 2 Report

Angelos Dimokas 2683

VERSIONS HISTORY

Date	Version	Description	Author
25/5/2020	1	Final TTS document editor application without the replay function	Dimokas Angelos

1 Introduction

This document provides information concerning the **<2>** sprint of the project.

1.1 Purpose

To educate and inform about the Text To Speech application project.

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	Angelos Dimokas
Scrum Master	Angelos Dimokas
Development Team	Angelos Dimokas

2.2 Sprint Backlog

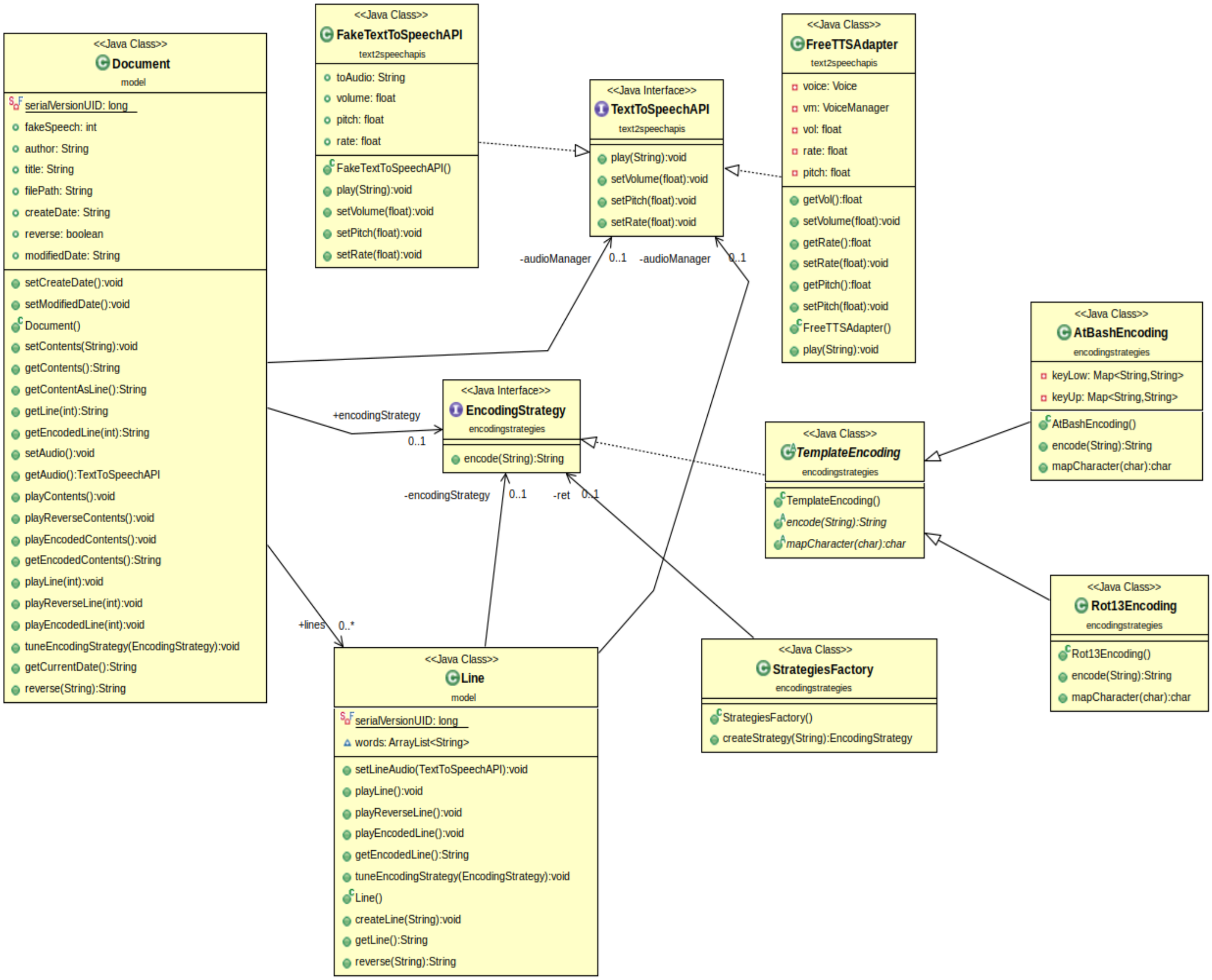
<List below the user stories that have been realized in this Sprint alongside their corresponding tests>

User-story 1: The user, creates a new empty document, by giving its title and author. The application automatically records the creation date	**Test Case 1**: Creates and executes new Document command. User inputs random title and author and finally the test checks if the new Document is empty
User-story 2: The user, edits the contents of the document, via the application's user interface. With 2 available option, first Replace word is for inputting a string and changing all occurrences of it with second input. Second is	**Test Case 2:** Creates a TextEditor, sets contents of textArea and Document to a sample text (alphabet) and executes an Edit command. The user must enter any lowercase letter and then replace it with anything. Finally the test checks if the contents of the Document has changed comparing to the sample text it has been set to before the execution of the command.
User-story 3: The user, can save the contents of the document to disk by providing a particular filename. The application should automatically record the save date.	**Test Case 3:** Creates and executes a SaveDocument command on a Document object with some contents. Then checks if the saved Document's contents are the same as the current Document held the application.
User-story 4: The user can open the contents of an existing document from disk by providing a particular file path, or by browsing the file system folders.	**Test Case 4:** Creates and executes an OpenCommand. Then checks if the contents of the document that has been read from the disk match the current document's contents.
User-story 5: The user can transform the contents of the current document handled by the GUI, to speech.	**Test Case 5:** Sets gui's text area to be a sample text, then executes a Transform command using the FakeTextToSpeechAPI of the current document. Finally checks if the contents that was passed to the FakeTTSapi are matched with the contents of the document.
User-story 6: The user can transform a single line of a document by entering the line's number, to speech.	**Test Case 6:** Sets gui's text area to a sample text with 3 lines. Calls a TransformLine command using as line number the 2nd . Checks the contents of the

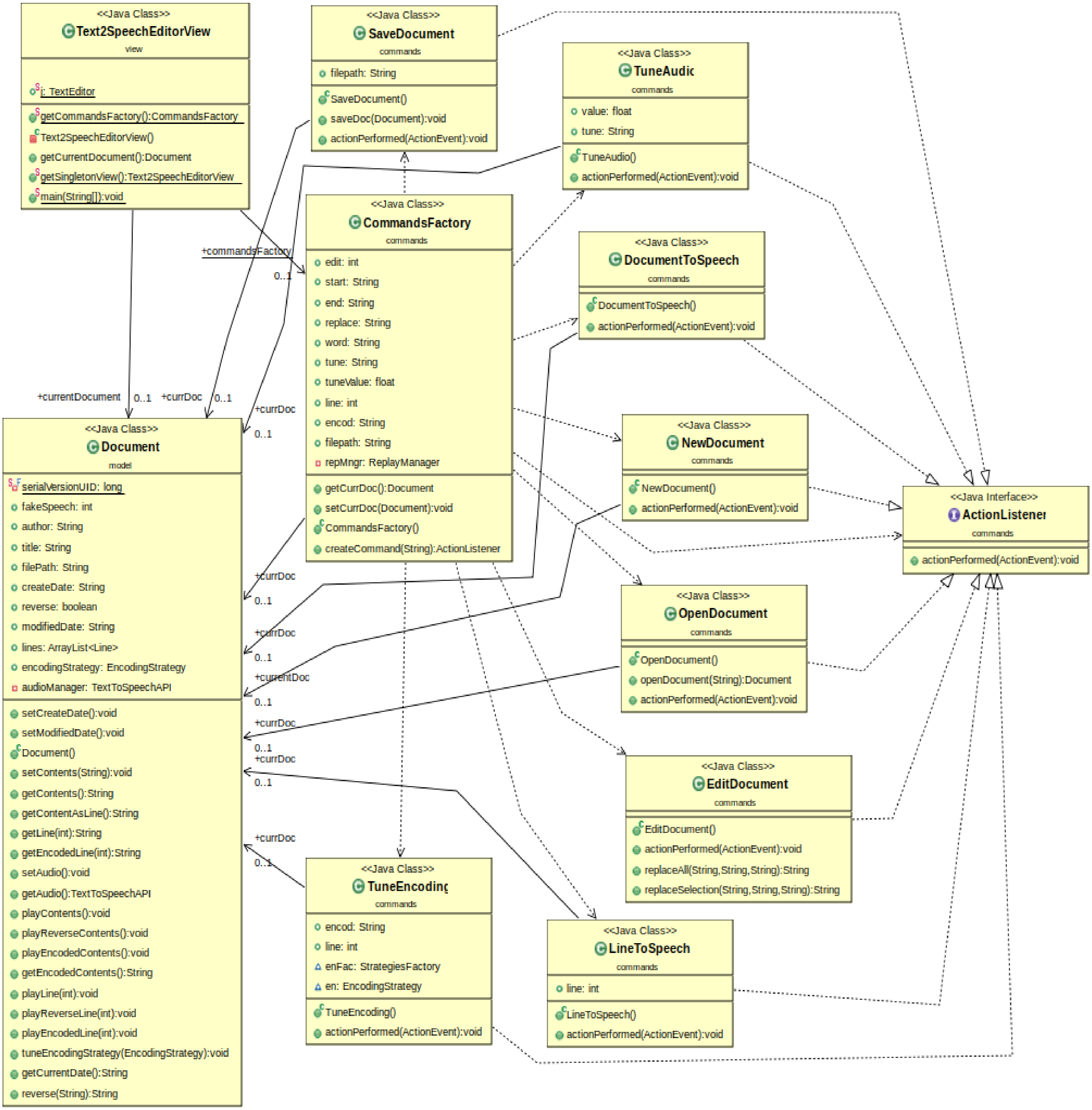
	2nd line of the document and the content that was passed to FakeTTSAPI.
User-story 7: The user can transform the contents of a document to speech in reverse. i.e. first line last and last line first and so on	**Test Case 7:** Same as Test Case 5 but test executes ReverseTransform command and checks if the contents that was passed to FakeTTSapi with the reverse contents of the document match.
User-story 8: The user can select a line by choosing its number and transforming its words to speech in reverse. I.e first word plays last and last word plays first and so on.	**Test Case 8:** Same as Test Case 6 but calls a ReverseTransformLine command on 1st line of the document. Finally it checks if the contents that was passed to FakeTTSapi an the reverse contents of the 1st line of the document match.
User-story 9: The user can encode the contents of a document and transform the encoded contents to speech.	**Test Case 9:** Sets gui's text area to a sample text, then calls Atbash encode command for the current document. After that it checks the contents of the FakeTTSapi's attribute (toAudio) with the encoded contents of the current Document. Finally does the same check with Rot13 encode command.
User-story 10: The user can select a line by choosing its number, encode it and transform it to speech.	**Test Case 10:** Sets gui's text area to a sample text with 3 lines. After that, calls EncodeLineAtbash command for the 3rd line of the current document and checks the contents of the FakeTTSapi with the encoded contents of the 3rd line of the document. Finally does the same check with the EncodeLineRot command.
User-story 11: The user is able to tune the audio parameters such as the volume, speech rate and pitch of the voice that is used by the application.	**Test Case 11:** This test case is to check if the tuneAudio commands work. There are 3 subtests for Volume, Pitch and SpeechRate. Each of them sets a fakeVolume-Pitch-Rate respectively and uses the FakeTTSapi to check if the values of its attributes are changed to the fake ones.
User-story 12: The user can tune the encoding techniques. Currently the application gives the user to choose between Atbash and Rot-13 encoding strategies.	**Test Case 12:** This test case executes an AtbashEncode command to an empty Document. After that, it saves the encoding strategy of the document to a variable. Then another Rot13Encode Command is executed. Finally the test checks if the first variable is not equal to the current encoding strategy of the document.

3 Design

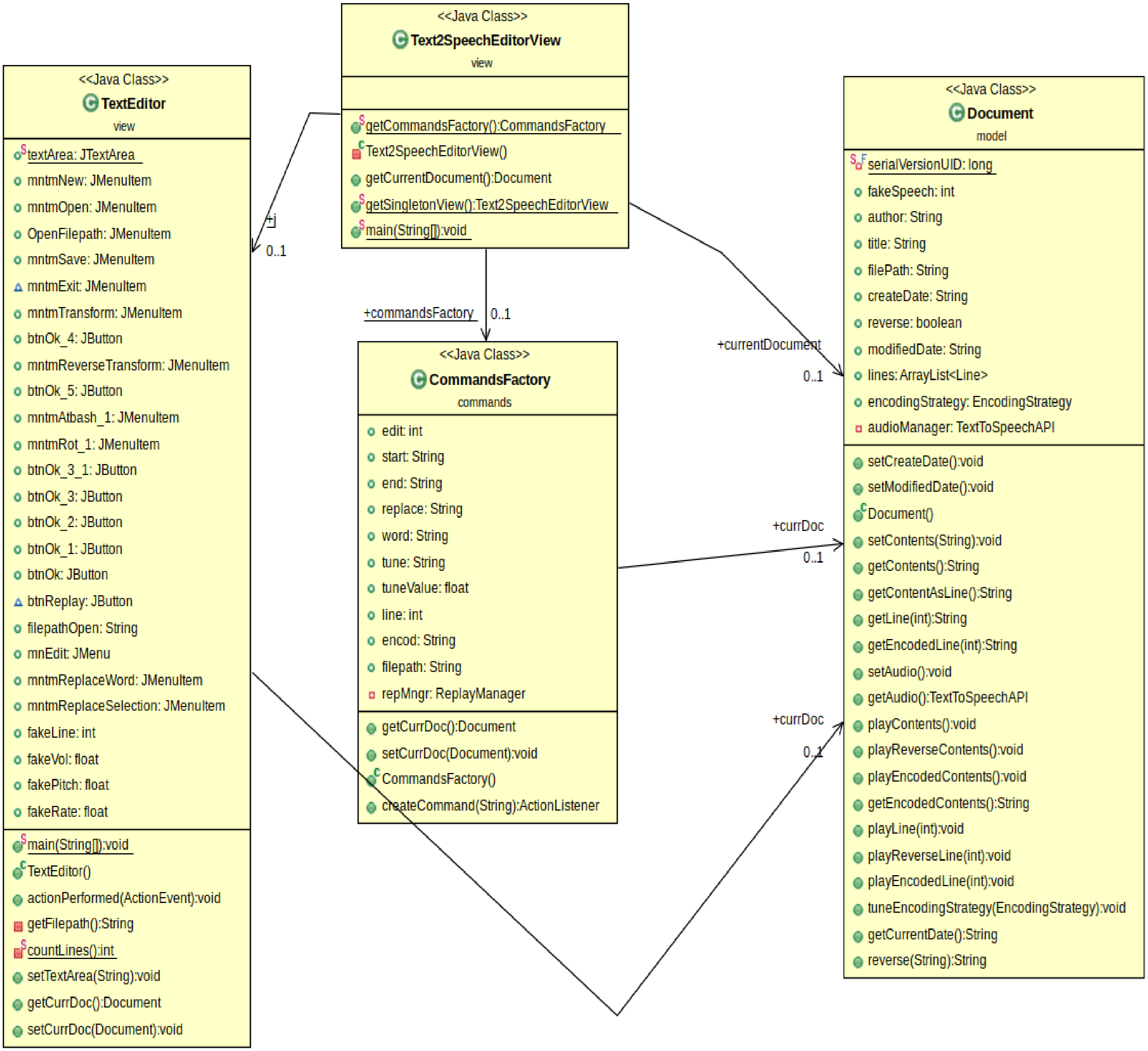
3.1 Architecture



3.1.1 encodingstrategies, text2speechapis and model packages dependecies



3.1.2 ActionListener, command classes and receiver classes with model.Document class and view.Text2SpeechEditor.



3.1.3 CommandsFactory and relationships with view package classes and document

3.2 Design

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

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

model PACKAGE CLASSES:

Class Name: Document	
Responsibilities:	**Collaborations:**
■ Holds the information about the object that is being transformed, encoded, edited, saved, opened and created by the gui classes ■ Has fields that are processed by the user or by the application such as author, title, filepath and dates of creation and modification. Also the lines Arraylist of line object for the contents. ■ Has methods to get and set the respective fields and the contents. ■ Has methods that transform and encode the contents or part of them.	■ Line objects are in the arraylist lines of Document holding each line of contents. ■ View package classes have temporary document object for passing arguments as fields to Command package classes and interfaces. ■ Command package classes also have temporary document objects for processing requests received after and actionEvent is sent by the respective buttons of the GUI. ■ Document object holds an EncodingStrategy Interface object used for encoding purposes of the contents. Also a TextToSpeechAPI Interface object that is used for transforming content to speech.

Class Name: Line	
Responsibilities:	**Collaborations:**
■ Holds an arraylist of words for each line of the Document object’s arraylist. ■ Has setter and getters for the words in the line. ■ Has methods responsible for encoding and transforming the line to Speech.	■ Direct access from Document objects for processing contents of the lines object in the arraylist lines. ■ Lines objects holds an EncodingStrategy interface same to the Document object. ■ Lines objects holds an TextToSpeechAPI interface object same to the Document object.

view PACKAGE CLASSES:

Class Name: Text2SpeechEditor	
Responsibilities:	**Collaborations:**
▪ Starts the program by creating a TextEditor class object and hearing for action events that are passed to Commands Factory class for further processing. ▪ Holds the main method of the application	▪ The class keeps a current Document object that is used by the gui and passed on to commands package. ▪ Also, a commandsFactory object that is used by the gui for hearing event from the user and starting execution of the request by passing it to the commands package.

Class Name: TextEditor	
Responsibilities:	**Collaborations:**
▪ This class keeps the Graphical User Interface used by the application. ▪ Main role is for hearing the requests that the user asks the application via the GUI.	▪ This class holds a current Document object that is used to pass arguments to the back end of the application. ▪ This class is called by the main of the Text2SpeechEditorView and uses its commandsFactory object to pass requests to the commands package.

commands PACKAGE CLASSES:

Interface Name: ActionListener	
Responsibilities:	**Collaborations:**
▪ This interface is used by the commands package classes to execute a certain action corresponding to an event.	▪ These classes implement this interface: SaveDocument, TuneAudio, DocumentToSpeech, NewDocument, OpenDocument, EditDocument, LineToSpeech, TuneEncoding. ▪ The method createCommand of the CommandsFactory class returns an object of one of the implementations of this inteface.

Class Name: CommandsFactory	
Responsibilities:	**Collaborations:**
▪ The main role of this class is to create ActionListener type objects when requested by the view package. ▪ The class has a method called createCommand responsible for creating those objects, with 1 argument that checks for the corresponding type of ActionListener implementation. ▪ Also the class has to pass some information acquired from the user in the view package to these objects via arguments or fields.	▪ Basically this class holds a currentDocument object like the others and a method that returns an ActionListener type implementation class object. ▪ It is called by the view package to process specific requests categorized by a string in the createCommand method.

Class Name: SaveDocument	
Responsibilities:	**Collaborations:**
▪ The SaveDocument class has 2 methods and 1 constructor. ▪ The saveDoc method takes as argument an object which, is the current object handled by the gui, and saves it using a Jfilechooser alongside an output file/objectstream. ▪ The actionPerformed method is inherited by the interface ActionListener and is not currently in any use.	▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for saving the current document that's in the GUI. ▪ It holds a field object of Document type.

Class Name: TuneAudio	
Responsibilities:	**Collaborations:**
▪ The TuneAudio class has 1 constructor that is not currently in use.	▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for tuning the audiomanager object held by document responsible for voicing text.

▪ And an actionPerformed method inherited from ActionListener interface which is responsible for setting the volume, pitch or speech rate of the voice the current Document is using.	▪ It holds a field object of Document type.

Class Name: DocumentToSpeech	
Responsibilities:	**Collaborations:**
▪ The DocumentToSpeech class has 1 constructor that is not currently in use. ▪ And an actionPerformed method inherited from ActionListener interface which is responsible for tranforming the contents of a document using the voice of the audiomanager object held by the document.	▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for transforming the contents of a document to speech. ▪ It holds a field object of Document type.

Class Name: NewDocument	
Responsibilities:	**Collaborations:**
▪ This class creates a new Document object.	▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for the creation of a new Document. ▪ It holds a field object of Document type.

Class Name: OpenDocument	
Responsibilities:	**Collaborations:**
▪ The OpenDocument class has 2 methods and 1 constructor.	▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for opening a document that’s saved in the disk.

▪ The openDocument method takes as argument a String which, is acquired by a jFileChooser in the GUI class TextEditor and passed on to this one. Then the class opens an input stream for file and object. Finally sets the current Document to be the one read by the disk. ▪ The actionPerformed method is inherited by the interface ActionListener and is not currently in any use.	▪ It's method openDocument returns a new Document object that is read from the disk and returned to the front end for edit, transform etc.

Class Name: EditDocument	
Responsibilities: ▪ This class has 3 methods and 1 constructor method. ▪ The actionPerformed method is inherited by the interface ActionListener and is not in use. ▪ The main role of this class is to Edit the Contents of the document held by the gui. When asked by the user this class return the new String after the edit is done using the inputs the user gave the application.	**Collaborations:** ▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for Editing the contents of the current document held by the gui. ▪ Both replaceAll and replaceSelection methods return a new String of the modified contents.

Class Name: LineToSpeech	
Responsibilities: ▪ This class has 1 constructor that is not in use. ▪ The actionPerformed method, which is inherited by the ActionListener interface, is called when the user wants to transform a single line to speech and not the whole document.	**Collaborations:** ▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for transforming the line of the document to speech. ▪ It holds a field object of Document type and an int type field for the line number.

Class Name: TuneEncoding	
Responsibilities: ▪ This class has 1 constructor that is not in use. ▪ The actionPerformed method which is inherited by the ActionListener Interface is called when the user wants to encode and trasnform the contents of a Document or a line of them.	**Collaborations:** ▪ This class is an implementation of the ActionListener Interface. ▪ It is called by the CommandsFactory class for encoding and transforming the contents or a line of the document to speech. ▪ This class has a current Document object field, a string field determining the type of encoding, an int line number field and finally a StrategiesFactory object used for creating the corresponding encodingStrategy.

Class Name: ReplayCommand (not functional)	
Responsibilities: ▪ ---	**Collaborations:** ▪ --

Class Name: ReplayManager (not functional)	
Responsibilities: ▪ ---	**Collaborations:** ▪ --

encodingstrategies PACKAGE CLASSES:

Interface Name: EncodingStrategies	
Responsibilities: ▪ The main interface of encoding is responsible for the implementation of the method encode of a string.	**Collaborations:** ▪ Used by Document and Line classes of model package. ▪ Implemented by an abstract class TemplateEncoding. Which is extended by the 2 concrete classes AtBashEncoding and Rot13Encoding.

Class Name: AtBashEncoding	
Responsibilities:	**Collaborations:**
▪ Holds the method for encoding a String based on the AtBash encoding.	▪ Is an extended implementation of the Interface EncodingStrategy.

Class Name: Rot13Encoding	
Responsibilities:	**Collaborations:**
▪ Holds the method for encoding a String based on the Rot13 encoding.	▪ Is an extended implementation of the Interface EncodingStrategy.

Class Name: StrategiesFactory	
Responsibilities:	**Collaborations:**
▪ The main role of this class is to create an object type of EncodingStrategy and bases on a String determines which implementation of it will be created and returned.	▪ This class is using an EncodingStrategy type object. ▪ It is used by TuneEncoding class of the commands package to determine which implementation of the interface will be used (or which encode strategy)

Abstract Class Name: TemplateEncoding	
Responsibilities:	**Collaborations:**
▪ This class contains a constructor not in use and 2 abstract methods, encode and mapCharacter.	▪ This class is extended by the AtBashEncoding and Rot13Encoding classes which inherit the 2 abstract methods. ▪ Also this class implements the Innterface EncodingStrategy.

text2speechapis PACKAGE CLASSES:

Interface Name: TextToSpeechAPI	
Responsibilities:	**Collaborations:**

▪ This interface’s method play is responsible for the speaking of the contents for the application ▪ Also it has tuneValues for the Volume, Speech and Speech rate of the voice that speaks the contents of the document or the line.	▪ This interface is implemented by the FakeTextToSpeechAPI and the FreeTTSAdapter.

Class Name: FakeTextToSpeechAPI	
Responsibilities:	**Collaborations:**
▪ This class was created for pure testing purposes. ▪ Its a pseudo implementation of the TextToSpeechAPI.	▪ It is used by some tests for checking if the setting of the voice (volume, pitch and rate) are changed and set correctly. Without actually transforming the voice to these values.

Class Name: FreeTTSAdapter	
Responsibilities:	**Collaborations:**
▪ This class is an implementation of the TextToSpeechAPI interface. ▪ It’s main role is to play the audio from the String that has been sent from the Document and Line classes. ▪ Also it can change the settings for the voice speaking (I.e volume speech rate and pitch).	▪ LineToSpeech and DocumentToSpeech sents a request to the current document to play the audio of contents (whole or line) through the Document or Line classes to this one. ▪ Also the TuneAudio class of the commands package is the one changing the settings of the voice kept inside this class.

3.3 Test Cases

Below will be presented the test cases mentioned before with greater detail.

Test Case Id : TU01
Test Priority : high
Module Name : Text Editor Document Creation command
Test Title : testNew
Description: Test the NewDocument ActionCommand for creating a new Document object.
Pre-Conditions:
Dependencies:

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Issue and execute a NewDocument command	te.actionPerformed(e);	Tester user must press ok in both input for title and author.	Document is created.		
2	Create an empty Document for comparison	emptyDocument				
3	Check if current Document held by TextEditor is not Empty				Pass	
4						

Post-Conditions: Tester has pressed ok in both input dialogs.

Test Case Id : TU02
Test Priority : high

Module Name : Text Editor Edit Document command
Test Title : testEdit
Description: Test the EditDocument ActionCommand for a document.
Pre-Conditions:
Dependencies:

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Set a current Document's contents in the gui with a set string of the alphabet.	String is equal to the alphabet in lowercase letters. "a b c d … x y z"				
2	Issues and executes an EditDocument command.	2 inputs request the tester for a word to replace and the word to be replaced with.	Tester should input a lowercase letter in the first input and w/e he wants on the second.	The document should be edited according the inputs of the tester.		
3	Check if current Document held by TextEditor is different from the string set in the beginning				Pass	

Post-Conditions: Tester has inputted both dialogs correctly.

Test Case Id : TU03
Test Priority : high

Module Name : Text Editor Save Document command
Test Title : testSave
Description: Test the SaveDocument ActionCommand for saving a Document to the disk.

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Set a new Document on the gui with contents equal to a string.	String = “sample text”		Document is set with contents.		
2	Issue a saveDocument command on the gui. The tester is asked to save the testing document somewhere in the disk.	A save document dialog is opened asking for the user to enter a path on the disk for the testing document.	User gives a valid path to the test.	Document is saved.		

3	The filepath is saved in a variable for comparison with getSavedDoc(String filepath) method to get the saved Document.	Filepath = newDoc.filePath;
		getSavedDoc(filepath).getContents()				

4	The test checks if the current documents contents match with the saved Document.				Pass	

Test Case Id : TU04
Test Priority : high
Module Name : Text Editor Open Document command
Test Title : testOpen

Description: Test the OpenDocument ActionCommand for opening a saved Document from the disk.
Pre-Conditions:
Dependencies:

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Issue and execute a OpenDocument command	User must enter a valid Document object that has been saved earlier. I.e by the testSave	Document is read from the file the user entered.	Document is read from the disk		
2	Check if the contents of the current Document object match the contents of the Document that was opened earlier.		Contents must match.	Contents match.	Pass	
3						
4						

Post-Conditions: Tester has opened a valid Document object.

Test Case Id : TU05
Test Priority : high
Module Name : Text Editor Transform Contents command
Test Title : testTTSdoc
Description: Test the DocumentToSpeech ActionCommand for contents to speech transformation.

Pre-Conditions:
Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String.	String s1 = “this is yet another sample text”	Document is set with FakeTTSapi and with sample contents.	checks		
2	Perform the DocumentToSpeech command for the current document using FakeTTSAPI.					
3	Check if the contents of the current Document match the field toAudio of the FakeTTSAPI that got commanded.	FakeTTSAPI.toAudio == getContents();	True	match	Pass	
4						

Post-Conditions:

Test Case Id : TU06
Test Priority : high
Module Name : Text Editor Transform Line command
Test Title : testLineTTS
Description: Test the LineToSpeech ActionCommand for line transform to speech.

Pre-Conditions:	
Dependencies: text2speechapis.FakeTextToSpeechAPI	

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String. Also a line number is being set.	String s21+s22+s23 = “this is yet\n another sample\n text”; int fakeLine = 2;	Document is set with FakeTTSapi and with sample contents alongside the fakeline.	checks		
2	Perform the LineToSpeech command for the current document’s (fake)line using its FakeTTSAPI obj.					
3	Check if the contents of the current Document’s 2nd line match the field toAudio of the FakeTTSAPI that got commanded.	FakeTTSAPI.toAudio == getLine(fakeLine -1);	True	match	Pass	

Post-Conditions:

Test Case Id : TU07
Test Priority : high
Module Name : Text Editor Reverse Transform Contents command
Test Title : testReverseTTS
Description: Test the (reverse)DocumentToSpeech ActionCommand for Document’s contents

Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String.	String s3 = “stay home and stay\nsafe”;	Document is set with FakeTTSapi and with sample contents .	checks		
2	Perform the (reverse)DocumentToSpeech command for the current document using its FakeTTSAPI obj.					
3	Check if the reverse contents of the current Document match the field toAudio of the FakeTTSAPI that got commanded.	FakeTTSAPI.toAudio == (reverse)getContentsAsLine();	True	match	Pass	

Test Case Id : TU08
Test Priority : high
Module Name : Text Editor Reverse Transform Line command
Test Title : testReverseLineTTS
Description: Test the (reverse)LineToSpeech ActionCommand for reverse transfrmation of a line of the current document.

Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String. Also a fake line int for choosing a line to reverse transform.	String s31 = “stay home\n”; String s32 = “and stay\n”; String s33=“safe”; contents = s31+s32+s33; fakeLine = 1 ;(1st line)	Document is set with FakeTTSapi and with sample contents alongside the fakeline.	checks		
2	Perform the (reverse)LineToSpeech command for the current document’s 1st line using its FakeTTSAPI obj.					
3	Check if the reverse contents of the current Document’s 1st line, match the field toAudio of the FakeTTSAPI that got commanded.	FakeTTSAPI.toAudio == (reverse)getContentsAsLine();	True	match	Pass	

Test Case Id : TU09
Test Priority : high
Module Name : Text Editor Encode and transform contents command
Test Title : testEncode
Description: Test the TuneEncoding ActionCommand for Document’s contents
Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String.	String s4 = “stay and stay\nsafe”;	Document is set with FakeTTSapi and with sample contents.	checks		
2	Perform the atbash encode using the TuneEncoding command for the current document using its FakeTTSAPI obj.					
3	Check if the atbash encoded contents of the current Document match the field toAudio of the FakeTTSAPI.	FakeTTSAPI.toAudio == getEncodedContents();	True	match	Pass	
4	Perform the Rot13 encode using the TuneEncoding command for the current document using its FakeTTSAPI obj.					
5	Check if the Rot13 encoded contents of the current Document match the field toAudio of the FakeTTSAPI.	FakeTTSAPI.toAudio == getEncodedContents();	True	match	Pass	

Test Case Id : TU010
Test Priority : high
Module Name : Text Editor Encode and transform line of contents command
Test Title : testLineEncode
Description: Test the TuneEncoding ActionCommand for a line of the current document.
Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Create and set the gui to a Document, that uses the FakeTTSAPI, to a String. Also a fakeLine int variable for choosing a line.	String s5 = “I gained15\nkilos during\nthis\nquarantine”; fakeLine = 1;	Document is set with FakeTTSapi and with sample contents alongside with fakeLine.	checks		
2	Perform the atbash encode using the TuneEncoding command for the current document’s 1st line using its FakeTTSAPI obj.					
3	Check if the atbash encoded 1st line of the current Document match the field toAudio of the FakeTTSAPI.	FakeTTSAPI.toAudio == getEncodedLine(fakeLine - 1);	True	match	Pass	
4	Perform the Rot13 encode using the TuneEncoding command for the 1st line of the current					

	document using its FakeTTSAPI obj.					
5	Check if the Rot13 encoded 1st line of the current Document match the field toAudio of the FakeTTSAPI.	FakeTTSAPI.toAudio == getEncodedLine(fakeline - 1);	True	match	Pass	

Test Case Id : TU11
Test Priority : high
Module Name : Text Editor tune volume, pitch or speech rate of voice command
Test Title : testVolume testPitch testRate
Description: Test the TuneAudio ActionCommand for changing the volume, pitch and speech rate of the voice that speaks the current document.
Dependencies: text2speechapis.FakeTextToSpeechAPI

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	TestVolume creates and set a new Document object with FakeTTSAPI and a fakeVol value. Then performs the tuneAudio and checks the values of FakeTTS with the fakeVol value.	Document.fakeVol = 0.344f;	Document fakeVol must match with the its FakeTTSAPI volume value.	match	Pass	
2	TestPitch with fakePitch value performs the tuneAudio and checks the values of FakeTTS with the fakePitch value.	Document.fakeVol = 230.44f;	Document fakePitch must match with the its FakeTTSAPI pitch value.	match	Pass	
3	TestRate with fakeRate value performs the tuneAudio and checks the values of FakeTTS with the fakeRate value.	Document.fakeVol = 358.28f;	Document fakeRate must match with the its FakeTTSAPI rate value.	match	Pass	

Test Case Id : TU12
Test Priority : high
Module Name : Text Editor tune encoding command
Test Title : testChangeEncode
Description: Test the TuneEncoding ActionCommand for changing the encoding of the method of the current document.
Dependencies:

Step	Test Steps	Test Data	Expected Results	Actual Results	Status	Notes
1	Creates a new document and sets its audio.					
2	Performs an Atbash encode at the document, setting its encode method to atbash.					
3	Saves the current encoding strategy of the document to a variable for later checking.	EncodingStrategy first = currentDocument.encodingStrategy;	First now has the encoding strategy of currentDocument	<-	Pass	
4	Performs an Rot13 encode at the current document, changing its encode to Rot13 type encode.					
5	Checks if the first encoding strategy and the current encoding of the document are different from each other	first != currentDocument.encodingStrategy;	The encoding is changed	The encoding has changed	Pass	