UNIVERSITY OF ENGINEERING AND TECHNOLOGY - VNU
FACULTY OF INFORMATION TECHNOLOGY

INT2204 - Object Oriented Programming: Project Report

Submitted by:

Hoang Bao An
Nguyen Duc Huy
Nguyen Anh Duc

Under the supervision of:
PhD To Van Khanh
MSc Nguyen Thu Trang

Hanoi, 2023

Abstract

Object-oriented programming (OOP) stands as a predominant programming paradigm widely employed in software development. This paradigm empowers programmers to organize their programs into modular and reusable components, akin to blueprints. Over the duration of this course, our endeavors have culminated in the creation of a Dictionary program in Java, a language highly regarded for its adherence to the principles of OOP. The following report has been meticulously crafted to elucidate and present the knowledge achieved during this period.

Table of Contents

List of Figures

Part 1

Overview

The primary aim of the Dictionary program is to afford users the capability to translate words or phrases from English to Vietnamese. Additionally, we have incorporated a mini word game named "Wordle" into the program. Adhering to the specified requirements, we have designed two principal interfaces for user interaction: the command line interface and the graphical interface.

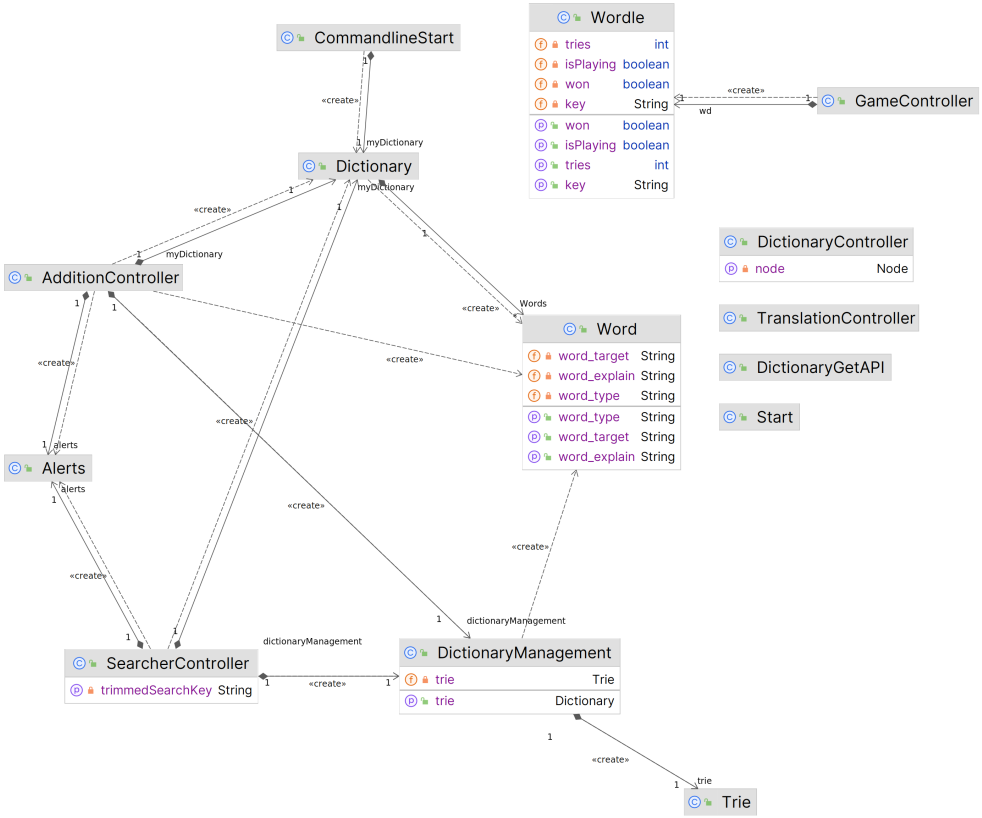

Figure 1.1: Program structure

Part 2

The command line version

The Dictionary program will consists of these main classes:

- **Word** - This class has been developed with the specific purpose of encapsulating a singular English word or phrase and its corresponding Vietnamese meaning.
- **Dictionary** - Serving as the pivotal component at the interface level of the program, this class encompasses essential functionalities, including the addition and removal of words, as well as word retrieval. The latter is facilitated by the **Trie** class.
- **DictionaryManagement** - This class is instrumental in furnishing utilities to ensure the proper functioning of the Dictionary class. Noteworthy functions include retrieving words from a file, housing an extensive repository of over $100,000$ words, and implementing various functions integral to the Dictionary class.

2.1 The Word object

This object contains three mains properties:

- **word_target (String):** Contains the word or phrase in English.
- **word_explain (String):** Contains the translation in Vietnamese.
- *word_type (String):* Originally intended to store the part of speech, but in our dataset, word explanations already encompass both the meaning and the part of speech. Consequently, this property is underutilized compared to its predecessors.

Also we provided some basic getter and setter methods for each property, and a hashing function (including all three fields, inherited from String hashing function from Java) along with a comparison function uses which will discussed soon.

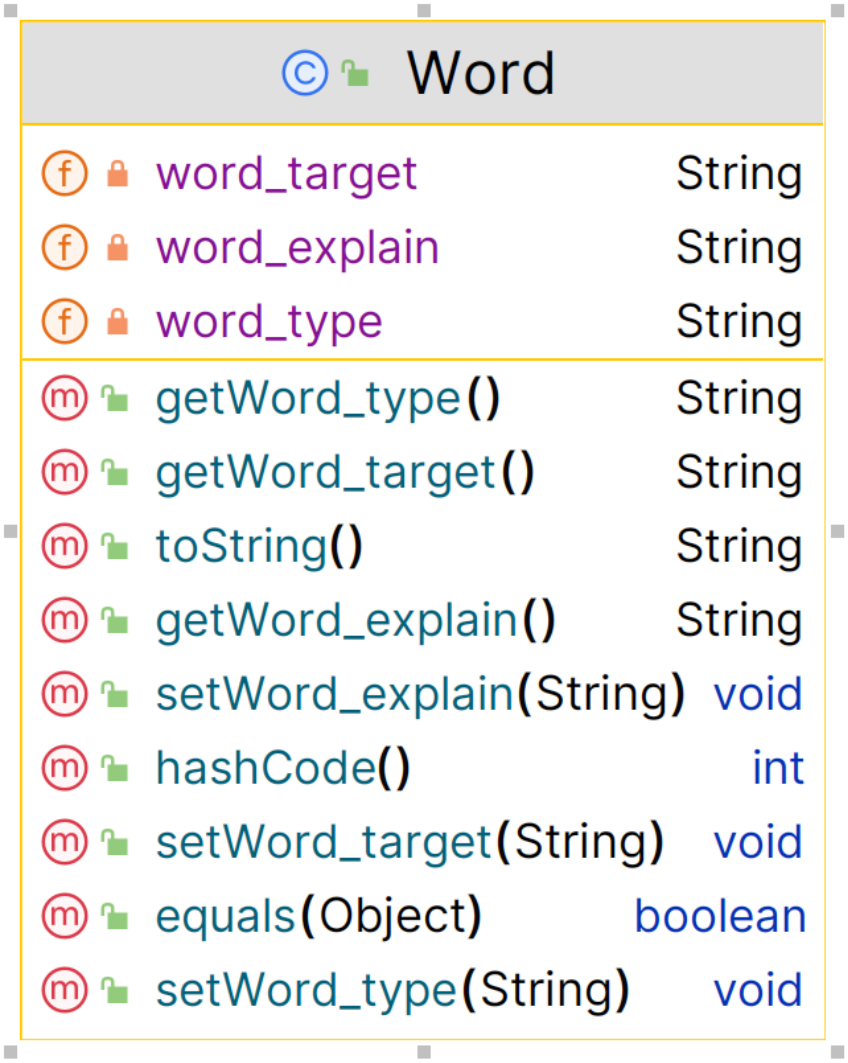

Figure 2.1: Diagram of the Word class

2.2 The Dictionary Object

Words are maintained within a Java Hash Set, prompting the implementation of a hashing function within the Word class. Regrettably, a lapse in communication among team members resulted in the removal function being implemented through a linear search within the sets. An optimization opportunity exists by exclusively hashing the word using its *word_target* property and subsequently conducting searches based on the hashed value.

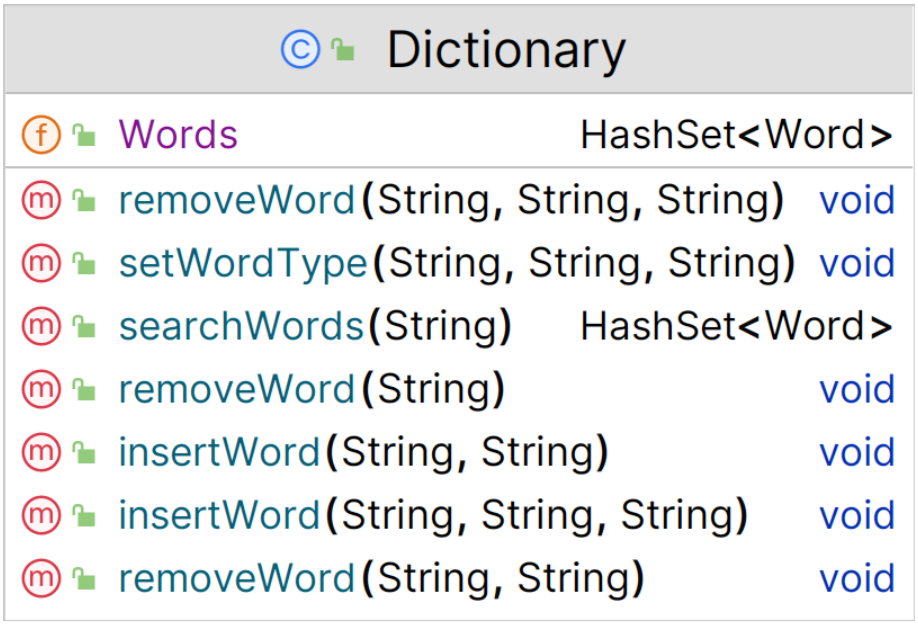

Figure 2.2: Diagram of the Dictionary class

2.3 The DictionaryManagement class

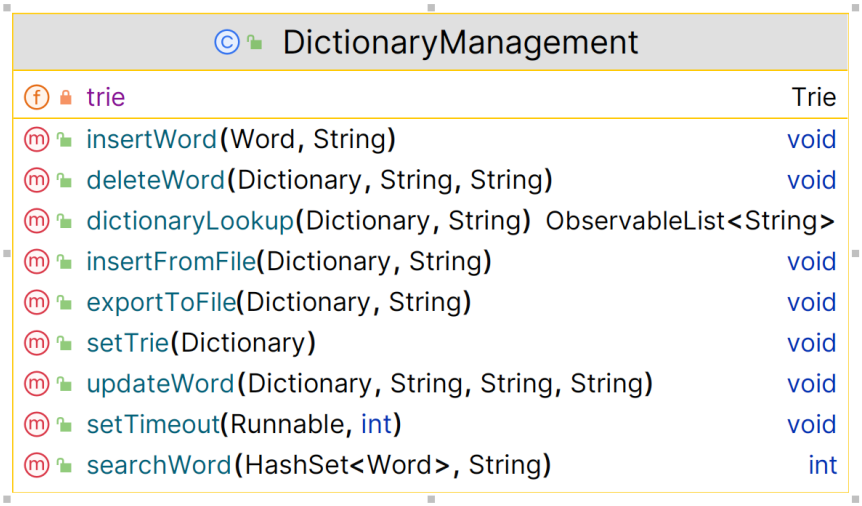

Figure 2.3: Diagram of the DictionaryManagement class

This class is made for some input and output operations,

- **insertWord:** Take in a Word object and a path to file where we're storing these words. In this file, each words are separated by a '|' character.

```
public void insertWord(Word word, String path) {
    try (FileWriter fileWriter = new FileWriter(path, true);
         BufferedWriter buffer = new BufferedWriter(fileWriter)) {

        buffer.write("|" + word.getWord_target() + "\n"
            + word.getWord_explain());
        buffer.newLine();

    } catch (IOException e) {
        System.out.println("IOException.");
    } catch (NullPointerException e) {
        System.out.println("Null Exception.");
    }
}
```

- **insertFromFile:** The method takes in 2 parameters: a Dictionary object which you wanted to insert the word in, and the other is the path to the dictionary text file. We used a BufferedReader object to read the content which is loaded from the text file

from FileReader object(which we feed it the file path needed).Then, a while loop is used to iterate through the file to the end. For each iteration, we will initialize an empty Word object and we set the English word or phrase based on the trimmed content of the current line. Next, we accumulate the Word object meaning by a StringBuilder object by appending the next lines until the '|' is encountered (because this indicates that it's the end of the word and after that is another new word).

```
public void insertFromFile(Dictionary myDictionary, String path) {
    try {
        FileReader fileReader = new FileReader(path);
        BufferedReader buffer = new BufferedReader(fileReader);

        String line;
        while ((line = buffer.readLine()) != null) {
            Word words = new Word();
            words.setWord_target(line.trim());

            StringBuilder meaning = new StringBuilder();
            while ((line = buffer.readLine()) != null) {
                if (!line.startsWith("|")) {
                    meaning.append(line).append("\n");
                } else {
                    words.setWord_explain(meaning.toString().trim());
                    myDictionary.insertWord(words.getWord_target(),
                    words.getWord_explain());
                    words.setWord_target(line.replace("|", "").trim());
                    meaning = new StringBuilder();
                }
            }

            words.setWord_explain(meaning.toString().trim());
            myDictionary.insertWord(words.getWord_target(),
            words.getWord_explain());
        }

        buffer.close();
    } catch (IOException e) {
        System.out.println("An error occurred with the file: " + e);
    } catch (Exception e) {
        System.out.println("Something went wrong: " + e);
    }
}
```

- **exportToFile:** This function write all the words from the Dictionary object that was taken in the parameter, and again, the other parameter is the path to the file that you

wanted to write the words in. The format of the file maintain similar to that in the *insertWord* method, and the logic is also not that much different from it.

- **updateWord, deleteWord and insertWord** These functions are made so that users can update and delete words that exists in the Dictionary. The similar logic are used. We'll first check if the input word exists in the dictionary, if it does, we do the relative operation to the Dictionary object, and the re-write the dictionary file using the *exportToFile* method, otherwise we will throw them an error.

```
public void updateWord(Dictionary myDictionary, String
        wordToUpdate, String meaning, String path) {
    try {
        boolean found = false;
        for (Word word : myDictionary.Words) {
            if (word.getWord_target().equals(wordToUpdate)) {
                word.setWord_explain(meaning);
                found = true;
                break;
            }
        }

        if (found) {
            exportToFile(myDictionary, path);
        } else {
            System.out.println("Word not found. Unable to update.");
        }
    } catch (NullPointerException e) {
        System.out.println("Null Exception.");
    }
}
```

2.4 Trie

The **Trie** class is designed to facilitate the efficient storage and retrieval of English words. Specifically, it provides a tree structure in which each node represents a character, and words are constructed through edges from the root to the leaf nodes.

- **insert:** This method inserts a word into the trie. It traverses each character of the word, building trie nodes corresponding to those characters. Ultimately, the last node of the word is marked as an end-of-word node

```
public void insert(String word) {
    if (word == null) {
        throw new IllegalArgumentException
        ("Null word entries are not valid.");
```

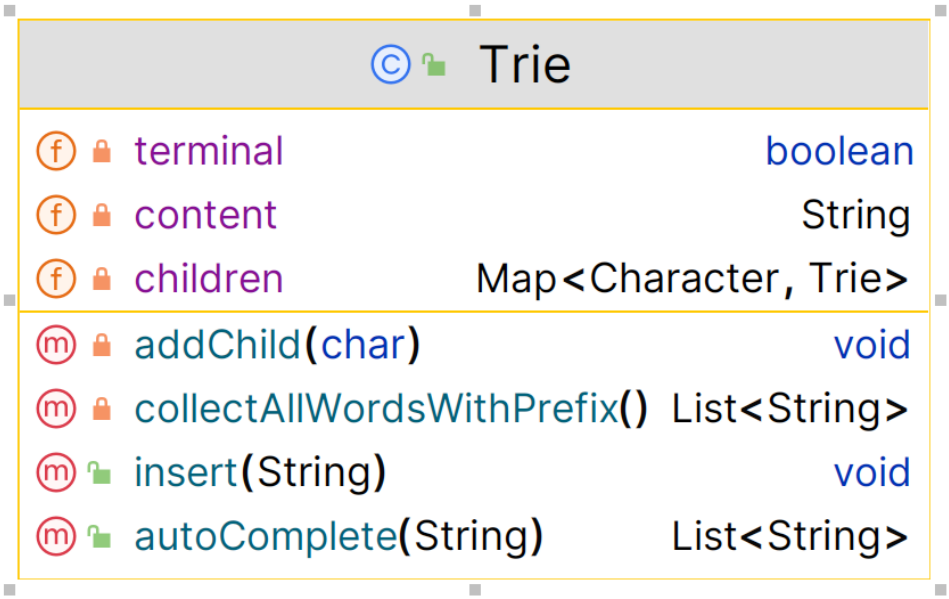

Figure 2.4: Diagram of the Trie class

```
    }

    Trie node = this;
    for (char c : word.toCharArray()) {
        if (!node.children.containsKey(c)) {
            node.addChild(c);
        }
        node = node.children.get(c);
    }
    node.terminal = true;
}
```

- **autoComplete:** This method traverses the nodes of the trie based on the characters of the input prefix. It searches and returns a list of words with the specified prefix. It traverses the trie from the root to the last node of the prefix and then uses a method called **collectAllWordsWithPrefix** to gather all words with the given prefix. If no node corresponding to a character in the prefix is found, it returns null, indicating that there are no matching words for the prefix.

```
public List<String> autoComplete(String prefix) {
    Trie trieNode = this;
    for (char c : prefix.toCharArray()) {
```

```
            if (!trieNode.children.containsKey(c)) {
                return null;
            }
            trieNode = trieNode.children.get(c);
        }
        return trieNode.collectAllWordsWithPrefix();
    }
```

- **collectAllWordsWithPrefix:** This method is called from the current trie node to collect all words with prefixes starting from that node. It checks whether the current node is an end-of-word node. If so, it adds the word to the result list. Then, it recursively calls itself on all child nodes to collect each result list from the entire trie

```
    private List<String> collectAllWordsWithPrefix() {
        List<String> wordResults = new ArrayList<>();
        if (this.terminal) {
            wordResults.add(this.content);
        }
        for (Map.Entry<Character, Trie> entry : children.entrySet()) {
            Trie child = entry.getValue();
            Collection<String> childWords
                = child.collectAllWordsWithPrefix();
            wordResults.addAll(childWords);
        }
        return wordResults;
    }
```

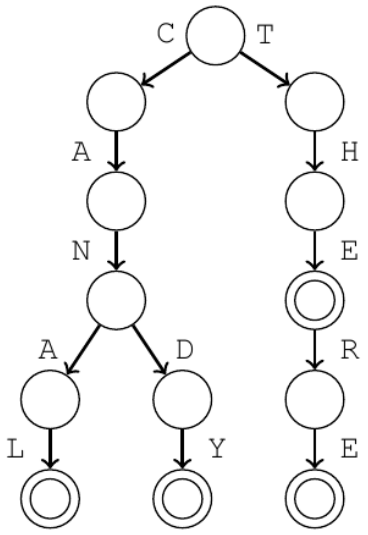

Figure 2.5: A trie that contains the strings CANAL, CANDY, THE, and THERE

These primary functionalities will be employed by the **CommandLineStart** class, responsible for managing user input through the **Scanner** class provided by Java. This class also facilitates the presentation of textual content and menus. The implementations have been kept straightforward and accessible for ease of comprehension. Although the source code is lengthy, its simplicity enhances readability. For a more in-depth examination, the code is available on our GitHub repository, and you are encouraged to explore it at your convenience.

There's also and English in depth translation being built, using SQLite database and a dictionary API made by meetDeveloper. All the functions to retrieve data using JDBC driver ran with no issues, however wasn't used in this project, although it brought us somewhat experience to use the JDBC driver for database querying inside Java.

Part 3

The GUI version

In the GUI version implemented by JavaFX, we provided user 3 main function:

- **Searching, adding, altering and deleting:** user will search for words that was retrieved from a text file mentioned in the preceding part.
- **English - Vietnamese translation using API:** this function allows user to translate their input either from English to Vietnamese or other way around.
- **Wordle game:** This is a clone of the famous Wordle from The New York Times

3.1 Word searcher, altering word and deleting word

:

Figure 3.1: Main scene

This scene is controlled by **SearcherController**. We provided user with a search bar (a

TextField from JavaFX), on the top-right of the scene. On each key that the user typed, we used a method called **handleOnKeyTyped** to clear the list of words below the search bar and updated it with the new key. The **observableArrayList** method is implemented in the DictionaryManagement class and was used to get all word with the prefix of user's input. The result will be rendered out to the recommendation list and if user click on the word, the **handleMouseClickAWord** even handler will be used to render out the key and word to the text fields on the left side of the window.

```
@FXML
private void handleOnKeyTyped() {
    clearListAndSearchKey();
    String searchKey = getTrimmedSearchKey();
    updateResultList(searchKey);
    updateResultsViewAndAlert();
}
private void updateResultsListView() {
    resultsListView.setItems(FXCollections.observableArrayList(list));
}

// In DictionaryManagement.java
public ObservableList<String> dictionaryLookup(Dictionary myDictionary,
                                String key) {
    ObservableList<String> list = FXCollections.observableArrayList();
    try {
        List<String> results = trie.autoComplete(key);
        if (results != null) {
            int length = Math.min(results.size(), 15);
            for (int i = 0; i < length; i++) {
                list.add(results.get(i));
            }

            for (Word word : myDictionary.Words) {
                if (word.getWord_target().equalsIgnoreCase(key)) {
                    list.add("Meaning in Vietnamese: "
                        + word.getWord_explain());
                    break;
                }
            }
        }
    } catch (Exception e) {
        System.out.println("Something went wrong: " + e);
    }
    return list;
}
```

The two button below the text-field that contains word meanings are buttons for:

- The left one: For *word configuration* If user click on it, a window will prompted out and ask if user wanted to configure that word, the text field that currently containing the meaning of the word will become editable. After your configurations, you have to click on that green tick on the top-right of the text field to save your alternation.

Figure 3.2: Main scene when editing a word meaning

- The right one: For *deleting a word*. If user click on it, a window also prompted out and ask if you really wanted to delete that word, if yes, then that word will be deleted from the Dictionary.

There are two methods, namely **handleClickSoundSourceLanguage** and **handleClick-SoundTargetLanguage**, designed to handle events when users wish to pronounce text in the source and target languages, respectively. The utilization of the Text-to-Speech service provided by VoiceRSS is employed to convert text into speech and articulate it. This will be discussed further in the latter section.

3.2 Adding word to the Dictionary

We provide a separate window for user to define their own word. The upper text field is used for the word key, and the field below is for definition. Both are of course, editable.

After adding your own word, the word will be added into the Dictionary. When you come into the main scene, if you search the word, it will show up.

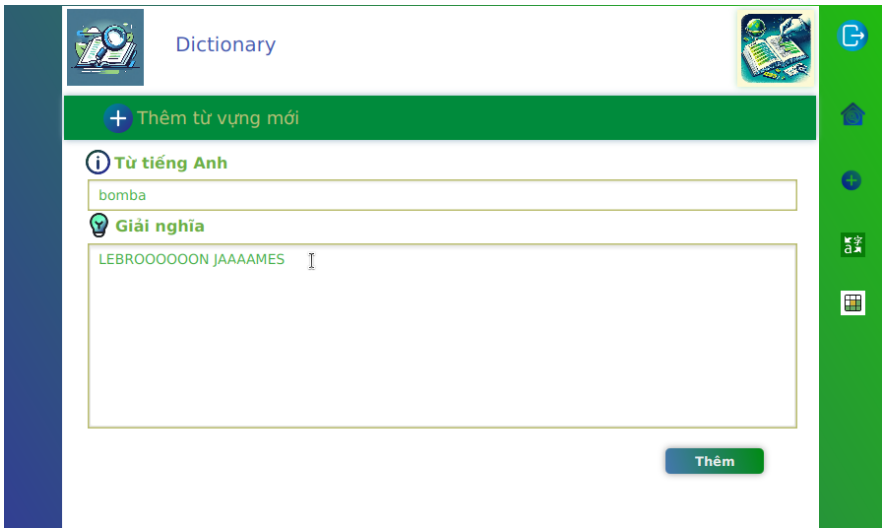

Figure 3.3: Window when adding a word in a dictionary

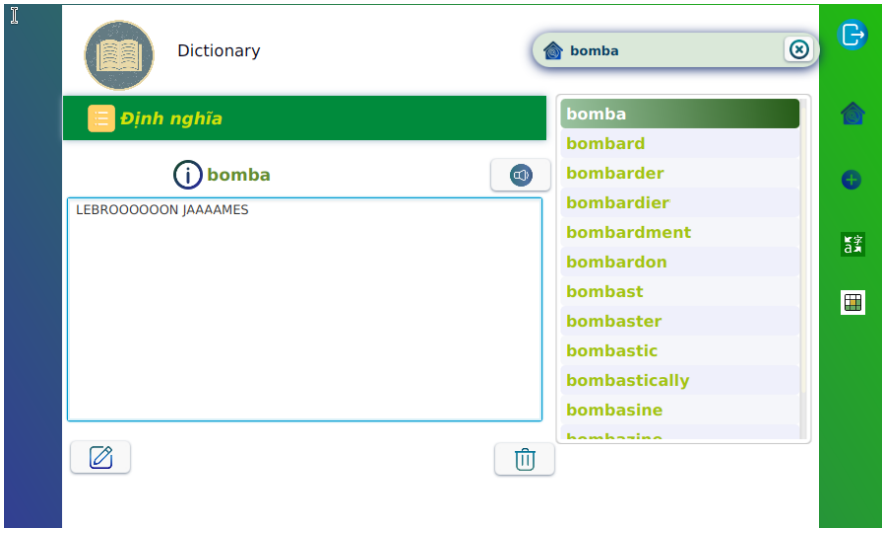

Figure 3.4: User defined word showing up on main scene

3.3 Translation API and Text-to-Speech (TTS)

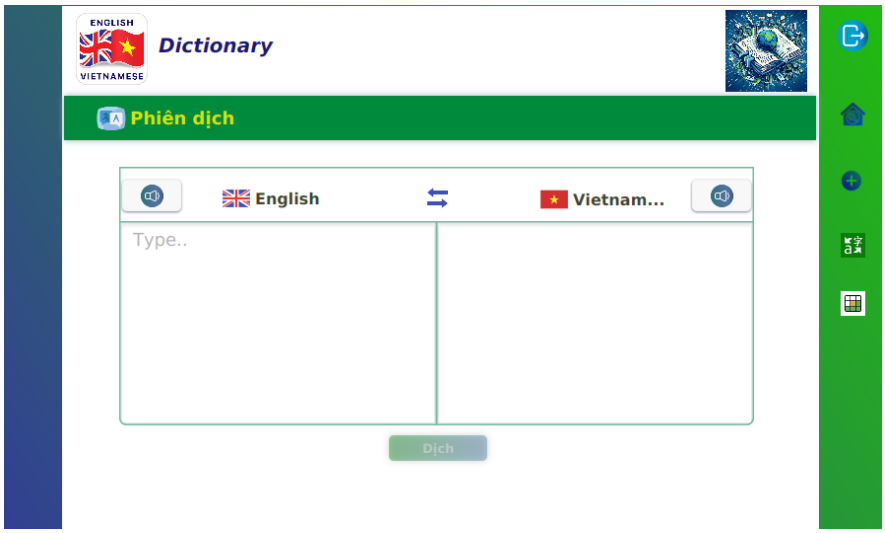

Figure 3.5: English - Vietnamese translation scene

API Keys and URLs:

Utilizes API keys (**TRANSLATE_KEY**, **rapidKey**, **voiceRSSKey**) and URLs for services such as Microsoft Translator and VoiceRSS.

Translation Event Handling:

When the user presses the translate button, the **handleOnClickTranslate** method is invoked. This code sends an HTTP request to the Microsoft Translator translation API and displays the translation result in the target language input field (**targetLanguageField**).

```
@FXML
private void handleOnClickTranslate() throws IOException, InterruptedException {
    String rootAPI;
    if (isTargetedToVietnameseLanguage) {
        rootAPI = "https://microsoft-translator-text.p.rapidapi.com/translate?api-
        version=3.0&to%5B0%5D=vi&textType=plain&profanityAction=NoAction&from=en";
        sourceLanguage = "en";
        targetLanguage = "vi";
        sourceSpeechLanguage = "en-us";
        targetSpeechLanguage = "vi-vn";
    } else {
        rootAPI = "https://microsoft-translator-text.p.rapidapi.com/translate?api-
        version=3.0&to%5B0%5D=en&textType=plain&profanityAction=NoAction&from=vi";
        sourceLanguage = "vi";
        targetLanguage = "en";
        sourceSpeechLanguage = "vi-vn";
```

```
        targetSpeechLanguage = "en-us";
    }

    String srcText = sourceLanguageField.getText();

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(rootAPI))
            .header("content-type", "application/json")
            .header("X-RapidAPI-Key", TRANSLATE_KEY)
            .header("X-RapidAPI-Host", "microsoft-translator-text.p.rapidapi.com")
            .POST(HttpRequest.BodyPublishers.ofString("[\r\n
            {\r\n        \"Text\": \"" + srcText + "\"\r\n    }\r\n]"))
            .build();

    HttpResponse<String> response = HttpClient.newHttpClient().send(request,
    HttpResponse.BodyHandlers.ofString());

    String result = response.body();
    JSONArray jsonArray = new JSONArray(result);
    JSONObject jsonObject = jsonArray.getJSONObject(0);
    JSONArray translationsArray = jsonObject.getJSONArray("translations");
    JSONObject translationsObject = translationsArray.getJSONObject(0);
    String trans = translationsObject.getString("text");

    targetLanguageField.setText(trans);
}
```

Switching Between Source and Target Languages:

The event of switching between source and target languages is handled by the **handleSwitchLanguageToggle** method. It adjusts the interface and updates relevant language parameters.

```
@FXML
private void handleSwitchLanguageToggle() {
    sourceLanguageField.clear();
    targetLanguageField.clear();
    if (isTargetedToVietnameseLanguage) {
        englishLabel.setLayoutX(426);
        vietnameseLabel.setLayoutX(104);
        sourceLanguage = "vi";
        targetLanguage = "en";
        sourceSpeechLanguage = "vi-vn";
        targetSpeechLanguage = "en-us";
    } else {
        englishLabel.setLayoutX(100);
```

```
        vietnameseLabel.setLayoutX(426);
        sourceLanguage = "en";
        targetLanguage = "vi";
        sourceSpeechLanguage = "en-us";
        targetSpeechLanguage = "vi-vn";

    }
    isTargetedToVietnameseLanguage = !isTargetedToVietnameseLanguage;
}
```

Pronunciation:

There are two methods, namely **handleClickSoundSourceLanguage** and **handleClick-SoundTargetLanguage**, designed to handle events when users wish to pronounce text in the source and target languages. The VoiceRSS Text-to-Speech service is used to convert text into speech and articulate it.

```
@FXML
private void handleClickSoundSourceLanguage() {

    String language;
    String textToSpeak;
    textToSpeak = sourceLanguageField.getText();
    if (!textToSpeak.isEmpty()) {
        speech(textToSpeak, sourceSpeechLanguage, 50);
    } else {
        // Handle the case where there is no text to speak
        System.out.println("No text to speak");
    }
}

@FXML
private void handleClickSoundTargetLanguage() {

    String language;
    String textToSpeak;
    textToSpeak = targetLanguageField.getText();
    if (!textToSpeak.isEmpty()) {
        speech(textToSpeak, targetSpeechLanguage, 50);
    } else {
        // Handle the case where there is no text to speak
        System.out.println("No text to speak");
    }
}
```

Speech Method:

- Utilizes the VoiceRSS API to convert text into speech and articulate it audibly.
- Sends an HTTP request to the API, processes the response, and then pronounces the speech using the JavaFX **MediaPlayer** object.

```
public static void speech(String text, String language, int speechRate) {
    speechRate = convertSpeedRate(50);
    String
        rapidKey = "a3c9ad3dbemshf0d0293637a2c13p1ec157jsn471a26834d72";
    String voiceRSSKey = "3a396a04478e45e9ba3d47021c359be4";
    String audioFormat = "wav";
    String audioFormatCode = "8khz_8bit_stereo";
    String encodedText;
    encodedText = URLEncoder.encode(text, StandardCharsets.UTF_8);

    HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create(
            "https://voicerss-text-to-speech.p.rapidapi.com/?key="
            + voiceRSSKey))
        .header("content-type", "application/x-www-form-urlencoded")
        .header("X-RapidAPI-Key", rapidKey)
        .header("X-RapidAPI-Host", "voicerss-text-to-speech.p.rapidapi.com")
        .method("POST", HttpRequest.BodyPublishers.ofString("f=" + audioFormatCode
                + "&c=" + audioFormat
                + "&r=" + speechRate
                + "&hl=" + language
                + "&src=" + encodedText))
        .build();
    HttpResponse<byte[]> response;
    try {
        response = HttpClient.newHttpClient().send(request,
        HttpResponse.BodyHandlers.ofByteArray());
    } catch (IOException | InterruptedException e) {
        System.out.println(e.getMessage());
        return;
    }

    try {
        Path tempFile = Files.createTempFile("audio", ".wav");
        Files.write(tempFile, response.body(), StandardOpenOption.CREATE);
        Media hit = new Media(tempFile.toUri().toString());
        MediaPlayer mediaPlayer = new MediaPlayer(hit);
        mediaPlayer.play();
    } catch (IOException e) {
```

```
        System.out.println(e.getMessage());
    }
}
```

Speech Rate Conversion:

The **convertSpeedRate** method is used to convert the speech rate from one format to another. This is necessary as Text-to-Speech services may require different formats for speech rates.

Additionally, in the Word Search Section:

handleClickSound Method:

- Called when the user presses the button to pronounce the selected word from the dictionary list.
- First, sets system properties to use speech from the FreeTTS library. The chosen voice is **"com.sun.speech.freetts.en.us.cmu_us_kal.KevinVoiceDirectory."**
- Retrieves the voice manager object from the FreeTTS library. Retrieves a list of available voices. Checks if any voices are available. If at least one voice is available, selects the first voice in the list and allocates resources for the voice.
- Then, iterates through the word list in the dictionary. If the selected word is found, uses the voice to pronounce the word and concludes the method.
- If the selected word is not found, throws an **IllegalStateException**. If no voices are available, throws an **IllegalStateException**.

```
@FXML
private void handleClickSound() {
    System.setProperty("freetts.voices",
    "com.sun.speech.freetts.en.us.cmu_us_kal.KevinVoiceDirectory");

    VoiceManager voiceManager = VoiceManager.getInstance();
    Voice[] voices = voiceManager.getVoices();

    if (voices.length > 0) {
        Voice voice = voices[0];
        voice.allocate();

        for (Word word : myDictionary.Words) {
            if (isWordSelected(word)) {
                voice.speak(word.getWord_target());
                return;
            }
        }
```

```
            throw new IllegalStateException
                ("Cannot find selected word in the dictionary.");
        } else {
            throw new IllegalStateException("No voices available.");
        }
    }
```

3.4 Mini Game: Wordle

This is a mini game where player were challenged to guess a word of 5 characters in 6 tries. With each tries, player will be able to see how their guess matches the answer. If it's yellow then the character DOES appear in the answer, but it's in the wrong position. If it's grey, then it's does NOT appear in the answer. If it's green, then the word is IN the RIGHT POSITION of the answer. In this report, we'll be calling this the *coordinate* of the character.

The game logic was easy to understand and implementations was simple.

We'll be tracking the number of tries using a variable *tries*, and the game playing state at two variable namely *isPlaying* and *won*. And of course, the game answer - which was selected randomly from a file - will be stored in *key*.

3.4.1 Get random key

A random key will be selected in a file, with each word lying in a seperate line. First, we'll be getting the number of line in the target file, using the **Files.lines(filePath).count()** method from java.nio.file.Files. Although generating a random number wouldn't be such a heavy task, we still used **ThreadLocalRandom** to generate a random number to separate it from the current running thread. After a number is desired, we'll take the content from the line of that number.

```
public static String getRandomLineFromFile(Path filePath) {
    long lineCount = 0;
    try {
        lineCount = Files.lines(filePath).count();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    // Generate a random line number
    long randomLineNumber = ThreadLocalRandom.current().nextLong
    (1, lineCount + 1);

    // Read the file and return the random line
    try (var lines = Files.lines(filePath)) {
        return lines.skip(randomLineNumber - 1).findFirst().orElse(null);
```

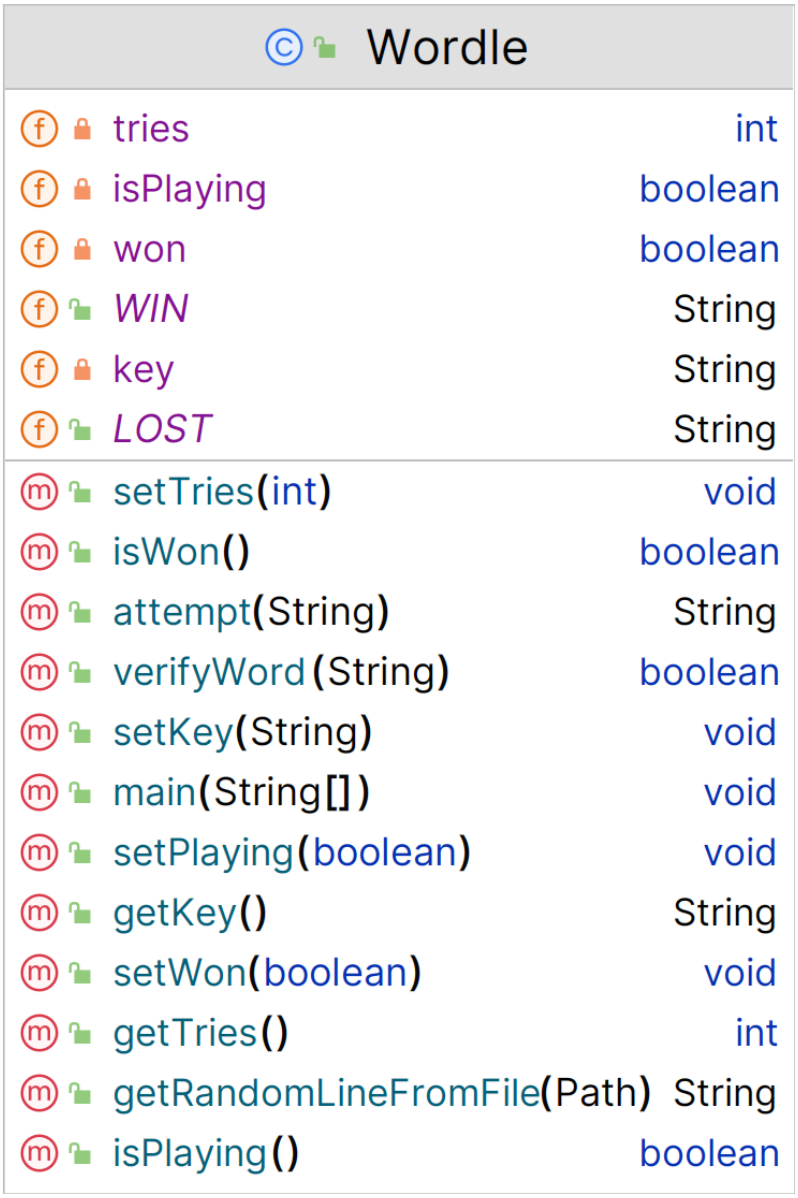

Figure 3.6: Wordle game scene

```
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

In the constructor, we'll be intializing the *key* property with the **getRandomLineFromFile** method.

3.4.2 Verifying user's guess

Veryfing the user's input wasn't complex. There's another file containing all valid input from the user - which is of the same format as the answers file. We'll check if user's input's contains exactly 5 characters first, and then we'll be searching it in the file.

```
public static boolean verifyWord(String s) {
    if (s.length() != 5) {
        return false;
    }
    Scanner scanner = null;
    try {
        scanner = new Scanner(new File("src/main/resources/Utils/valid_wordle.txt"));
    } catch (FileNotFoundException e) {
        throw new RuntimeException(e);
    }
    while (scanner.hasNextLine()) {
        String wordle = scanner.nextLine();
        if (s.toLowerCase().equals(wordle.toLowerCase())) {
            return true;
        }
    }
    return false;
}
```

3.4.3 Handling attempts

We'll begin by taking the lower case version of the user's input, then for each characters. If the game state returns still playing, then we'll check the user's input with the actual answer. In case of wrong guess from user, we'll give them the *coordinates* of the characters. 'n' represent that the character at the position i doesn't exits in the answer, 't' represent the correct appearance of the character in the answer, and 'f' means that it does appear in the answer but in the wrong position.

```
public String attempt(String input) {
    String s = input.toLowerCase();
    if (this.isPlaying) {
        if (s.equals(this.key)) {
            this.isPlaying = false;
```

```
            this.won = true;
            return Wordle.WIN;
        } else {
            String states = "";
            for (int i = 0; i < 5; i++) {
                // If key doesn't contain the character from user's answer
                if (this.key.indexOf(s.charAt(i)) == -1) {
                    states += "n";
                } else {
                    if (this.key.charAt(i) != s.charAt(i)) {
                        states = states + "f";
                    } else {
                        states = states + "t";
                    }
                }
            }
            this.tries += 1;
            if (this.tries == 6) {
                this.isPlaying = false;
                return Wordle.LOST;
            }
            return states;
        }
    } else {
        return Wordle.LOST;
    }
}
```

This class gives us a solid base to built a graphic interface on.

3.4.4 Wordle GUI

The graphics users interface layout was simple. However, due to our lack of proficiency in JavaFX, we had to figure how to put initialize those 36 square label manually. We've tried our best to mimic this version of Wordle made in JavaFX made by jpkhawam, where he direct user input directly to those label by handling every of the key events. We used a separate text field for guess validation and another text field below to print out the answer state. The controller was very simple to implements, since all game logic was implemented in the **Wordle** class. All handled to the controller was to render out the colors of labels according to the **Wordle.attempt** result, and print out guess validation in the text-field.

Figure 3.7: Wordle Class Diagram

For guess checking and game resetting, we present the 2-D array for better use of the 36 labels.

```
Label[][] boxesArray = {
        {box00, box01, box02, box03, box04},
        {box10, box11, box12, box13, box14},
        {box20, box21, box22, box23, box24},
        {box30, box31, box32, box33, box34},
        {box40, box41, box42, box43, box44},
        {box50, box51, box52, box53, box54}
};
```

For each guess, we'll be checking if the word is valid. In the case of invalid input, we'll print out the line noting user that their input is of incorrect state, and they'll be allowed to enter another input. Otherwise, we'll be checking their inputs with the key answer. This below is an example snippet of code that renders out the color of the user attempt, note that the Wordle object have the property *tries*, which can be used to indicate the row number needed to be styled.

```
for (int i = 0; i < 5; i++) {
    boxesArray[row][i].setText(guess.substring(i, i + 1).toUpperCase());
    if (attemptResult.charAt(i) == 'n') {
        boxesArray[row][i].setStyle("-fx-background-color: #808080;" +
                "-fx-text-fill: #fbfcff;" +
                "-fx-font-size: 18");
    } else if (attemptResult.charAt(i) == 't') {
        boxesArray[row][i].setStyle("-fx-background-color: #79b851;" +
                "-fx-text-fill: #fbfcff;" +
                "-fx-font-size: 18");
    } else {
```

```
        boxesArray[row][i].setStyle("-fx-background-color: #f3c237;" +
                "-fx-text-fill: #fbfcff;" +
                "-fx-font-size: 18");
    }
}
```

Resetting the game state was also very simple, we'll clear out all the contents of all related fields and reset their styles.

```
@FXML
private void handleClickReset(MouseEvent mouseEvent) {
    wd = new Wordle();
    Label[][] boxesArray = {
            {box00, box01, box02, box03, box04},
            {box10, box11, box12, box13, box14},
            {box20, box21, box22, box23, box24},
            {box30, box31, box32, box33, box34},
            {box40, box41, box42, box43, box44},
            {box50, box51, box52, box53, box54}
    };
    for (int i = 0; i < 6; i++) {
        for (int j = 0; j < 5; j++) {
            boxesArray[i][j].setText("");
            boxesArray[i][j].setStyle("-fx-background-color: #fbfcff;" +
                    "-fx-border-color: #000000;");
        }
    }
    guessInput.setText("");

    logger.setText("");
}
```

Acknowledgements

We would like to express our sincere gratitude to our instructors, Mr. To Van Khanh and Ms. Nguyen Thu Trang, for imparting invaluable lessons and exhibiting an energetic attitude towards our learning. This project still harbors plenty opportunities for improvement, and we will remain committed to enhancing our programming and problem-solving skills in the future as a gesture of respect and appreciation for their guidance.