Criterion C: Development

Software used: IntelliJ, Scenebuilder, JavaFX

Languages: Java

Techniques used:

A. OBJECTS AND CLASSES
B. ERROR VALIDATION
C. CUSTOM METHODS
D. JAVA FX FRAMEWORK
E. GSON & JSON
F. API
G. JavaFX Tables and CellFactory

A: Objects and classes

```java
public class Vocab {

    String newWord;
    String definition;
    String translation;
    String pronunciation;
    String links;


    public Vocab(String newWord, String definition, String translation, String pronunciation, String links) {
        this.newWord = newWord;
        this.definition = definition;
        this.translation = translation;
        this.pronunciation = pronunciation;
        this.links = links;

    }

    public String getNewWord() { return newWord; }

    public void setNewWord(String newWord) {this.newWord = newWord;}

    public String getDefinition() {return definition;}

    public void setDefinition(String definition) { this.definition = definition; }

    public String getTranslation() { return translation; }

    public void setTranslation(String translation) { this.translation = translation; }
```

```
Collections.sort(imports, new Comparator<Vocab>() {
    @Override
    public int compare(Vocab vocab1, Vocab vocab2) {
        return vocab1.getNewWord().compareTo(vocab2.getNewWord());
    }
});
```

Explanation:

For my project I needed a way for my user to store a collection of all the new vocabulary they learn. Using classes and objects is an appropriate technique as it allows me to logically organize objects of my vocabulary words. The vocabulary was made into a class and sections for each word in the class were added. This class “Vocab” contains the objects newWord for the input word, its definition, translation, pronunciation, and links, which is a space that allows the user to input any word related or connected to the new vocabulary word. Using the getter and setter methods in my class I can link this to the GUI. All these vocabulary words are put in an array and sorted alphabetically using an algorithm that compares an item to the next one. The words are then displayed in the GUI. I could have done this using separate string arrays for the above but using a class enables me to make modifications and improvements if necessary.

B: Error and validation

```
Gson gson = new Gson();
try (Reader reader = new FileReader( fileName: "vocab.json")) {
    //convert JSON file to Java Object
    ArrayList<Vocab> imports = gson.fromJson(reader, new TypeToken<ArrayList<Vocab>>() {
    }.getType());

    Collections.sort(imports, new Comparator<Vocab>() {
        @Override
        public int compare(Vocab vocab1, Vocab vocab2) {
            return vocab1.getNewWord().compareTo(vocab2.getNewWord());
        }
    });

    StartApplication.vocab = FXCollections.observableArrayList(imports);
} catch (IOException e) {
    e.printStackTrace();
```

```
public void CheckInput(){
    String str = input.getText(); //set the user's input as a string
    if (randomWord.contains(str)) { //check if the word selected contains the user's input (letter guessed)
        int index = 0;
        for(int i=0; i<randomWord.length(); i++) { //check every index of the word selected
            char c = randomWord.charAt(i);

            if (String.valueOf(c).equals(str)) { //if the current index (letter it's looking at) matches the user's input
                setLetter(index, Character.toString(c)); //if true, display the letter in the text box
                wrongRight.setText("Right!"); //display "Right!" for affirmation
                score += 100; //add 100 points to user's score
                String scoreText = String.valueOf(score); //turn int into string
                points.setText(scoreText);
                guessed++; //add 1 to correct guesses
                System.out.println(guessed + " out of " + randomWord.length()); //keep track of correct guesses

                if(guessed == randomWord.length()) {
                    wrongRight.setText("Done!"); //Display "Done!" once all the whole word has been guessed correctly
                }
            }
            index++;
        }
    }
    else if (isInteger(str)) {
        wrongRight.setText("No integers!"); //notify to not use integers
    }
    else {
        wrongRight.setText("Wrong"); //display "Wrong" is the letter guessed is not in the word
        score -= 10; //subtract 10 points from score
        String scoreText = String.valueOf(score);
        points.setText(scoreText); //update score
    }
}
```

```
private boolean isInteger(String str) {
    try {
        Integer.parseInt(str); //try to convert the input string to an integer
        return true; //input string can be parsed as an integer
    } catch (NumberFormatException e) {
        return false; //input string cannot be parsed as an integer
    }
}
```

Explanation:

A try-catch catches an IOException which is effective as it throws an exception if there is an error reading the vocab.json file for the vocabulary array list. Using FileReader, the Reader reads the file's content. The fromJson() method of the GSON object converts the JSON data in the file into an ArrayList<Vocab> object. If an exception occurs in the try section during this code's execution, the catch block will execute to catch the exception and print it using the printStackTrace() method.

To meet success criteria 8, I made sure the user couldn't give invalid input to the program's game. The second image shows the use of isInteger(). The try-catch in this method (shown

in the third image) verifies if the input for the game is an integer or not with a boolean in order to try to convert the input string to an integer.

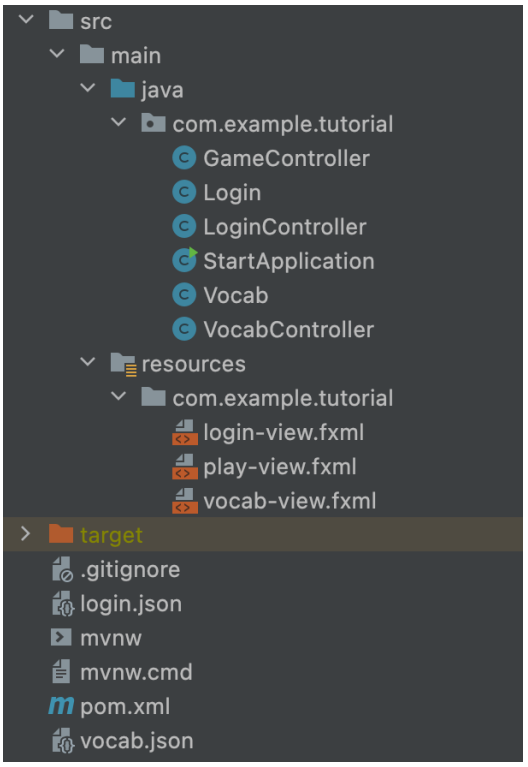
C: Custom methods

```
public void CheckInput(){
    String str = input.getText();
    if (randomWord.contains(str)) {
        int index = 0;
        for(int i=0; i<randomWord.length(); i++) {
            char c = randomWord.charAt(i);

            if (String.valueOf(c).equals(str)) {
                setLetter(index, Character.toString(c));
                wrongright.setText("Right!");
                score += 100;
                String scoretext = String.valueOf(score);
                points.setText(scoretext);
                guessed++;
                System.out.println(guessed + " out of " + randomWord.length());

                if(guessed == randomWord.length()) {
                    wrongright.setText("Done!");
                }
            }
            index++;
        }
    }
    else {
        wrongright.setText("Wrong");
        score -= 10;
        String scoretext = String.valueOf(score);
        points.setText(scoretext);
    }
}
```

Explanation:

This method is used in the Java class: game controller. After the game selects a random word from the user's vocabulary list (the array list), the user then inputs letters one at a time to guess what the word selected is. To check if the random word selected contains the letter the user input, outer if/else statements are used. Inside these outer if/else statements, there's a for loop that iterates through each character of the word selected (labeled as "c"). If the input (letter guessed) is equal to a letter in the selected word, the letter that was guessed correctly is displayed in a box that belongs to the index of the letter in the word. The program displays "right!" and updates the user's score by adding 100 points. The number of correct guesses is kept track of in the console, and when the number of guesses equals the length of the word, the program displays "done" to indicate completion. In the case that the user's input is not in the word, then the outer else statement makes the game display "wrong" and take out 10 points of the user's score.

D: JavaFX Framework



Explanation:

Tools like AWT, Swing, and JavaFX are used to create GUIs, but AWT doesn't have built-in packages and libraries to create things like tables [5], so JavaFX was used, considering that the client's data would be better viewed in a table. Each controller has its own fxml view page. The program's first view is the login. The login controller (LoginController) checks the user's username. After the user enters their username clicks enter in the GUI, the program directs the user to the second view for the vocabulary list (vocab-view.fxml). The array list of the vocabulary words in VocabController gets its items from the vocab.json file which saves the words, their definitions, and links. This vocab.json file and the vocabulary array list is also used in the GameController for the game, as the game selects a random word from the list for the user to guess.

E: GSON & JSON

```
private void loadVocab() {
    //load vocab from saved file.
    //Open and read Json for any previously saved data.

    Gson gson = new Gson();
    try (Reader reader = new FileReader( fileName: "vocab.json")) {
        //convert JSON file to Java Object
        ArrayList<Vocab> imports = gson.fromJson(reader, new TypeToken<ArrayList<Vocab>>() {
        }.getType());
```

```
while ((inputLine = in.readLine()) != null) {
    response.append(inputLine);
}
in.close();

JSONParser parser = new JSONParser();
JSONArray jsonArray = (JSONArray) parser.parse(response.toString());
JSONObject jsonObject = (JSONObject) jsonArray.get(0);

JSONArray meaningsArray = (JSONArray) jsonObject.get("meanings");
JSONObject meaningsObject = (JSONObject) meaningsArray.get(0);

JSONArray definitionsArray = (JSONArray) meaningsObject.get("definitions");
JSONObject definitionsObject = (JSONObject) definitionsArray.get(0);

String definition = (String) definitionsObject.get("definition");
cDefinitionTxt.setText(definition); //set definition in definition box
```

Explanation:

To meet success criteria 2, I needed to save my client's vocabulary words in an external file. Instead of using a text document, I opted for JSON, as it keeps the structure of the classes I made. The GSON Java library can convert JSON data into Java objects and vice versa. This is effective, as in this case, the GSON object reads the data in the vocab.json file where the vocabulary words are stored. The user can input or delete vocabulary words in the GUI and the changes will be saved in the vocab.json file. The fromJson() method parses the JSON data into an ArrayList<Vocab> object. It takes two parameters: the Reader and the TypeToken to read the data and convert it. The JSON Jar was downloaded and implemented into the program specifically for the dictionary API.

The JSON Array, Object, and Parser were imported at the top of VocabController, as the button that retrieves a vocabulary words' definition needed these. The section of code in the image creates a JSON parser object and parses the response string into a JSON array to then extract the JSON object from the first element of the array. The code gets the "meanings" JSON array to extract a JSON object from the first element of the array and then

retrieves the “definitions” array to extract an object. The value of the words’ definition key is retrieved as a string and is set to a specific text field in the GUI for the user to view.

F: API

```
public void searchBtn(ActionEvent actionEvent) throws Exception {
    String word = cNewWordTxt.getText();
    String url = "https://api.dictionaryapi.dev/api/v2/entries/en_US/" + word;

    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    // optional default is GET
    con.setRequestMethod("GET");

    BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();
```

Explanation:

An alternative to the client typing all the definitions themselves which is time consuming and increases chances of inaccuracies due to human error, is using a dictionary API to retrieve a vocabulary word’s definition. This is an appropriate technique as the user no longer has to search each word’s definition and type it in the program. It also ensures that the definition is accurate, as it should be more detailed or precise than the one thought by the user. This dictionary API searches a word in its database using the URL, and the definition is displayed in the GUI for the user as a string by setting the string in the definition label [6].

G: JavaFX Tables and CellFactory

```java
public void initialize() {

    loadVocab(); //load words from the vocabulary list

    vocabWord.setCellValueFactory(new PropertyValueFactory<Vocab, String>("newWord"));

    contactsTable.getColumns().add(vocabWord);
    contactsTable.setItems(StartApplication.vocab);

    contactsTable.setRowFactory(rowClick -> {
        TableRow<Vocab> row = new TableRow<>();
        row.setOnMouseClicked(event -> {
            if (!row.isEmpty() && event.getButton() == MouseButton.PRIMARY && event.getClickCount() == 2) {
                Vocab clickedRow = row.getItem();
                cNewWordTxt.setText(clickedRow.getNewWord());
                cDefinitionTxt.setText(clickedRow.getDefinition());
                cTranslationTxt.setText(clickedRow.getTranslation());
                cPronunciationTxt.setText(clickedRow.getPronunciation());
                cLinksTxt.setText(clickedRow.getLinks());
            }
        });
        return row;
    });
}
```

```java
StartApplication.vocab = FXCollections.observableArrayList(imports);
```

Explanation:

Using JavaFX Tables is a better option than AWT tables because JavaFX tables can be linked to an observable list, which is part of JavaFX, which keeps the data in the table up to date with the items in the observable list. The images show that the observable list was used to add a custom on-mouse click event that would allow having updates in a text box. The new vocabulary that is added to the list is automatically added to the table, which makes maintenance easier. This is better than using an arraylist of objects because it discards the need to update the table manually each time a new word is added to the observable list. Cell Factory is good to use because it will create a new cell whenever a new one is needed. The cells fit the visual space in the program [7].