Criterion C: Development

Table of contents

Home page and key methods:

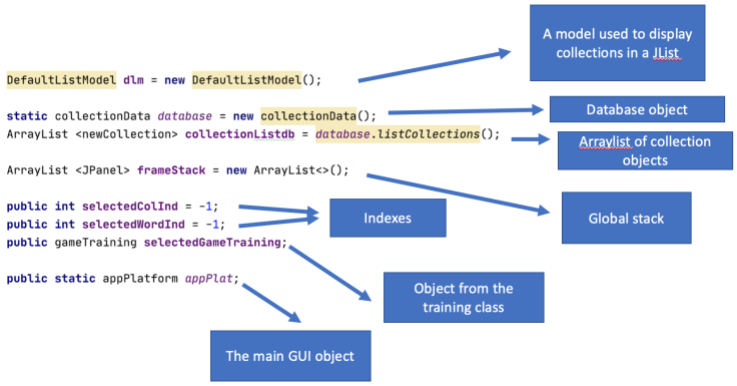
For the application's GUI, I have chosen the integrated GUI system from IntelliJ as it is more intuitive and allows the design to be abstracted and not seen in the program, allowing the program to be more organized. The CardLayout format was selected since it treats the frames as a deck of cards. The container acts as a stack of cards and only one card is visible at a time. CardLayout defines a set of methods that allow an application to flip through these cards sequentially, or to show a specified card ("CardLayout"). This format becomes essential to the application's work as the program's methods will revolve around the concept of "card stack".

Imports and containers used:

```
import javax.swing.*;
import java.awt.event.*;
import java.io.IOException;
import java.util.ArrayList;
JPanel appCardLayout;
JPanel homePanel;
private JButton wordCollectionsButton;
private JButton trainingModeButton;
private JButton exitButton;
private JPanel collectionPage;
private JButton previousButton;
private JButton backToHomeButton;
private JPanel parentPanel;
private JPanel frame2;
private JButton createButtonFrame2;
private JButton viewEditButtonFrame2;
private JPanel frame4;
private JPanel parentPanel2;
private JPanel trainingPage;
private JButton backToHomeButton2;
private JButton previousButton2;
private JButton newCollectionButtonFrame4;
private JList collectionSelectionlistFrame4;
private JButton proceedWithTheSelectedCollectionButtonFrame4;
private JPanel frame5;
private JTextField textFieldNewCollectionFrame5;
private JButton proceedButtonFrame5;
private JPanel frame6;
private JTextField textFieldFrontCreation;
private JTextField textFieldBackCreation;
private JButton proceedButtonFrame6;
private JPanel frame7;
private JButton proceedWithTheSelectedCollectionButtonFrame7;
private JList existentCollections;
```

```
private JPanel frame8;
private JPanel viewPanel;
private JLabel collectionNameGUI;
private JPanel editPanel;
private JTextArea textArea1;
private JTextArea textArea2;
private JButton editButtonViewPanel;
private JList collectionSpecific;
private JButton proceedWithTheSelectedWordButtonFrame8;
private JButton editFinishButtonEditPanel;
private JTextField textFieldFrontEdit;
private JTextField textFieldBackEdit;
private JPanel frame3;
private JList trainingList;
private JButton editcollectionNameButtonframe8;
private JPanel editColNamePanel;
private JTextField textFieldEditColNamePanel;
private JButton proceedEditColNamePanel;
private JButton deleteWordButtonViewPanel;
private JButton deleteCollectionFrame8;
private JButton proceedWithTheSelectedCollectionButtonFrame3;
private JButton guessAndTypeButton;
private JButton multiChoiceButton;
private JButton rememberncheckButton;
private JButton rememberncheckReversedButton;
private JButton multiChoiceReversedButton;
private JButton guessAndTypeReversedButton;
private JPanel frame10;
```

Global variables:



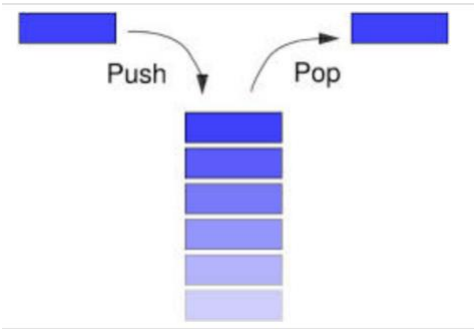
Stack:
Stack, an abstract data type based on the Last-In-First-Out structure("Stack"), will be used throughout the program. When the user clicks a button that leads to a panel that wasn't registered in the stack, the method nextFrameStack is used, showing the intended frame while adding the frame to the frameStack:

```
public void nextFrameStack (JPanel parent, JPanel frame){
    parent.removeAll();
    parent.add(frame);
    parent.repaint();
    parent.revalidate();
    frameStack.add(frame);

}
```

Else, if the button leads to an already existent frame(e.g if the user clicked on the "previous" button), the method getFrame is used instead (the stack is not involved here):

```
public void getFrame (JPanel parent, JPanel frame){
        parent.removeAll();
        parent.add(frame);
        parent.repaint();
        parent.revalidate();
    }
```

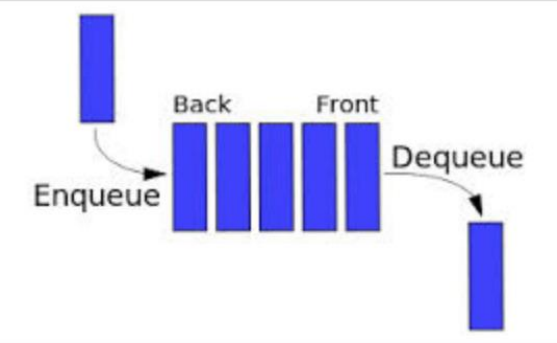
Visualization of a stack structure ("Stacks and Queues."):



Queue:
Queue, an abstract data type based on the First-In-First-Out structure("Queue"), is used during the training part of the program. When the user clicks on the "guess and type" or "Multiple choice" exercise, questions are generated in order. With that in mind, the array list questionWordInd will then record the indexes of the question words in that order, and if the user gets the answer right or wrong, the program will dequeue the first element in the queue to edit its familiarity count. The code would look as followed:

```
int dequeueQuestionWordInd (){
    int hold = questionWordInd.get(0);
    questionWordInd.remove( index: 0);
    return hold;
}
```

Visualization of a queue structure ("Stacks and Queues."):



Returning frames (previous and home button):
There will be a previous and a home button in every panel after the home panel to allow the user to go back. For the “previous” button, the top element of frameStack is removed and the element below is displayed instead. For the “home” button, the entire stack is cleared, and the home panel is displayed. The code is shown as followed:

```java
previousButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (frameStack.size() == 1) { // the only panel existent is panel2 -> functions as a home button
            getFrame(appCardLayout, homePanel);
            frameStack.clear();
            selectedColInd = -1;
            selectedWordInd = -1;
            selectedGameTraining = null;

        } else if (frameStack.get(frameStack.size()-1).equals(frame8)) { // if the current panel panel8(needs to reset the list)
            frameStack.remove( index: frameStack.size() - 1);
            getFrame(parentPanel,frame7);
            dlm.clear();
            existentCollections.setModel(dlm);
            for (int i = 0; i < collectionListdb.size(); i++) {
                dlm.addElement(collectionListdb.get(i).collectionName);
            }
            selectedColInd = -1;
            selectedWordInd = -1;
            selectedGameTraining = null;

        } else if (frameStack.get(frameStack.size()-1).equals(viewPanel)) { // if the current panel viewPanel(needs to reset the list)
        frameStack.remove( index: frameStack.size() - 1);
        getFrame(parentPanel,frame8);
        textArea1.setText(null);
        textArea2.setText(null);
        selectedWordInd = -1;

        }else {
            getFrame(parentPanel, frameStack.get(frameStack.size() - 2));
            frameStack.remove( index: frameStack.size() - 1);
            selectedColInd = -1;
            selectedWordInd = -1;
            selectedGameTraining = null;
        }
    }
});
```

```java
backToHomeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent actionEvent) {
        getFrame(appCardLayout, homePanel);
        frameStack.clear();
        selectedColInd = -1;
        selectedWordInd = -1;
    }
});
```

Sorting:

I first planned to make the program present every list ordered alphabetically (a to z) but since the words are grouped by subtopics (meaning they are related), I decided to only make the list of collections ordered alphabetically. To do that, I used the selection sorting algorithm (refer to the key algorithms in criterion B). The code is shown below (located in the Collection database class):

```
static void selectionSort(ArrayList<newCollection> arr) {
    // One by one move boundary of unsorted subarray
    // Time complexity O(n^2)
    for (int i = 0; i < arr.size()-1; i++) {
        // Find the minimum element in unsorted array
        int min_index = i;
        int comp = 0;
        for (int j = i+1; j < arr.size(); j++) {
            comp = compareTo(arr.get(j), arr.get(min_index));
            if (comp < 0){ // if arr.get(j) is "smaller" than arr.get(min_index)
                min_index = j;
            }
        }
        // Swap the found minimum element with the first element
        Collections.swap(arr, i, min_index);

    }
}


private static int compareTo(newCollection col1, newCollection col2) {
    return col1.collectionName.toLowerCase().compareTo(col2.collectionName.toLowerCase());
}
```

File Reading/Writing using BufferedReader and BufferedWriter:
To ensure that data can be reloaded if the application restarts, data records must be saved in local files. Furthermore, once data records are altered, files must be updated. BufferedReader and BufferedWriter are used in the software to perform the reading and writing operations. BufferedReader and BufferedWriter are Java classes that read text from a character-input stream ("Bufferedreader.") and write text to a character-output stream ("Bufferedwriter.") respectively, buffering characters to allow for efficient reading/writing of characters, lines, and arrays (Jumba).

Without buffering, each invocation of reading/writing can be wasteful, as bytes may be read from the file, converted to characters, and then returned for readers alone, and characters may be converted to bytes and then written to the file immediately for writers only (Jumba).

As a result, BufferedReader and BufferedWriter are both utilized to increase efficiency by using buffers.

I have then created some txt files from the list that Mrs. Morey has sent to me (view in Appendix 2) and made it into the "front ~~~~ back ~~~~ familiarCount" format. In the collectionData class, all the text files in the "collection" folder are read and converted into objects each with 3 array lists named "front", "back" and "familiarCount".

An example of a txt file created:

Aspecto físico.txt

```
(Tiene / lleva) el pelo rizado y castaño ~~~~ Curly Brown Hair ~~~~ 10
Tiene el pelo corto y negro ~~~~ Short Black Hair ~~~~ 9
Tiene el pelo liso y rubio ~~~~ Straight Blonde Hair ~~~~ 11
Tiene el pelo largo y es pelirrojo ~~~~ Long Red Hair ~~~~ 9
Tiene los ojos castaños ~~~~ Brown Eyes ~~~~ 10
Tiene los ojos verdes ~~~~ Green Eyes ~~~~ 11
Tiene los ojos azules ~~~~ Blue Eyes ~~~~ 10
Tiene los ojos grises ~~~~ Grey Eyes ~~~~ 10
Tiene los ojos negros ~~~~ Black Eyes ~~~~ 9
es alto ~~~~ He is tall ~~~~ 10
es alta ~~~~ She is tall ~~~~ 10
es de estatura mediana ~~~~ Medium Height ~~~~ 11
es bajo ~~~~ He is short ~~~~ 8
es baja ~~~~ She is short ~~~~ 11
es guapa ~~~~ beautiful, handsome ~~~~ 9
es guapo ~~~~ He/she is beautiful ~~~~ 9
es feo ~~~~ Ugly ~~~~ 10
es fea ~~~~ She is ugly ~~~~ 10
es atractivo ~~~~ He is attractive ~~~~ 11
es atractiva ~~~~ She's attractive ~~~~ 11
es calvo ~~~~ Bald ~~~~ 8
es calva ~~~~ He/she is bald ~~~~ 10
es gordo ~~~~ He is fat ~~~~ 11
es gorda ~~~~ She is fat ~~~~ 8
es delgado ~~~~ He is slim ~~~~ 9
es delgada ~~~~ She is thin ~~~~ 8
es fuerte ~~~~ Built ~~~~ 11
(tiene / lleva) barba ~~~~ Beard ~~~~ 10
(tiene / lleva) bigote ~~~~ Moustache ~~~~ 9
lleva tatuaje ~~~~ Tatoo ~~~~ 9
```

Imports and variables in collectionData:

```
import javax.swing.*;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;


public class collectionData {
    public static ArrayList<newCollection> collectionList;

    public collectionData() { collectionList = new ArrayList<>(); }
```

Code in collectionData:

```
public static void collectionCollect() {
    collectionList.clear();
    File folder = new File( pathname: "src/collections");
    File[] listOfFiles = folder.listFiles();


    for (int i = 0; i < listOfFiles.length; i++) {
        File file = listOfFiles[i];

        if (file.isFile() && file.getName().endsWith(".txt")) {
            newCollection collectionidv = new newCollection();
            collectionidv.readFile(file);
            collectionList.add(collectionidv);
        }

    }

    selectionSort(collectionList);
}
```

Imports and variables in the collection class:

```
import javax.swing.*;
import java.io.*;

import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.ArrayList;

public class newCollection {

    public String collectionName;
    public ArrayList<String> front;
    public ArrayList<String> back;
    public ArrayList<Integer> familiarCount;
    public File collectionFile;
    public String filePath = "";
```

Constructor of the collection class:

```
public newCollection (){
    collectionName = "";
    front = new ArrayList<>();
    back = new ArrayList<>();
    familiarCount = new ArrayList<>();
}
```

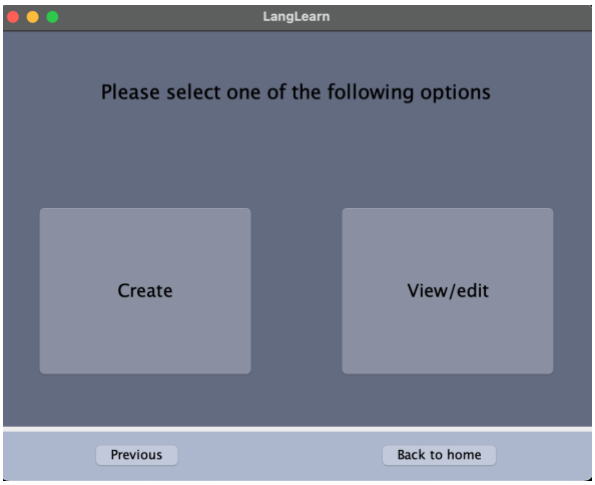
Code for reading file in the collection object:

```java
public void readFile (File collectedFile){
    // reads the file and turns it to an object with arraylists front and back

    //-----------------------------------------------------------------// get the collection's name - ".txt"
    StringBuffer buf = new StringBuffer(collectedFile.getName());

    buf.replace( start: buf.length()-4, buf.length(), str: ""); // remove ".txt" part

    collectionName = buf.toString();

    //-----------------------------------------------------------------//

    collectionFile = collectedFile;
    filePath = "src/collections/" + collectionName + ".txt";

    String input;

    if (collectionFile.exists() && collectionFile.length() != 0) {
        try {
            BufferedReader reader = new BufferedReader(new FileReader(collectionFile));
            input = reader.readLine(); // reads "front ~~~~ back ~~~~ familiarityCount"

            while (input != null) { // while not at the end of file
                String[] parts= input.split( regex: " ~~~~ "); // split into parts
                front.add(parts[0]); // add parts[0] into the front arraylist
                back.add(parts[1]); // add parts[1] into the back arraylist
                familiarCount.add(Integer.parseInt(parts[2])); // add parts[2] into the familiarCount arraylist
                input = reader.readLine(); // repeat if necessary
            }

        } catch (IOException e) {
            System.out.println(e);
        }
    } else {
        System.out.println("File " + collectionFile + " is not found or is empty");
    }

}
```

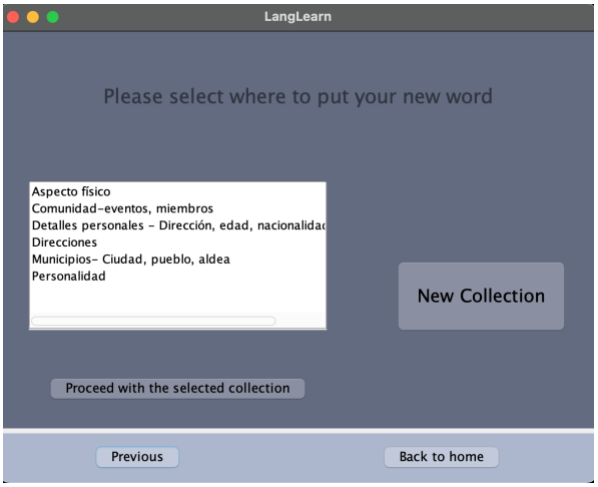
When the program is run, the user is presented with the home panel, where he could choose what to do: “Word Collection” or “Training Mode”.

Word Collection page:

This is the place where the user can create new collections, create new words, edit collections, or edit existent words.



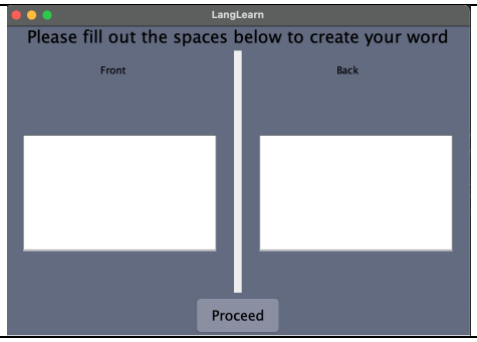
In the “create” section, the user can choose to either add a new word to an existing collection or create a new collection and add the word to it afterward. Error messages are shown when the user does not follow the instructions (like leaving a text field blank).



Creating a new word:

Step		
1.1	Process description	GUI illustration
	To create a new word in a new collection, the user would click on “new collection” which would lead him to the collection naming frame (Frame5)	
	Code involved:	

```
proceedButtonFrame5.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (textFieldNewCollectionFrame5.getText().isEmpty()) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "The field cannot be left blank. You must enter in a name");
        } else {
            nextFrameStack(parentPanel, frame6);
        }
    }
});
```

2	Process description	GUI illustration
	After successfully entering the collection name, the user can proceed to make the word in Frame6	
	Code involved:	

```
proceedButtonFrame6.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        if (textFieldFrontCreation.getText().isEmpty() || textFieldBackCreation.getText().isEmpty()) {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Both text fields must be filled to proceed.");
        } else {
            if (selectedColInd >= 0) { // creating a word inside an existing collection
                collectionListdb.get(selectedColInd).addCollectionWord(textFieldFrontCreation.getText(), textFieldBackCreation.getText());
                showSuccess( congrat: "Word was created with success", parentPanel, frame2);
                refreshDatabase();
            } else { // creating a word inside a new collection (create collection and put the word inside)
                database.addCollection(textFieldNewCollectionFrame5.getText(),textFieldFrontCreation.getText(),textFieldBackCreation.getText());
                showSuccess( congrat: "Word was created with success", parentPanel, frame2);
                refreshDatabase();
            }
            textFieldFrontCreation.setText(null);
            textFieldBackCreation.setText(null);
            textFieldNewCollectionFrame5.setText(null);
        }
    }
});
```

3	Process description	GUI illustration
	JOptionPanel would pop up showing that the collection was successfully created. Meanwhile, in the database, the addCollection method is activated and a new text file with the name with a line "frontInputted ~~~~ backInputted ~~~~ 10 (default familiarity count)" is added.	N/A
	Code involved:	

```java
public static void addCollection(String name, String frontInput, String backInput){
    try {
        File myObj = new File( pathname: "src/collections/" + name + ".txt");
        if (myObj.createNewFile()) {
            BufferedWriter myWriter = new BufferedWriter(new FileWriter(myObj));
            myWriter.write( str: frontInput + " ~~~~ " + backInput + " ~~~~ " + 10);
            myWriter.close();
        } else {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "File name already exists.");
        }
    } catch (IOException error) {
        System.out.println("An error occurred.");
        error.printStackTrace();
    }
}
```

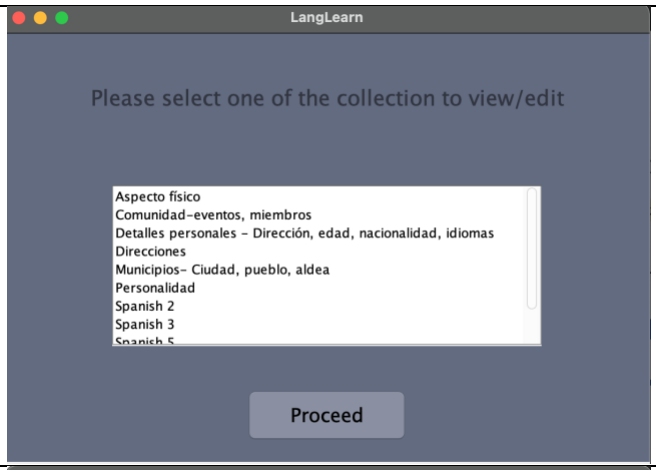
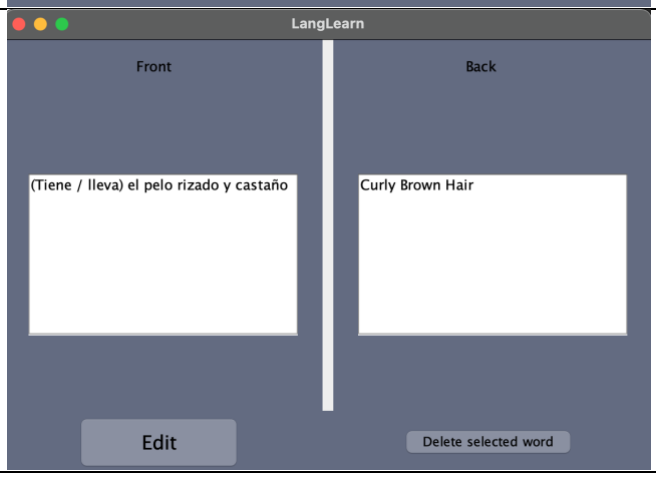
4	Process description	GUI illustration
	After the operations are finished, the database is refreshed, and the user is taken back to Frame2.	N/A

If the user decides to add a new word to an existent collection, he will have to select one of the collections listed in the "create" section. The user is directly led to Frame6 instead (step 2), where he would create the word. The addCollectionWord in the selected collection object is activated afterward.

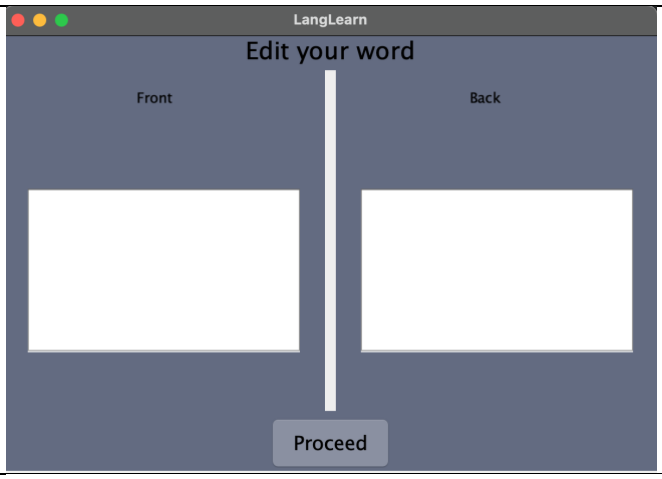
Code:

```java
public void addCollectionWord(String newFront, String newBack){
    front.add(newFront);
    back.add(newBack);

    try {
        BufferedWriter myWriter = new BufferedWriter(new FileWriter(filePath));
        for (int i = 0; i < front.size()-1; i++) {
            myWriter.write( str: front.get(i) + " ~~~~ " + back.get(i) + " ~~~~ " + familiarCount.get(i));
            myWriter.newLine();
        }
        myWriter.write( str: newFront + " ~~~~ " + newBack + " ~~~~ " + 10);
        myWriter.close();

    } catch (IOException error) {
        System.out.println("An error occurred.");
        error.printStackTrace();
    }

}
```

Viewing a word:

Step	Process description	GUI demonstration
1	In the “view/edit” section, the user first chooses a collection from the collection list to view.	
2	After the selection, the user is led to the “word list” frame, where he can proceed to view the word’s front and end completely, edit the collection’s name, and delete the collection.	
3	After selecting a word, the user is led to the view panel, where he can view the “front” and “back” of the word	

Editing an existing word:

Description	If the “edit” button was selected in the view panel, the user can now input the new front or new back and the word will be changed accordingly (just the front, just the back, or both). The user is brought back to frame2 afterward.
GUI illustration	
GUI code	

```java
editFinishButtonEditPanel.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (textFieldFrontEdit.getText().isEmpty() && textFieldBackEdit.getText().isEmpty()) {
            JOptionPane.showMessageDialog( parentComponent: null,  message: "Either text fields must be filled to proceed.");
        } else {
            if (selectedWordInd >= 0) {
                if (!textFieldFrontEdit.getText().isEmpty() && !textFieldBackEdit.getText().isEmpty()) {
                    try {
                        collectionListdb.get(selectedColInd).editCollectionWord(selectedWordInd,
                                textFieldFrontEdit.getText(), textFieldBackEdit.getText());
                    } catch (IOException ioException) {
                        ioException.printStackTrace();
                    }
                } else if (!textFieldFrontEdit.getText().isEmpty() && textFieldBackEdit.getText().isEmpty()){
                    try {
                        collectionListdb.get(selectedColInd).editCollectionWord(selectedWordInd,
                                textFieldFrontEdit.getText(), collectionListdb.get(selectedColInd).back.get(selectedWordInd));
                    } catch (IOException ioException) {
                        ioException.printStackTrace();
                    }
                } else if (textFieldFrontEdit.getText().isEmpty() && !textFieldBackEdit.getText().isEmpty()){
                    try {
                        collectionListdb.get(selectedColInd).editCollectionWord(selectedWordInd,
                                collectionListdb.get(selectedColInd).front.get(selectedWordInd), textFieldBackEdit.getText());
                    } catch (IOException ioException) {
                        ioException.printStackTrace();
                    }
                }
                showSuccess( congrat: "Word was edited with success",parentPanel, frame2);
            }
            refreshDatabase();
            textFieldFrontEdit.setText(null);
            textFieldBackEdit.setText(null);
        }
    }
```

Database code involved:	

```java
public void editCollectionWord(int wordIndex, String newFront, String newBack) throws IOException {
    front.set(wordIndex, newFront);
    back.set(wordIndex, newBack);
    int famCount = familiarCount.get(wordIndex);

    File inputFile = new File(filePath);
    File tempFile = new File( pathname: "src/collections/myTempFile.txt");

    BufferedReader reader = new BufferedReader(new FileReader(inputFile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

    String currentLine;
    int count = 0;

    while ((currentLine = reader.readLine()) != null) {
        if (count == wordIndex) {
            writer.write( str: newFront + " ~~~~ " + newBack + " ~~~~ " + famCount);
            writer.newLine();
            count++;
        } else {
            writer.write(currentLine);
            writer.newLine();
            count++;
        }
    }
    writer.close();
    reader.close();
    inputFile.delete();
    tempFile.renameTo(inputFile);

}
```

Delete a word:

Description	If the user chooses the “delete selected word” button in the view panel, the program first asks if he is sure he wants to delete the word using JOptionPanel to prevent human accidents. If the answer is yes, the method deleteCollectionWord in the selected collection is activated. The user is brought back to frame2 afterward.
GUI code:	

```
deleteWordButtonViewPanel.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (JOptionPane.showConfirmDialog( parentComponent: null,  message: "Do you want to delete this word?",  title: "",
                JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
            try {
                collectionListdb.get(selectedColInd).deleteCollectionWord(selectedWordInd);
            } catch (IOException ioException) {
                ioException.printStackTrace();
            }
            showSuccess( congrat: "Word was deleted with success", parentPanel, frame2);
            refreshDatabase();
        }
    }
});
```

Database code involved

```
public void deleteCollectionWord(int wordIndex) throws IOException {

    front.remove(wordIndex);
    back.remove(wordIndex);

    File inputFile = new File(filePath);
    File tempFile = new File( pathname: "src/collections/myTempFile.txt");

    BufferedReader reader = new BufferedReader(new FileReader(inputFile));
    BufferedWriter writer = new BufferedWriter(new FileWriter(tempFile));

    String currentLine;
    int count = 0;

    while ((currentLine = reader.readLine()) != null) {
        if (count == wordIndex) {
            count++;
        } else {
            writer.write(currentLine);
            writer.newLine();
            count++;
        }
    }
    writer.close();
    reader.close();
    inputFile.delete();
    tempFile.renameTo(inputFile);

}
```

Edit collection name:

Description	If the “edit collection name” button is selected in the “word list” frame, a frame similar to the new collection naming frame is shown. The user is brought back to frame2 afterward.
GUI illustration	

GUI code:	`proceedEditColNamePanel.addActionListener(new ActionListener() {`
	`@Override`
	`public void actionPerformed(ActionEvent e) {`
	`  if (textFieldEditColNamePanel.getText().isEmpty()) {`
	`  JOptionPane.showMessageDialog( parentComponent: null,`
	`  message: "The field cannot be left blank. You must enter in a name");`
	`  } else {`
	`  boolean nameChanged = false;`
	`  while (!nameChanged) {`
	`  nameChanged = collectionListdb.get(selectedColInd).editCollectionName(textFieldEditColNamePanel.getTe`
	`  }`
	`  showSuccess( congrat: "Collection's name was edited with success",parentPanel, frame2);`
	`  }`
	`  refreshDatabase();`
	`}`
	`});`

Database code involved	`public boolean editCollectionName(String nameChanged){`
	`  collectionName = nameChanged;`
	
	`  // Replace the file path with path of the directory`
	`  File rename = new File( pathname: "src/collections/"+nameChanged+".txt");`
	
	`  // store the return value of the method`
	`  boolean isChanged = collectionFile.renameTo(rename);`
	
	`  // if the method return true then if block is executed`
	`  if (rename.exists()) {`
	`  if (!isChanged) {`
	`  System.out.println("Operation Failed");`
	`  }`
	`  } else {`
	`  JOptionPane.showMessageDialog( parentComponent: null,`
	`  message: "Collection name already taken, please try a new one");`
	`  }`
	`  return isChanged;`
	
	`}`

Delete a collection:

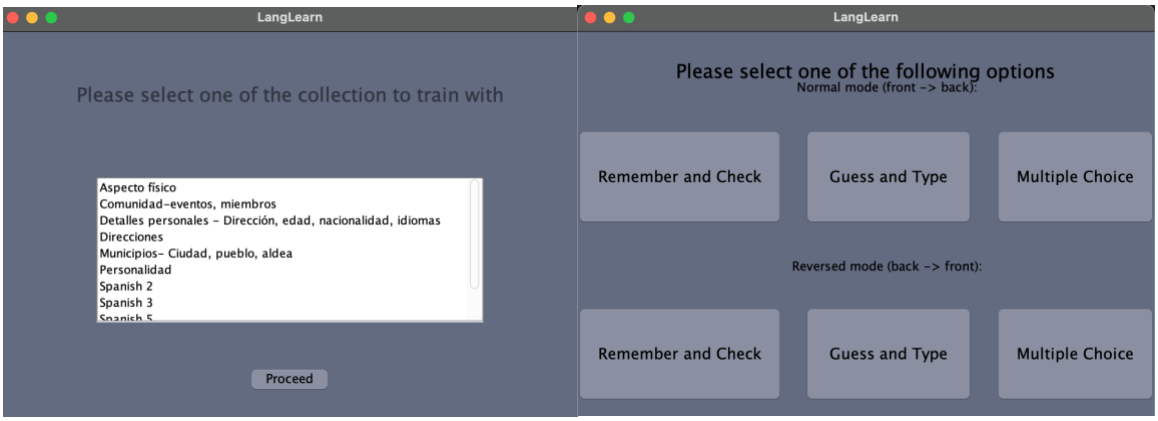
Description	If the user chooses the “delete collection” button in the “word list” panel, the program first asks if he is sure he wants to delete the word using JOptionPanel to prevent human accidents. If the answer is yes, the method deleteCollection in the database object is activated. The user is brought back to frame2 afterward.

GUI code:	`deleteCollectionFrame8.addActionListener(new ActionListener() {`
	`  @Override`
	`  public void actionPerformed(ActionEvent e) {`
	`  if (JOptionPane.showConfirmDialog( parentComponent: null,`
	`  message: "Do you want to delete this collection?", title: "",`
	`  JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {`
	`  database.deleteCollection(selectedColInd);`
	`  showSuccess( congrat: "Collection was deleted with success", parentPanel, frame2);`
	`  refreshDatabase();`
	`  }`
	`  }`
	`});`

Database code involved	`public static void deleteCollection(int colIndex){`
	`  collectionList.get(colIndex).collectionFile.delete();`
	`  collectionList.remove(colIndex);`
	`}`

Training Page:

This is the place where the user can train his familiarity with the words. A collection must be selected to continue as an object will be created and the exercises will be based on the words in that collection object.



Imports and containers:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

public class trainingPlatform {
    private JPanel trainingPage;
    private JButton previousButton2;
    private JButton backToHomeButton2;
    private JPanel parentPanel2;
    private JPanel multiChoiceFrame;
    private JLabel questionMultiChoice;

    private JButton option1, option2, option3, option4;
    private JPanel buttonPanel2;
    private JPanel guessnTypingFrame;
    private JLabel questionTyping;
    private JTextField textFieldTyping;
    private JButton proceedTyping;
    private JPanel rememberCheckFrame;
    private JLabel questionRemember;
    private JLabel answerCheck;
    private JButton checkButton;
```

Global variables:

```
String correct_answer;

public boolean firstTime = true;
public ArrayList<Integer> questionWordInd = new ArrayList<>();

public newCollection collectionSelected;

ArrayList<rememberCheck> qRC = new ArrayList<>(); // create an array
ArrayList<guessType> qGT = new ArrayList<>(); // create an array
ArrayList<multiChoice> qMC = new ArrayList<>(); // create an array
```

Roulette wheel selection and the picking system:
In order to pick words that the students are weaker against, I have chosen the roulette wheel selection algorithm (refer to the key algorithms in criterion B).
With the algorithm, the picking system's code would look like followed:

Remember and Check:

```java
public void createQuestionsR(ArrayList<rememberCheck> quiz, boolean reverse) {
    // select a random word from the collection

    int indexChosen = rouletteSelect();
    if (!reverse) {
        String question = collectionSelected.front.get(indexChosen);
        String correctAnswer = collectionSelected.back.get(indexChosen);

        rememberCheck q = new rememberCheck(question, correctAnswer);
        quiz.add(q);
    } else {
        String question = collectionSelected.back.get(indexChosen);
        String correctAnswer = collectionSelected.front.get(indexChosen);

        rememberCheck q = new rememberCheck(question, correctAnswer);
        quiz.add(q);
    }
}
```

Guess and Type:

```java
public void createQuestionsT(ArrayList<guessType> quiz, boolean reverse) {
    // select a random word from the collection

    int indexChosen = rouletteSelect();
    questionWordInd.add(indexChosen); // the word's index will be recorded for editing
    if (!reverse) {
        String question = collectionSelected.front.get(indexChosen);
        String correctAnswer = collectionSelected.back.get(indexChosen);

        guessType q = new guessType(question, correctAnswer);
        quiz.add(q);
    } else {
        String question = collectionSelected.back.get(indexChosen);
        String correctAnswer = collectionSelected.front.get(indexChosen);

        guessType q = new guessType(question, correctAnswer);
        quiz.add(q);
    }
}
```

Multiple Choice:

```java
public void createQuestionsM(ArrayList<multiChoice> quiz, boolean reverse){
    // select a random word from the collection
    // gets its front(served as the question) and back(served as the correct answer)
    // fill the rest of the alternative answers randomly(must not be the same as the correct answer)

    if (!reverse) {
        int[] listGenerated = generateRandom(collectionSelected.front.size(), questionsNeeded: 4);
        questionWordInd.add(listGenerated[0]); // the word's index will be recorded for editing
        String question = collectionSelected.front.get(listGenerated[0]);
        String correctAnswer = collectionSelected.back.get(listGenerated[0]);
        shuffleArray(listGenerated);
        String[] op = new String[4];

        for (int j = 0; j < listGenerated.length; j++) {
            op[j] = collectionSelected.back.get(listGenerated[j]);
        }
        multiChoice q = new multiChoice(question, op[0], op[1], op[2], op[3], correctAnswer);

        quiz.add(q);

    } else {
        int[] listGenerated = generateRandom(collectionSelected.front.size(), questionsNeeded: 4);
        questionWordInd.add(listGenerated[0]); // the word's index will be recorded for editing
        String question = collectionSelected.back.get(listGenerated[0]);
        String correctAnswer = collectionSelected.front.get(listGenerated[0]);
        shuffleArray(listGenerated);
        String[] op = new String[4];

        for (int j = 0; j < listGenerated.length; j++) {
            op[j] = collectionSelected.front.get(listGenerated[j]);
        }

        multiChoice q = new multiChoice(question, op[0], op[1], op[2], op[3], correctAnswer);

        quiz.add(q);
    }
}
```

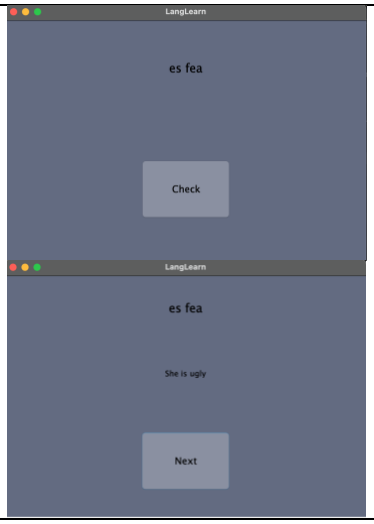
Additional code involved	

```java
int[] generateRandom(int max, int questionsNeeded) { // generate a list of random indexes depending on the
    Random rand = new Random(); //instance of random class

    int[] list = new int[max]; // generate a list of ascending numbers with length "max"
    for (int i = 0; i < list.length; i++) { //fill it up
        list[i] = i;
    }
    shuffleArray(list);

    int chosenIndex = rouletteSelect(); // this would be the index of the word most unfamiliar

    int[] finalList = new int[questionsNeeded]; // Generate an array based on the number of questions Needed
    finalList[0] = chosenIndex; // the first one will be the question asked

    int chosenIndexinlist = -1;
    for (int j = 0; j < list.length; j++) {
        if(list[j] == chosenIndex){
            chosenIndexinlist = j;
            break;
        }
    }
    int[] listwithRemoved = new int[list.length-1];

    for (int i = 0, j = 0; i < list.length; i++) {
        if (i != chosenIndexinlist) {
            listwithRemoved[j++] = list[i];
        }
    }

    int indexRand = rand.nextInt( bound: listwithRemoved.length-(questionsNeeded-2));

    for (int k = indexRand; k < indexRand+(questionsNeeded-1); k++) {
        finalList[k-indexRand+1] = listwithRemoved[k]; // fill the rest of the array with filler questions
    }

    return finalList;
}
static void shuffleArray(int[] arr) { // shuffle array
    Random rnd = new Random();
    for (int i = arr.length - 1; i > 0; i--) {
        int index = rnd.nextInt( bound: i + 1); // Simple swap
        int a = arr[index];
        arr[index] = arr[i];
        arr[i] = a;
    }
}
```

Activities explanation:

There are 3 different types of exercises in the program, with each having a reverse mode where instead of guessing the back with the front, the front is guessed instead (which allowed me to use polymorphism).

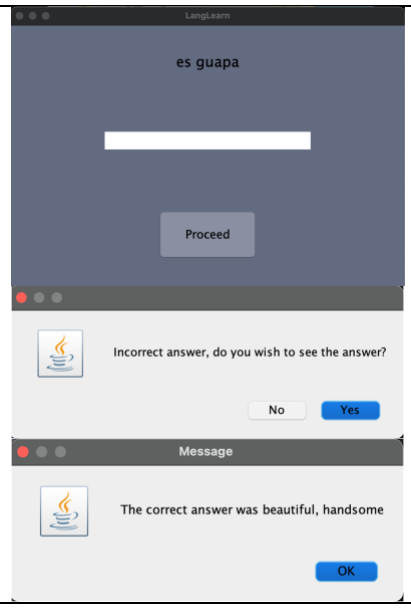
Remember and Check:

Description:	GUI demonstration:
This activity begins as followed: the program shows the front of a word but not the back. When the user decides to check and click the “Check” button, the program then shows the back of the word, and the “Check” button is changed to a “Next” button. When the user clicks it, the program presents with another word’ front and the cycle continue. This goes on indefinitely for as long as the user like before he clicks the “previous” or “Home” button.	

Code involved:

```
public void rememberrnCheckGame(JFrame window, boolean reverse) {
    createQuestionsR(qRC, reverse); // add a question
    createQuestionsR(qRC, reverse); // add a question
    getTraining (qRC.get(0), window, reverse); // create the quiz panel

}
public void getTraining(rememberCheck obj, JFrame window, boolean reverse) {
    getFrame(parentPanel2, rememberCheckFrame);
    questionRemember.setText(obj.question);
    answerCheck.setText(obj.correct_answer);
    answerCheck.setVisible(false);

    window.setContentPane(this.trainingPage);
    window.setVisible(true);

    checkButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if(!answerCheck.isVisible()){
                answerCheck.setVisible(true);
                checkButton.setText("Next");

            } else if (answerCheck.isVisible()){
                answerCheck.setVisible(false);
                checkButton.setText("Check");
                change(qRC.get(qRC.size()-1));
                createQuestionsR(qRC, reverse);
            }

        }
    });

}
void change (rememberCheck obj){
    this.questionRemember.setText(obj.question);
    this.answerCheck.setText(obj.correct_answer);
}
```

Guess and Type:

Description:	GUI demonstration:
This activity begins as followed: the program shows the front of a word. The user has to then type the back of the word correctly (with accents) and click the “Proceed” button. If the typed answer is correct, the program shows that it is correct and the user proceeds to train with the next word, else, it would show that is incorrect and ask if the user wants to see the answer. If the user chooses yes, the answer is shown, and the program presents with another word’ front and the cycle is continued afterward. This goes on indefinitely for as long as the user wants to before he clicks the “previous” or “Home” button.	

Code involved:

```java
public void guessnTypeGame(JFrame window, boolean reverse) {
    createQuestionsT(qGT, reverse);
    createQuestionsT(qGT, reverse);
    getTraining (qGT.get(0), window, reverse); // create the quiz panel

}
```

```java
public void getTraining(guessType obj, JFrame window, boolean reverse) {
    getFrame(parentPanel2,guessnTypingFrame);

    questionTyping.setText(obj.question);
    correct_answer = obj.correct_answer;

    window.setContentPane(this.trainingPage);
    window.setVisible(true);

    proceedTyping.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (textFieldTyping.getText().toLowerCase().equals(correct_answer.toLowerCase())) {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "correct answer!");
                try {
                    changeFamiliarCount(dequeueQuestionWordInd(),firstTime);
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
                change(qGT.get(qGT.size()-1));
                createQuestionsT(qGT, reverse); // add a question in the array
            } else {
                firstTime = false;
                if (JOptionPane.showConfirmDialog( parentComponent: null,  message: "Incorrect answer, do you wish to see the answer?",  title: "",
                        JOptionPane.YES_NO_OPTION) == JOptionPane.YES_OPTION) {
                    JOptionPane.showMessageDialog( parentComponent: null,  message: "The correct answer was " + correct_answer);
                    try {
                        changeFamiliarCount(dequeueQuestionWordInd(),firstTime);
                    } catch (IOException ioException) {
                        ioException.printStackTrace();
                    }
                    change(qGT.get(qGT.size()-1));
                    createQuestionsT(qGT, reverse); // add a question in the array
                }
            }
        }
```

```java
void change (guessType obj){
    firstTime = true;

    this.questionTyping.setText(obj.question);
    this.textFieldTyping.setText(null);
    this.correct_answer = obj.correct_answer;
```

Multiple choice:

Description:	GUI demonstration:
This activity begins as followed: the program shows the front of a word. The correct answer is in one of the 4 button options. If the chosen answer is correct, the user is proceeded to train with the next word, else, the selected button is disabled, and the user cannot select that button anymore. The cycle goes on indefinitely for as long as the user wants to before he clicks the “previous” or “Home” button.	

Code involved:

```java
public void multiChoiceGame(JFrame window, boolean reverse) {
    createQuestionsM(qMC, reverse); // add a question in the array
    createQuestionsM(qMC, reverse); // add a question in the array
    getTraining (qMC.get(0), window, reverse); // create the quiz panel with the first question added

}

public void getTraining(multiChoice obj, JFrame window, boolean reverse) {

    questionMultiChoice.setText(obj.question);
    option1.setText(obj.op1);
    option2.setText(obj.op2);
    option3.setText(obj.op3);
    option4.setText(obj.op4);
    correct_answer = obj.correct_answer;

    window.setContentPane(this.trainingPage);

    getFrame(parentPanel2,multiChoiceFrame);

    window.setVisible(true);

    option1.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (option1.getText().equals(correct_answer)) {
                try {
                    changeFamiliarCount(dequeueQuestionWordInd(),firstTime);
                } catch (IOException ioException) {
                    ioException.printStackTrace();
                }
                change(qMC.get(qMC.size()-1));
                createQuestionsM(qMC, reverse); // add a question in the array
            } else {
                option1.setEnabled(false);
                firstTime = false;
            }
        }
    });
```

```java
void change (multiChoice obj){
    firstTime = true;

    this.questionMultiChoice.setText(obj.question);
    this.option1.setText(obj.op1);
    this.option2.setText(obj.op2);
    this.option3.setText(obj.op3);
    this.option4.setText(obj.op4);
    this.correct_answer = obj.correct_answer;
    this.option1.setEnabled(true);
    this.option2.setEnabled(true);
    this.option3.setEnabled(true);
    this.option4.setEnabled(true);

}
```

Editing familiarity count:

During the “Guess and Type” and “Multiple Choice” activities, if the user gets the answer correct in the first attempt, the editFamiliarityCount method in the collection object is activated. familiarity count is decreased by 1(meaning that it has gotten more familiar). If the user gets the answer wrong once or more, the familiarity count is increased by 1 instead (meaning that it has gotten less familiar).

```
public void changeFamiliarCount(int wordIndex, boolean firstTime) throws IOException {
    // edit the word's familiarity(lower = more familiar)
    if (firstTime){
        if(collectionSelected.familiarCount.get(wordIndex) > 1) {
            collectionSelected.editFamiliarCount(wordIndex, value: -1);
        }
    } else if(!firstTime) {
        collectionSelected.editFamiliarCount(wordIndex, value: 1);
    }
}
```

Bibliography:

"Bufferedreader." *Java Platform SE 8*, 5 Jan. 2022, https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html.

"Bufferedwriter." *Java Platform SE 8*, 5 Jan. 2022, https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html#newLine--.

"CardLayout." *Java Platform SE 7*, 24 June 2020, https://docs.oracle.com/javase/7/docs/api/java/awt/CardLayout.html.

Jumba, Isaac. "Why Use Bufferedreader and Bufferedwriter Classses in Java." *Medium*, Medium, 23 Sept. 2015, https://medium.com/@isaacjumba/why-use-bufferedreader-and-bufferedwriter-classses-in-java-39074ee1a966.

"Stacks and Queues." *Everything Computer Science*, 2013, https://everythingcomputerscience.com/discrete_mathematics/Stacks_and_Queues.html#:~:text=Stack%20is%20a%20container%20of,%2Dout%20(FIFO)%20principle.

Word count: 1059