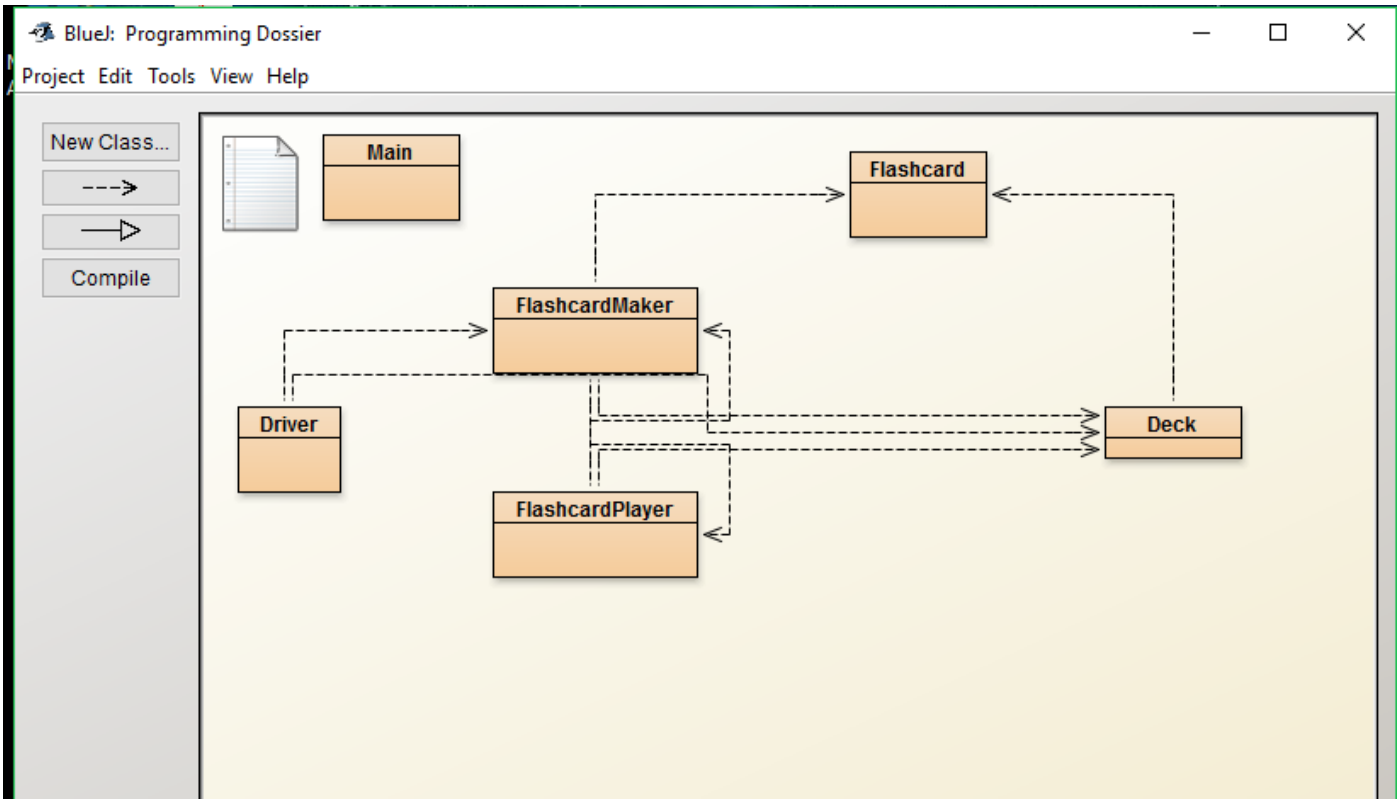
Criterion C: Development

Techniques used to create the program

- Program structure – explanation and justification, tables related to structure/GUI development
- Usage of Swing to develop a GUI
- Use of array/array manipulation to store cards and vocab

Program Structure



Allow for use of Vocab Practice

The entire program is based around 5 classes and a driver to execute them. They are as follows:

- Flashcard – This class consists of the structure of the flashcard in itself – it has an untranslated word and a translated word String declared, with getter and setter functions to set the untranslated/translated words, respectively.

- Deck – Another simple class like Flashcard, it serves to keep track of how many flashcards have been added in the process. It has setters and getters here as well to keep track of how many you got correct and incorrect at the end of the process. It also develops the arraylist that holds the cards and all of their independent values in untranslated and translated words. If the length of the untranslated and translated words exceed 0, then they can be added as a flashcard.
- FlashcardMaker – This class serves to finally take advantage of Swing. Developing a simple GUI in Java is something new, and so as a challenge a lot of what went into this was developing that GUI in a very simple but attractive way. How the GUI is developed will be touched on further in Criterion C. The purpose of this class is to develop the actual card making portion of the program – you can create cards with the untranslated word that will be shown, and then have the translated portion that will be shown after you input the correct answer. It will add it to the given arraylist, and store cards until the user is ready to quiz themselves on the cards added. It builds the menu bar at the top with an option to begin the exam. Once it begins, it closes itself and uses the data from the arraylist in the FlashcardPlayer class.
- FlashcardPlayer – This class takes all the arraylist data stored in Deck and puts it to fruition. A GUI is developed again with Swing, which will be touched on in the next section. This is a fairly simple class, after the GUI is set up which is similarly structured to the previous menu, it displays each value from the arraylist and then allows you to choose if your answer was correct or incorrect. It keeps a counter for each value as you press the correct or incorrect buttons. Once you have played every card in the arraylist, it concludes the test and displays in the same window the results. From there you are free to quit the program and practice again as necessary.

- Main – This class simply serves to run the rest, as it is opened by the Driver class and initiates the FlashcardMaker class. It servers to greet the user and inform them what this product is from the start. Once they click the “Begin” button, the rest functions as it ensues.

These are all strongly interconnected. The Flashcard class is implemented into the Deck class, and the Deck class is used in the FlashcardMaker and FlashcardPlayer classes as a way to use the arraylist of Flashcards to know what words are to be implemented in the exam.

The driver, finally, is simply there to begin the program. It calls for FlashcardMaker, which runs its build and brings up the GUI. The rest of the GUI is developed in FlashcardMaker and into FlashcardPlayer.

Usage of Swing to develop a GUI

Developing a GUI in Java posed the biggest challenge for this program, as from what we have learned of Java developing a GUI was not a large part of it, with more emphasis on the code itself rather than developing a final product. Part of developing this was learning how to utilize Swing as a way to develop it. This mainly pertains to the “FlashcardMaker” and “FlashcardPlayer” classes.

To start, the most important part of using Swing was to consider the order of the build. In the order it was implemented is how it would be presented, so I planned a design for how I wanted this program to look. The frame and bar were most important, then from there, the menu bar was built and going from top to bottom the labels and text areas and button were added.

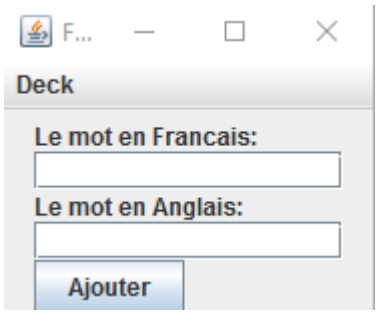
```
void run() {
    buildFrame();
    buildBackground();
    buildBar();
    buildLabels(new JLabel("Le mot en Francais:"));
    buildTextArea(qText);
    buildLabels(new JLabel("Le mot en Anglais:"));
    buildTextArea(aText);
    buildButtons();
    displayFrame();
}
```

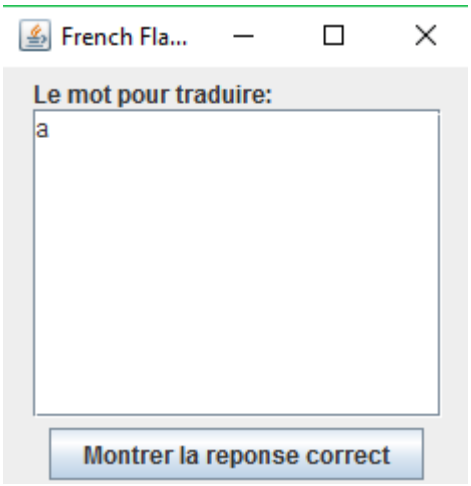
This was the first step to implementing each GUI for both classes. They call the functions later to give the buttons and labels the correct alignment and such as necessary. With this order done, I was able to go into each step one by one and implement each tool to fulfill a necessary function.

One surprising perk of Swing was how customizable it was. I experimented with various orientations and alignments to get the most simple, clean and effective looking product.

```
void run(){
    buildFrame();
    buildBackground();
    buildLabels();
    buildTextArea();
    buildButtons();
    displayFrame();
}
```

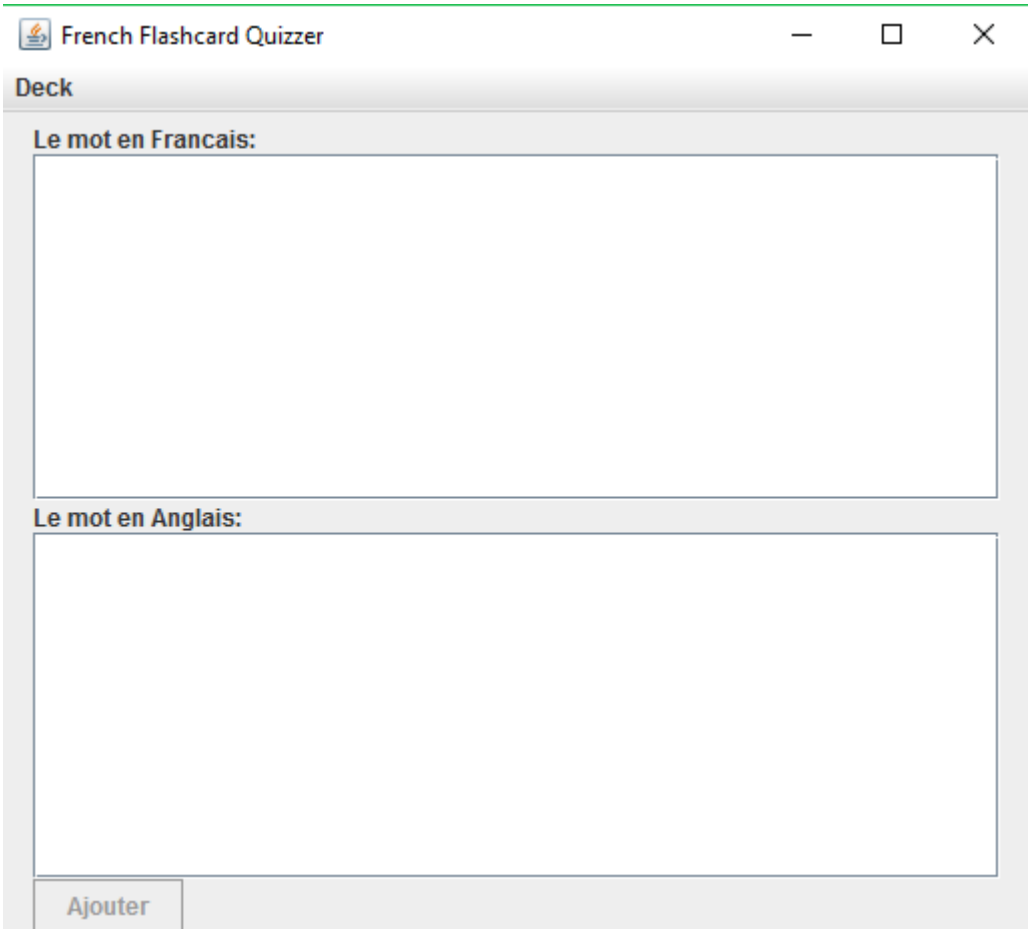
This was the result:





Allow for customizable use

One feature I wanted to add to allow for more customizable use was the fact that the boxes could be expanded. Perhaps the user would need to use a paragraph or something of the sort, or multiple words at once. Alignment was used to keep that in mind and allow the user to have their needs met. For example:



The purpose of taking advantage of Swing's extensive customizability was to allow for this. By using alignment it makes for adjustable features without drastically affecting the rest of the structure. The same works for the FlashcardPlayer class.

Functionality was kept in mind while choosing the structure here. I designed the program specifically to prevent accidents.

- The "add card" button is specifically at the bottom to represent that it's the last step after making one, to keep the flow of how information in the program is processed linear.

- Once you finish making a deck, the button to begin practicing is at the top – it's out of the way to prevent accidentally clicking it and forcing you to start even though you haven't added the necessary cards yet.

- Using a menu bar for the beginning of the examination means it has to be clicked twice in order to begin, to further prevent accidents.

- For the FlashcardPlayer class, again, buttons are kept at the bottom to sort of finalize the process, to show the answer and address if the answer was correct or not.

Another important part of using Swing was the ability to layer objects in a way, where they could appear when necessary and also disappear when they were not. This was crucial for the buttons especially in the FlashcardPlayer. Instead of creating another window with entire new buttons, what was more optimal was to simply layer them in and out between button presses. Once all of the cards had been used, the Yes/No buttons disappear:

```
private void showFinish(){
    label.setText("Les resultes:");
    textArea.setText("Tu as " + deck.getNumCorrect() + " correct et " + deck.getNumWrong() + " pas correct.");
    answer.setText("Finis");
    answer.setVisible(true);
    correct.setVisible(false);
    incorrect.setVisible(false);
    cardCountCompleted++;
}
```

This is the function for when the Show Results button is pressed. The label in the box changes, it displays the number of correct and incorrect cards according to the user and then sets the button text to simply show “Finish.” The other two buttons become invisible. This allows for a simplified code that is less taxing on the processing system.

Use of array/array manipulation to store cards and vocab

- The entire use of storing the cards is based around the array list “flashcardList.” Arraylists are an optimal choice here since the amount of cards isn’t set, and there is no reason for it

to be since the user may store as many as they need to.

- An array list also functions to allow for multiple of the same element, so if the user feels the need to practice a word multiple times, it is available.