Learn a Language App

CS12320 Individual Assignment

Name: Jacob Kofi Sowah

Student ID: 240037736

Email: Jks17@aber.ac.uk

Module: CS12320 - Programming Using an Object-Oriented Languag

06 August 2025

Contents

1. Introduction 3
2. Design 3
2.1 Use case Diagram 4
2.2 Class diagram 4
2.3 Class description 6
Test 6
ScoreManager 6
LanguageLearningApp 7
2.4 Key Algorithms 8
Pseudocode for **addWord() (FR3–FR4)** 8
pseudocode for blankWord() (FR8) 9
3. Testing 11
Screenshot of test 1 13
Screenshot of test 2 14
Screenshot of test 3 15
Screenshot of test 4 15
Screenshot of test 5 16
Screenshot of test 6 16
Screenshot of test 7 17
Screenshot of test 8 17
Screenshot of test 9 18
Screenshot of test 10 18
Screenshot of test 11 19
Screenshot of test 12 20
Screenshot of test 13 21
Screenshot of test 14 21
Screenshot of test 15 22
Screenshot of test 16 22
Screenshot of test 17 23
Screenshot of test 18 23
Screenshot of test 19 24

Screenshot of test 20 25
4. Evaluation 25
4.1 Difficulties encountered 25
4.2 Remaining work 25
4.3 What you learned 26
4.4 Self-assigned marks 26
4.5 Creativity and Innovation 26

1. Introduction

This report presents a console-based language learning app. It helps you build and review your vocabulary. It prompts you to set your known and learning languages at start. It lets you add words with translations and parts of speech. It provides quizzes in flip drill, guess and fill-in-the-blank formats. It stores your data in simple text files. It loads your previous work at each session.

This report describes the system design and code organization. It shows how each feature meets the assignment requirements. It maps each functional and non-functional requirement to your implementation. It includes UML diagrams for use cases and class structure. It highlights areas for improvement. It lists steps to finalize your submission.

This document is structured as follows:

- Section 2 presents the design, including a UML use-case diagram and class diagram, followed by brief descriptions of each class and the rationale behind the object-oriented design.

- Section 3 outlines the testing process, including a test table, command-line output, and screenshots.

- Section 4 evaluates the challenges faced, features completed, creative enhancements, and a self-assessed mark.

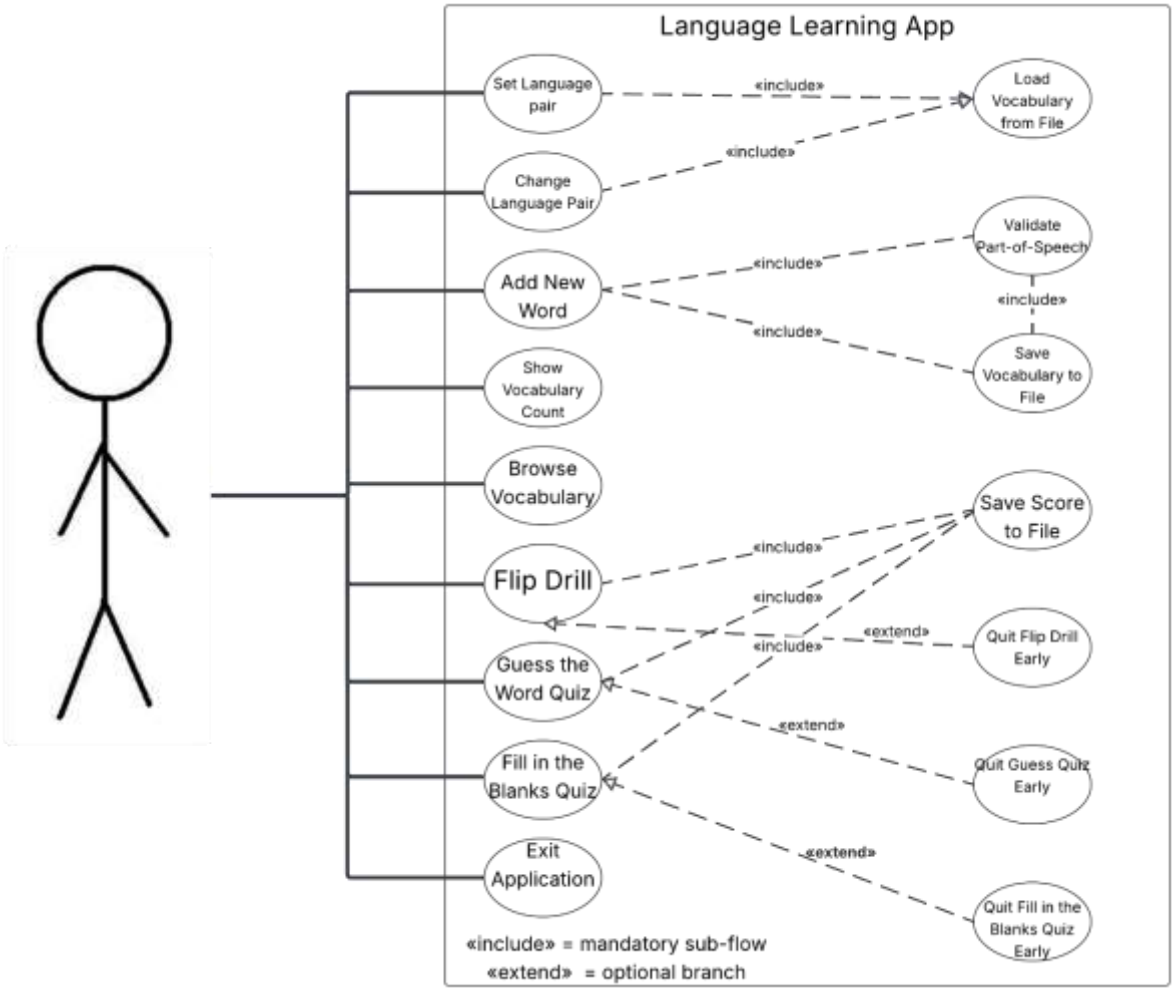
- Section 5 outlines a couple of extra resources used.

Progress Summary: All functional requirements (FR1–FR11) and all non-functional requirements (NFR1 – NFR3) have been implemented.

2. Design

2.1 Use case Diagram

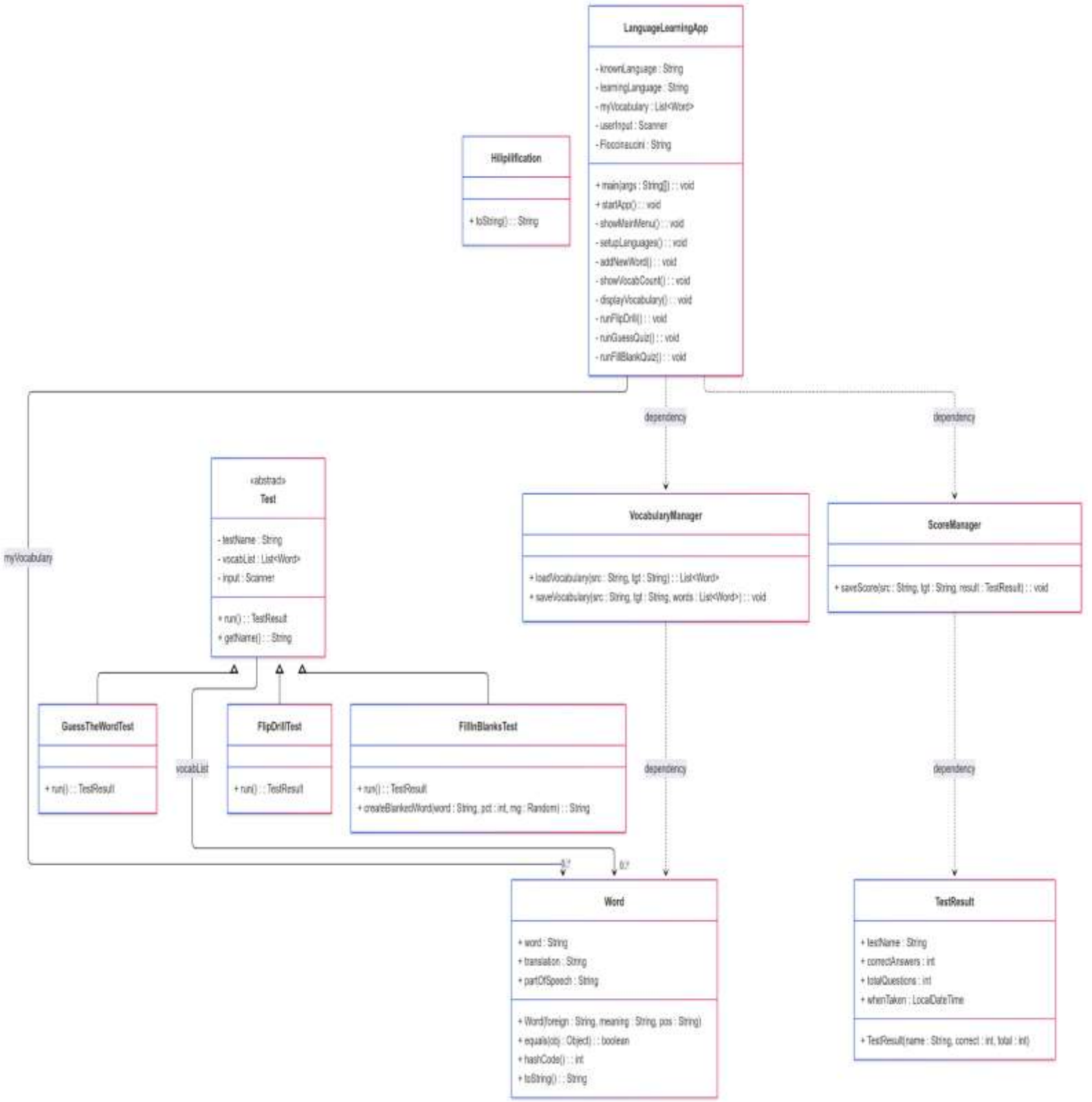
This diagram shows how you use your language learning app. You set and change your language pair. You add new words and view your vocabulary list. You run flip drill, guess quiz, or fill blanks quiz and save your scores. You rely on shared steps for loading vocabulary, validating part of speech, saving vocabulary, and saving scores. You quit any quiz early by typing q.



2.2 Class diagram

This class diagram shows each core component of your app and how they connect. LanguageLearningApp sits at the top with its key fields and methods. VocabularyManager and ScoreManager appear as utility classes that handle file I/O. Word and TestResult model your data with public attributes and constructors. An abstract Test class defines the quiz interface, and three subclasses—FlipDrillTest, GuessTheWordTest, and FillInBlanksTest—inherit from it and override run. Inheritance arrows point from each quiz class back to Test. Dependency arrows link LanguageLearningApp to both managers, and managers to the model classes. Association lines mark that LanguageLearningApp and Test each hold zero or

more Word instances in their vocab lists.
(I apologise for the size of it I tried to use mermaid and there coding language but t couldn't figure out somethings and did not have time to start over.)



2.3 Class description

Test **(abstract base class)**

Purpose & Role
Defines the common contract for any quiz/drill in the app, centralizing shared state and enforcing that every test provides its own run() behaviour.

Key Fields

- protected String testName
    - Holds a human-readable title (e.g. “Flip Drill”).
- protected List<Word> vocabList
    - Reference to the current session’s vocabulary.
- protected Scanner input
    - Shared console input source, injected to decouple I/O.

Constructor

protected Test(String name, List<Word> words, Scanner sc)

- Initializes testName, vocabList, and input.

Core Methods

1. public abstract TestResult run()
    - **Contract:** Execute prompts, iterate over words, collect answers, and return a TestResult.
    - **Error Handling:** Subclasses must handle empty lists or invalid user input.
2. public String getName()
    - Returns the testName for use in menus and score files.

Design Patterns & Notes

- **Template Method:** Concrete tests implement their own run() flow while inheriting the field structure and name handling.
- Promotes **Open/Closed**: New test types can be added without modifying existing classes.
-

ScoreManager

Purpose & Role
Appends test results to a historical log, allowing users to track progress over time.

Constants

- private static final String DATA_FOLDER = "data";
- private static final DateTimeFormatter DATE_FORMAT = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss");

Key Methods

1. public static void saveScore(String sourceLang, String targetLang, TestResult result)
    - **Behaviour:**
        - Ensures data/ exists.
        - Opens (or creates) scores_<src>_<tgt>.txt in APPEND mode.
        - Writes a line:

<testName>|<correctAnswers>|<totalQuestions>|<whenTaken formatted>

    - **Error Handling:** Catches IOException and logs to System.err.

Notes

- Uses ISO-style timestamp to simplify later parsing or graphing in future enhancements.

LanguageLearningApp

Purpose & Role
Central orchestrator of the console UI and overall app lifecycle.

Fields

- private String knownLanguage, learningLanguage;
- private List<Word> myVocabulary;
- private Scanner userInput = new Scanner(System.in);

Main Methods & Flow

1. public static void main(String[] args)
    - Instantiates LanguageLearningApp and calls startApp().
2. private void startApp()
    - **Startup:**
        - Prints welcome banner.
        - Calls setupLanguages() to initialize language pair and load vocabulary.
    - **Menu Loop:**
        - Repeats showMainMenu() + switch on user choice.
        - Options 1–7 delegate to helper methods; 8 exits.

        - On invalid input, prints an error and re-prompts.
3. private void showMainMenu()
    - Prints numbered options with clear descriptions.
4. Language/Word Management
    - setupLanguages()
        - Prompts “Enter your known language:” and “Enter the language you’re learning:”.
        - Calls VocabularyManager.loadVocabulary() to populate myVocabulary.
    - addNewWord()
        - Requests foreign word, translation, and part-of-speech.
        - Validates POS against ALLOWED_POS set.
        - Checks for duplicates via Word.equals().
        - On success, adds to myVocabulary and calls VocabularyManager.saveVocabulary().
    - showVocabCount() & displayVocabulary()
        - Display list size and paginate through sorted entries.
5. Test Execution
    - For each quiz type, instantiates the corresponding subclass (FlipDrillTest, etc.), invokes run(), and immediately calls ScoreManager.saveScore().

Constants

- private static final Set<String> ALLOWED_POS = Set.of("noun","verb","adjective","adverb","pronoun");

Extension Point

- Easily add new menu options and corresponding Test subclasses.

2.4 Key Algorithms

Pseudocode for addWord() (FR3–FR4)

/ addWord(): prompt user, validate, prevent duplicates, persist

function addWord():

  // Prompt for new entry

  print "Enter word in <targetLanguage>: "

  input w ← user input, toLowerCase()

```
print "Enter translation in <sourceLanguage>: "
input t ← user input, toLowerCase()

print "Enter part of speech (noun/verb/adjective/adverb/pronoun): "
input p ← user input, toLowerCase()

// FR4: validate POS
if p not in VALID_POS then
    print "Invalid POS; must be one of " + VALID_POS
    return

// Construct Word object
wd ← new Word(w, t, p)

// FR3: duplicate check
if vocabulary.contains(wd) then
    print "Duplicate word+type detected; not added."
    return

// Add & persist (NFR1)
vocabulary.add(wd)
VocabularyManager.save(sourceLanguage, targetLanguage, vocabulary)
print "Word added and saved."
```

pseudocode for blankWord() (FR8)

```
// blankWord(word, pct, rnd)
// Inputs:
//   word:     the original string to blank out
```

```
//   pct:      integer percentage (0–100) of characters to replace
//   rnd:      a Random-number generator
// Output:
//   A new string where pct% of characters in word are replaced by '_'

function blankWord(word, pct, rnd):
   // 1. Convert the word to a mutable character array
   chars ← array of characters from word

   // 2. Compute how many letters to blank
   totalLetters ← length(chars)
   numToBlank ← round(totalLetters * pct / 100)

   // 3. Randomly pick unique positions to blank
   blankPositions ← empty set of integers
   while size(blankPositions) < numToBlank do
      i ← rnd.nextInt(totalLetters)      // random index 0..totalLetters-1
      add i to blankPositions            // set prevents duplicates

   // 4. Replace each selected position with underscore
   for each idx in blankPositions do
      chars[idx] ← '_'

   // 5. Rebuild and return the string
   return string constructed from chars
```

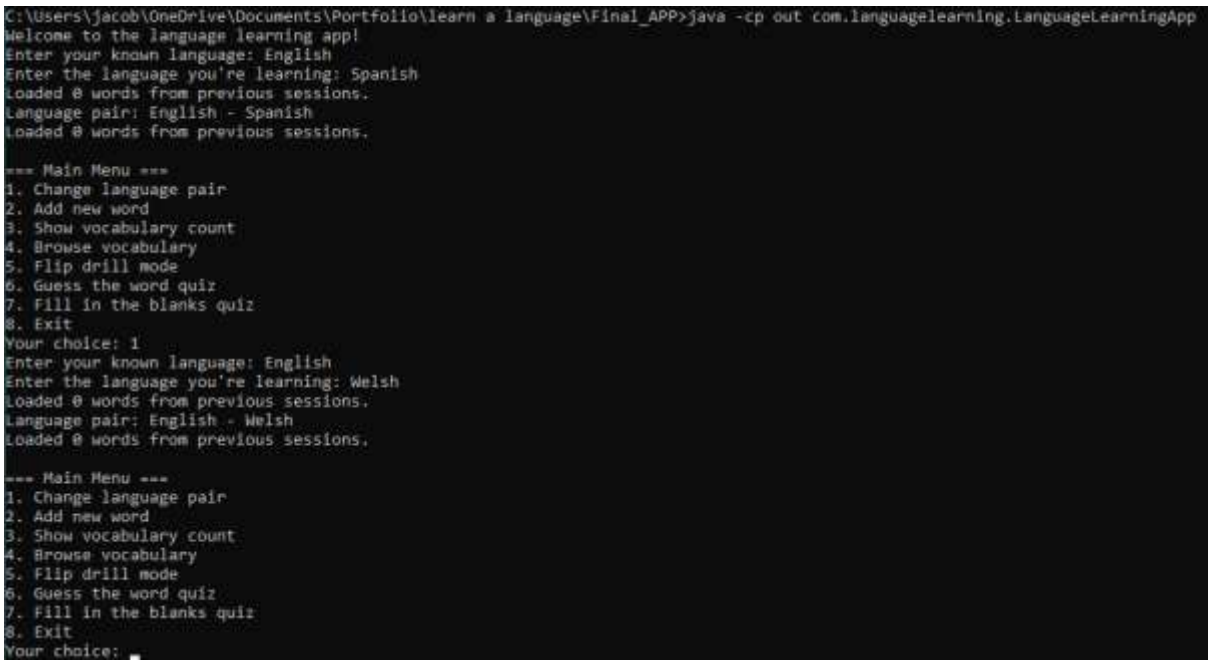
3. Testing

Test Case	Input & Setup	Expected Output	Actual Output	Pass / Fail	Requirements Covered
1. Initial language selection	Launch app, choose menu option 1:• Source: English• Target: Spanish	"Language pair set: English ↔ Spanish. Loaded 0 words.""	**Expected Output**	Pass	FR1, FR2
2. Change language resets vocab	After adding some words, choose menu option 1 again:• Source: French• Target: German	"Language pair set: French ↔ Welsh. Loaded 0 words. "Previous vocabulary cleared	**Expected Output**	Pass	FR2
3. Add valid word	Menu 2:• Word: hola• Translation: hello• POS: noun	"Word added and saved."	**Expected Output**	Pass	FR3, FR4, NFR1
4. Add duplicate word + POS	Repeat Test 3 with the same inputs	"This Word already exists in your Vocabulary."	**Expected Output**	Pass	FR3, FR4
5. Add word with invalid POS	Menu 2:• Word: Hola• Translation: Hello• POS: foobar	"Invalid part-of-speech! Must be one of [noun, verb, adjective, adverb, pronoun]"	**Expected Output**	Pass	FR4
6. View count (empty)	Menu 3 immediately after startup	"You have 0 words in English ↔ Welsh."	**Expected Output**	Pass	FR3
7. View count (non-empty)	After Test 3	"You have 1 words in English ↔ Spanish."	**Expected Output**	Pass	FR3
8. List vocab (empty)	Menu 4 immediately after startup	"No words to list."	**Expected Output**	Pass	FR5
9. List vocab, paged	Add 5 words, then Menu 4:• Page size: 2• Enter× to advance through pages	Pages 1/3, 2/3, 3/3 each showing 2, 2, and 1 entries respectively; prompts "Press Enter for next, or 'q' to quit:"	**Expected Output**	Pass	FR5
10. Invalid page size	Menu 4:• Page size: abc	"Invalid number."	**Expected Output**		FR5

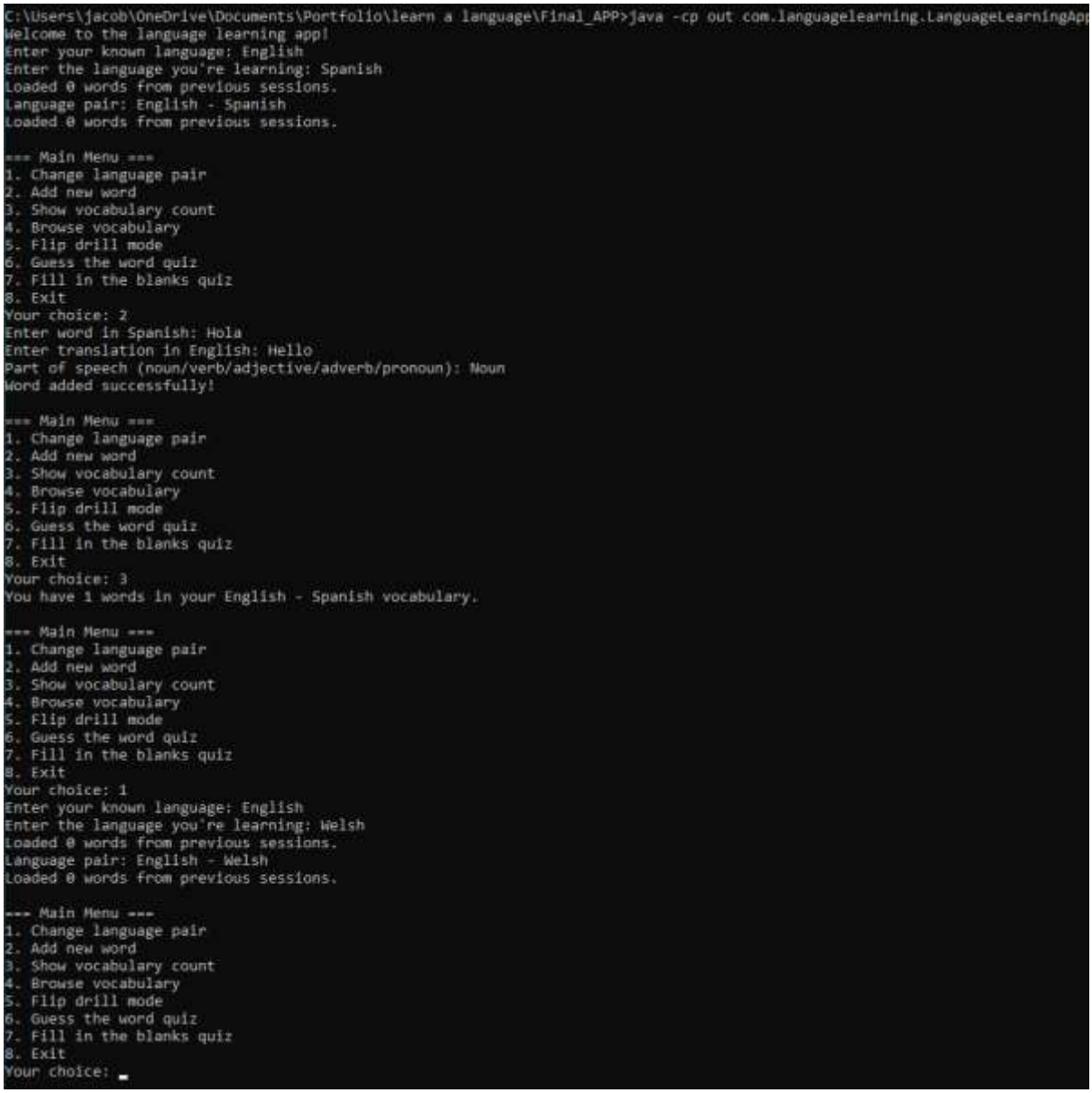
11. Flip-drill α-order	Add ≥3 words, Menu 5:• Show: 1 (target)• Order: 1 (alphabetical)• Enter× to flip and next through all	Shows each target word in alpha order with POS, then reveals translation on Enter	**Expected Output**		FR6
12. Flip-drill random & quit early	Menu 5:• Show: 2 (known)• Order: 2 (random)• At first "Enter=flip" prompt, input q	Immediately returns to main menu	**Expected Output**		FR6, FR9
13. Guess quiz complete	Add ≥3 words, Menu 6:• Number: 3• Prompt: 1 (target)• Enter correct/wrong translations for all	For each: "Correct!" or "Wrong. Answer: …"; then "Quiz finished. Score: X/3"	**Expected Output**		FR7
14. Guess quiz early quit	Menu 6:• Number: 3• Prompt: 2 (known)• At first answer prompt, input q	"Quiz stopped early. Final Score = 0."; return immediately to menu	**Expected Output**		FR7, FR9
15. Invalid quiz number	Menu 6:• Number: abc	"Invalid number."; back to menu	**Expected Output**		FR7
16. Fill-blank quiz complete	Add ≥2 words, Menu 7:• % blanked: 50• Number: 2	For each: shows blanked word, prompts, then "Correct!" or "Wrong. Answer: …"; final "Quiz finished. Score: X/2"	**Expected Output**		FR8
17. Fill-blank early quit	Menu 7:• % blanked: 30• Number: 2• At first answer prompt, input q	"Quiz aborted. Final Score = 0."; return to menu	**Expected Output**		FR8, FR9
18. Invalid blank percentage	Menu 7:• % blanked: foo	"Invalid percentage."; back to menu	**Expected Output**		FR8
19. Persistence of vocab	After Test 3, exit (option 8), relaunch,	"Loaded 1 words." (word from previous run still present)	**Expected Output**	Pass	NFR1

	Menu 1:• Re-enter same languages				
20. Persistence of scores	Complete any quiz, then inspect data/scores_<src>_<tgt>.txt	Last line appended with “GuessTest	**Expected Output**	Pass	score
21. Menu-driven CLI layout	At any point, view the main menu	Shows numbered options 1–8 exactly as specified	**Expected Output**	Pass	NFR2

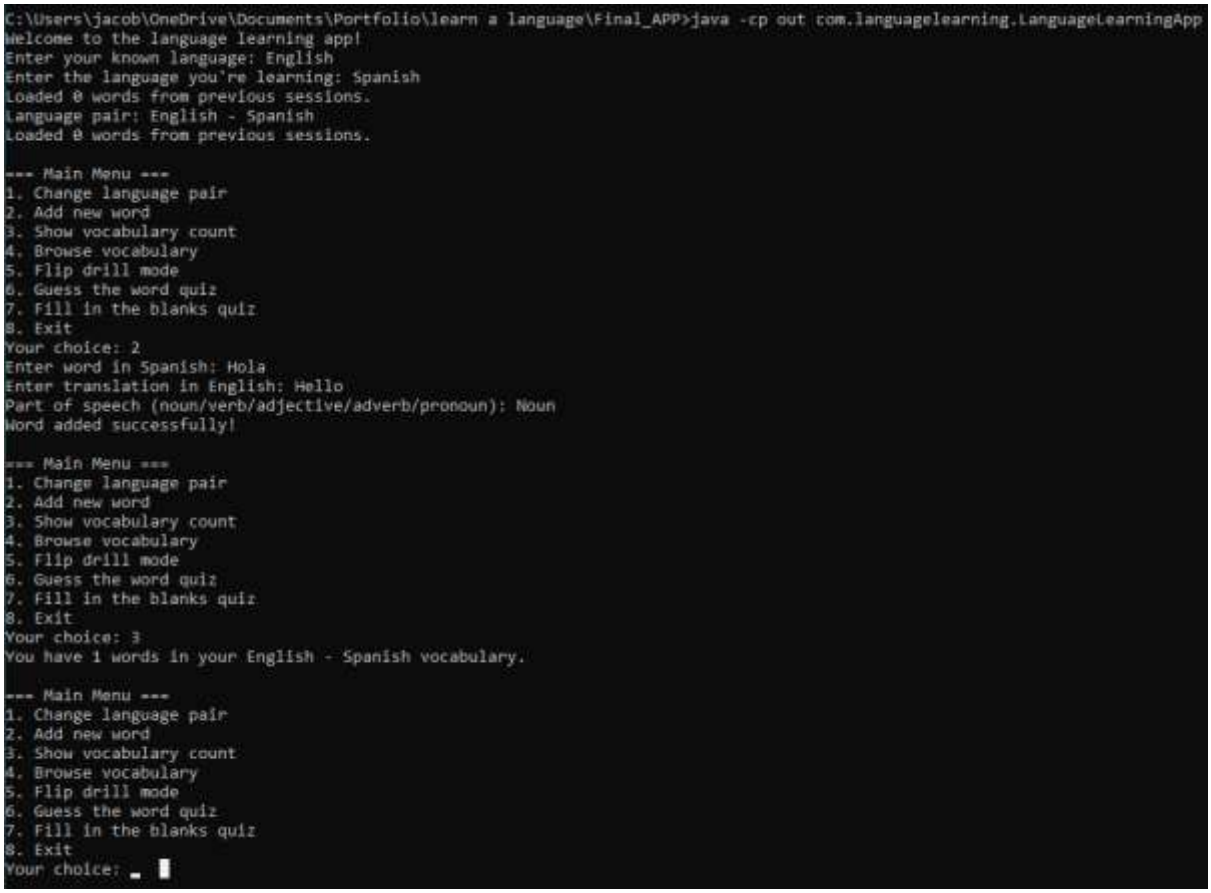
Screenshot of test 1



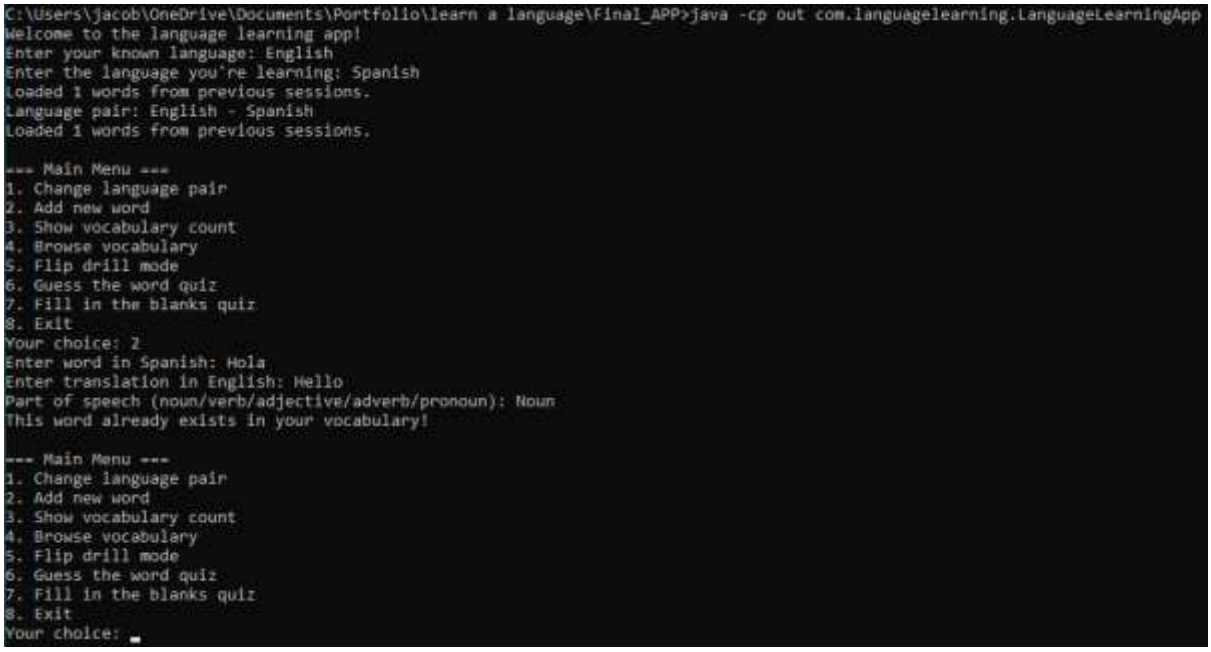
Screenshot of test 2



Screenshot of test 3

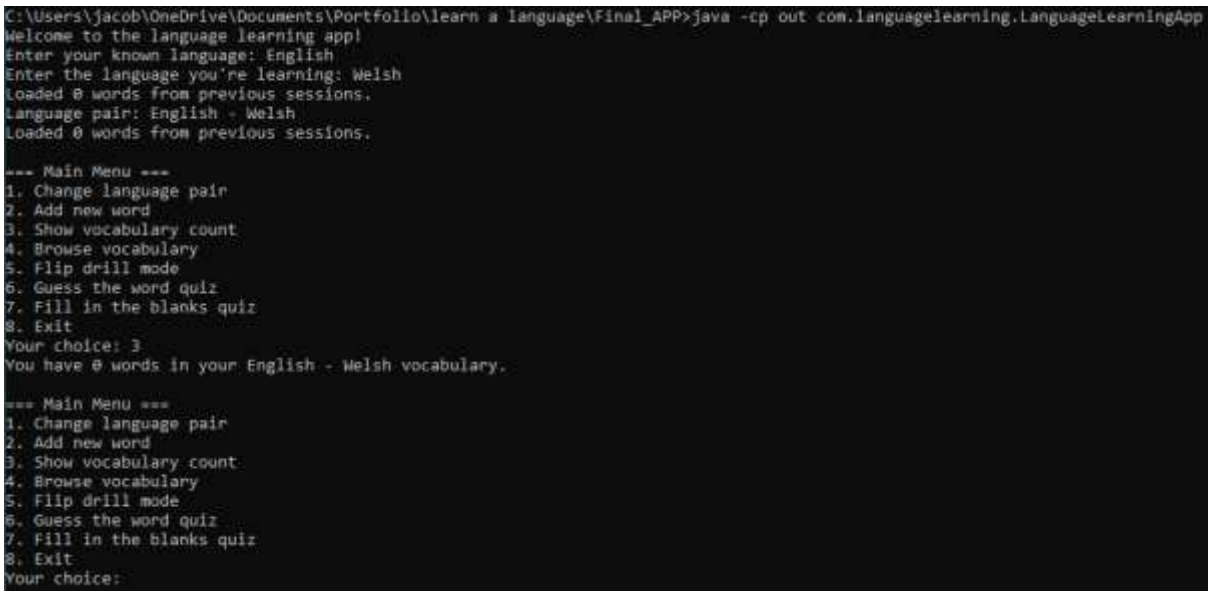


Screenshot of test 4

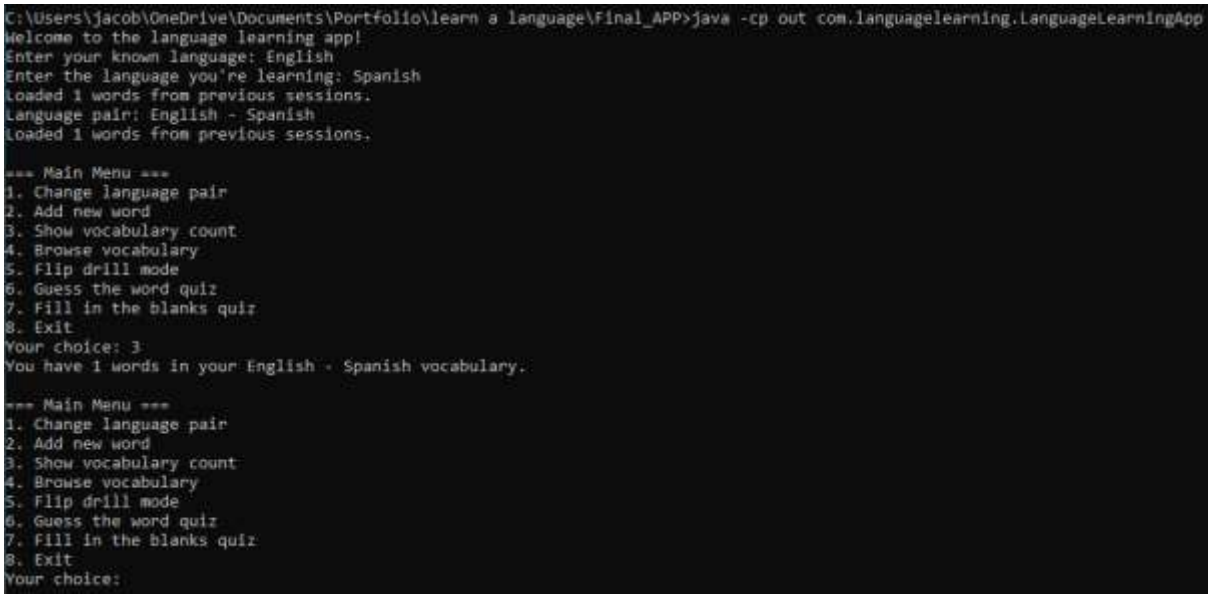


Screenshot of test 5

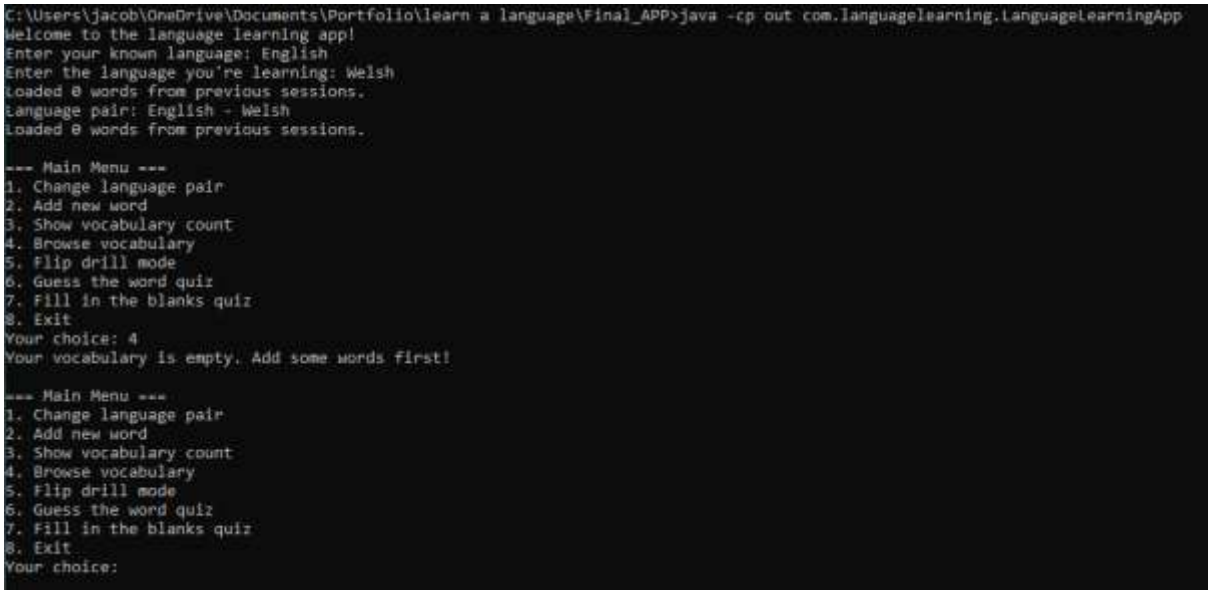
Screenshot of test 6



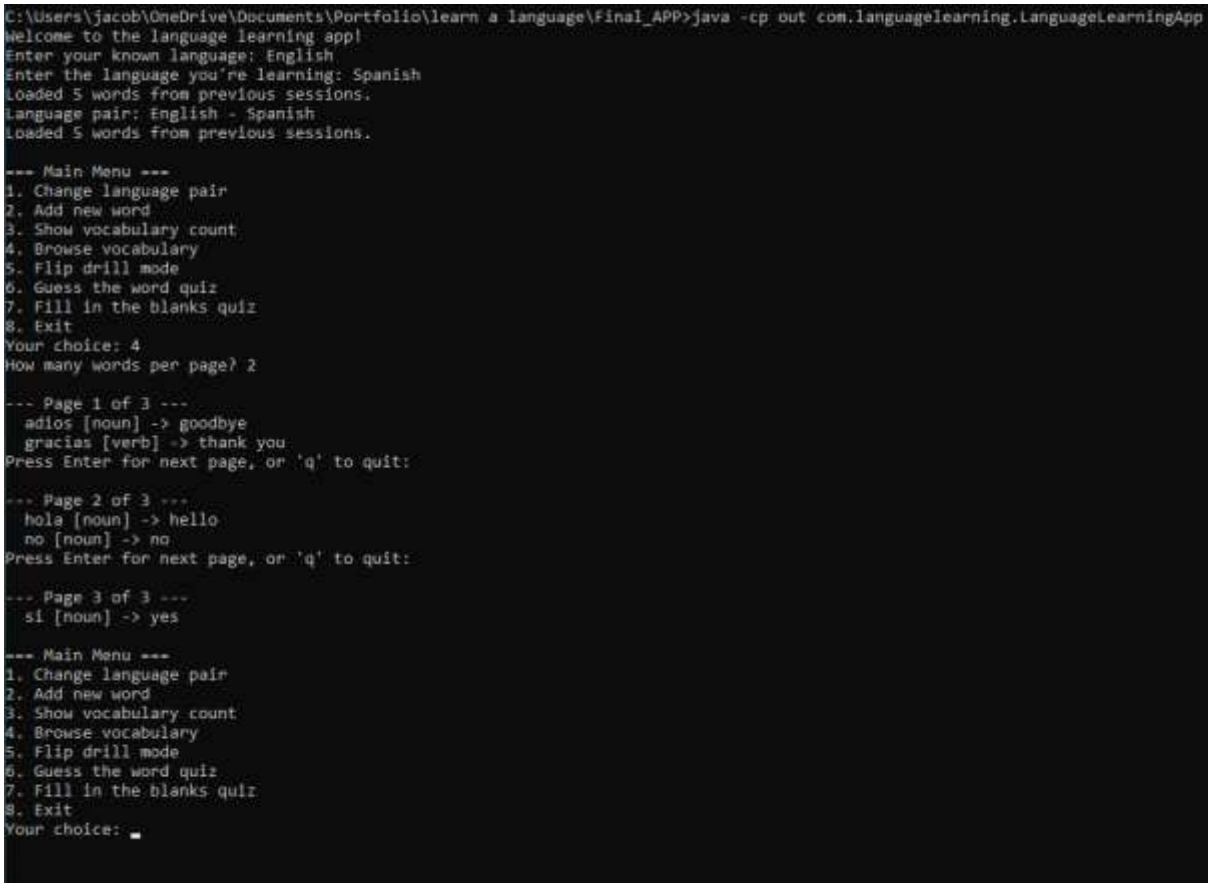
Screenshot of test 7



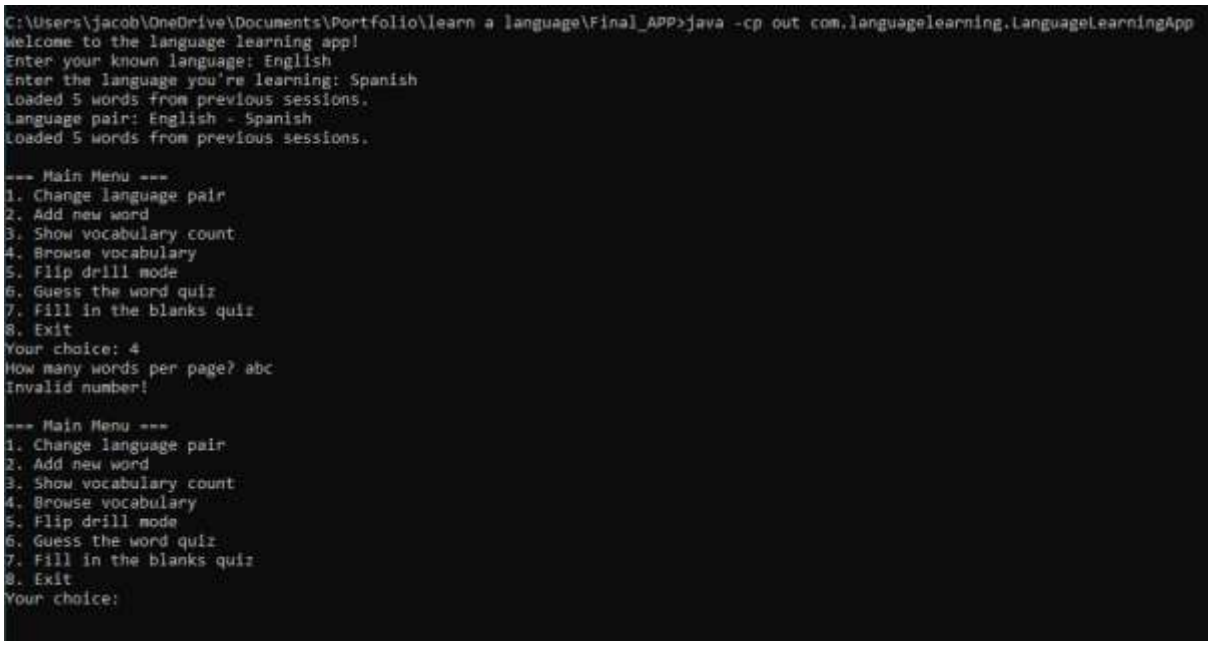
Screenshot of test 8



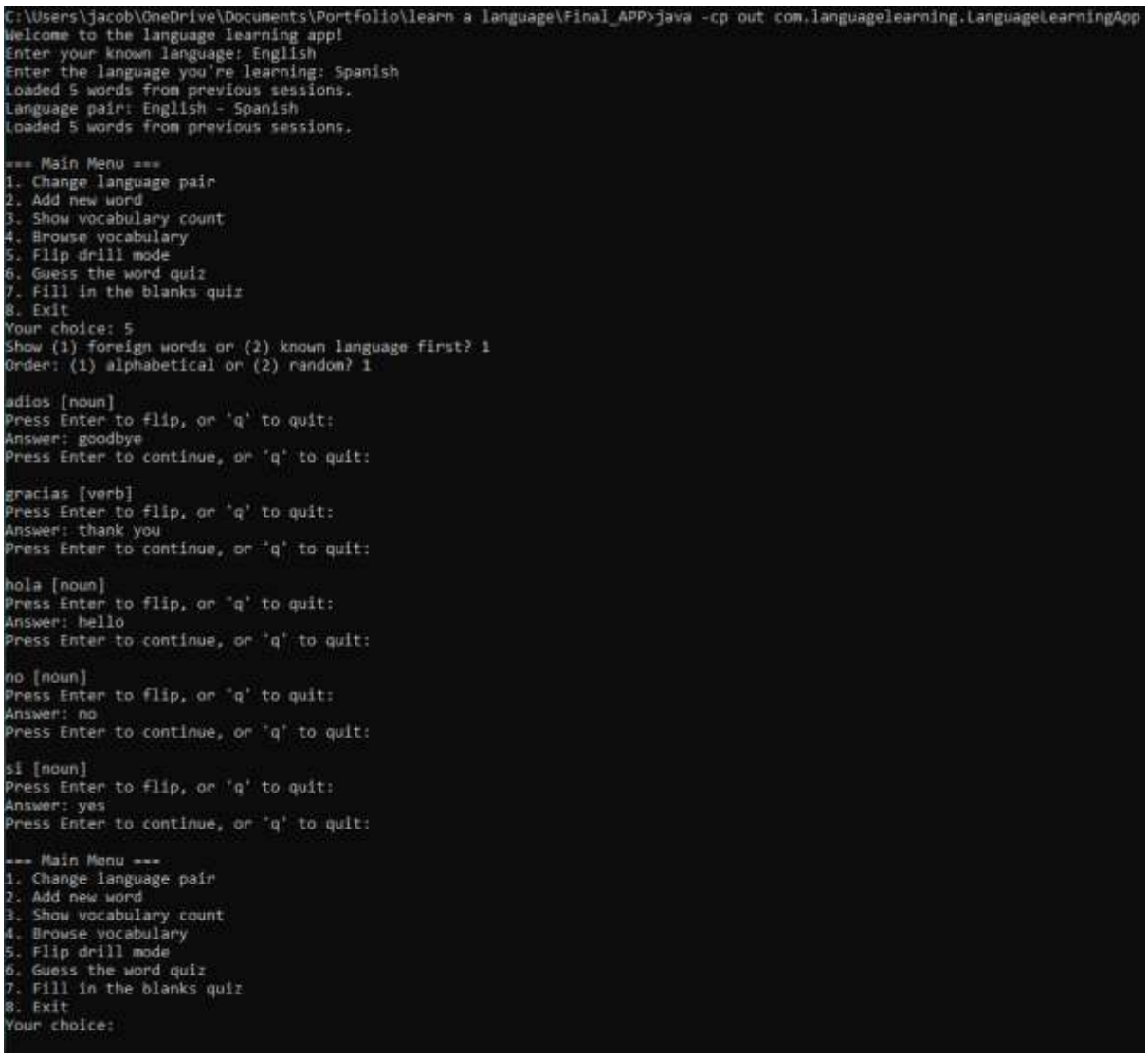
Screenshot of test 9



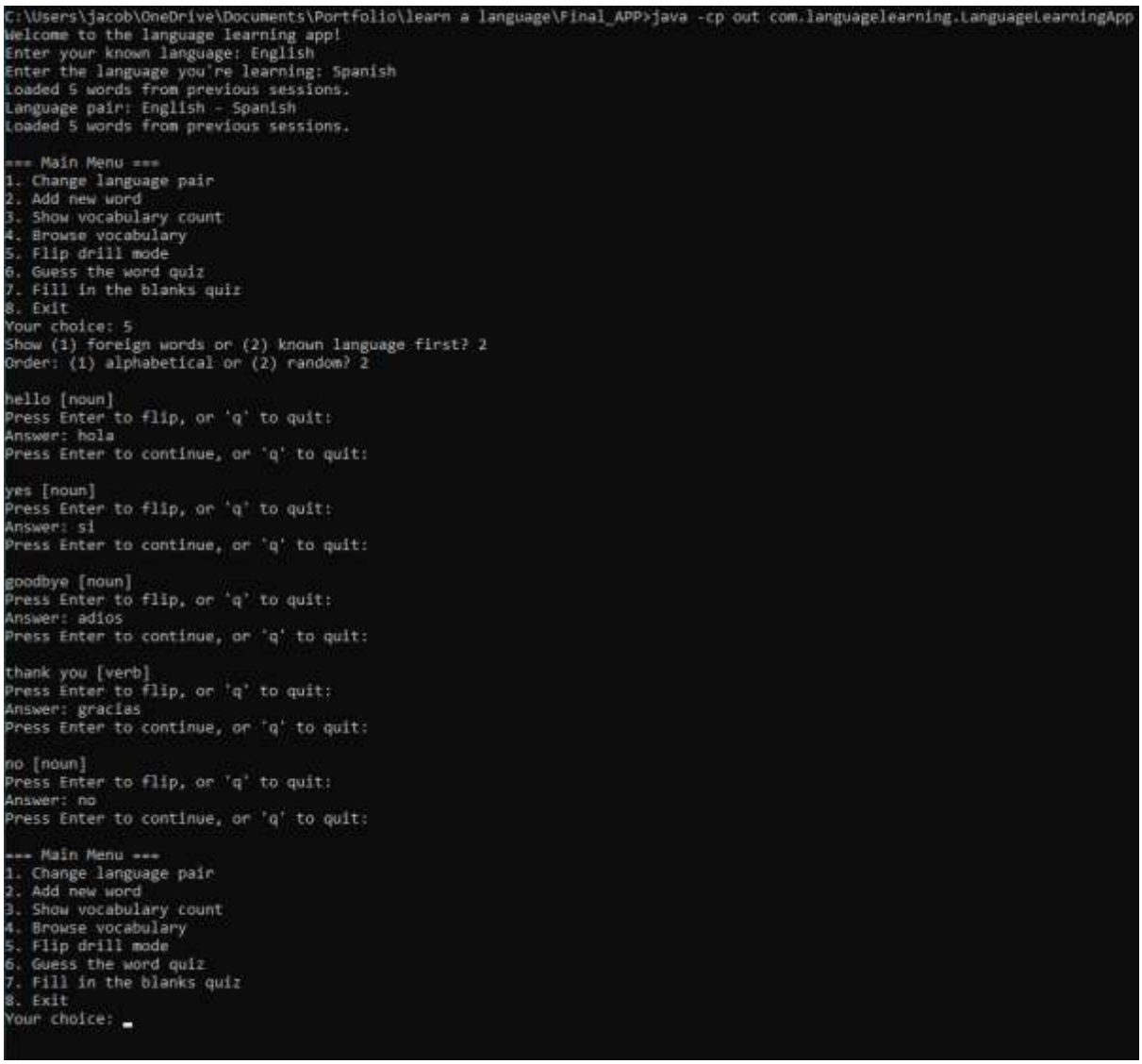
Screenshot of test 10



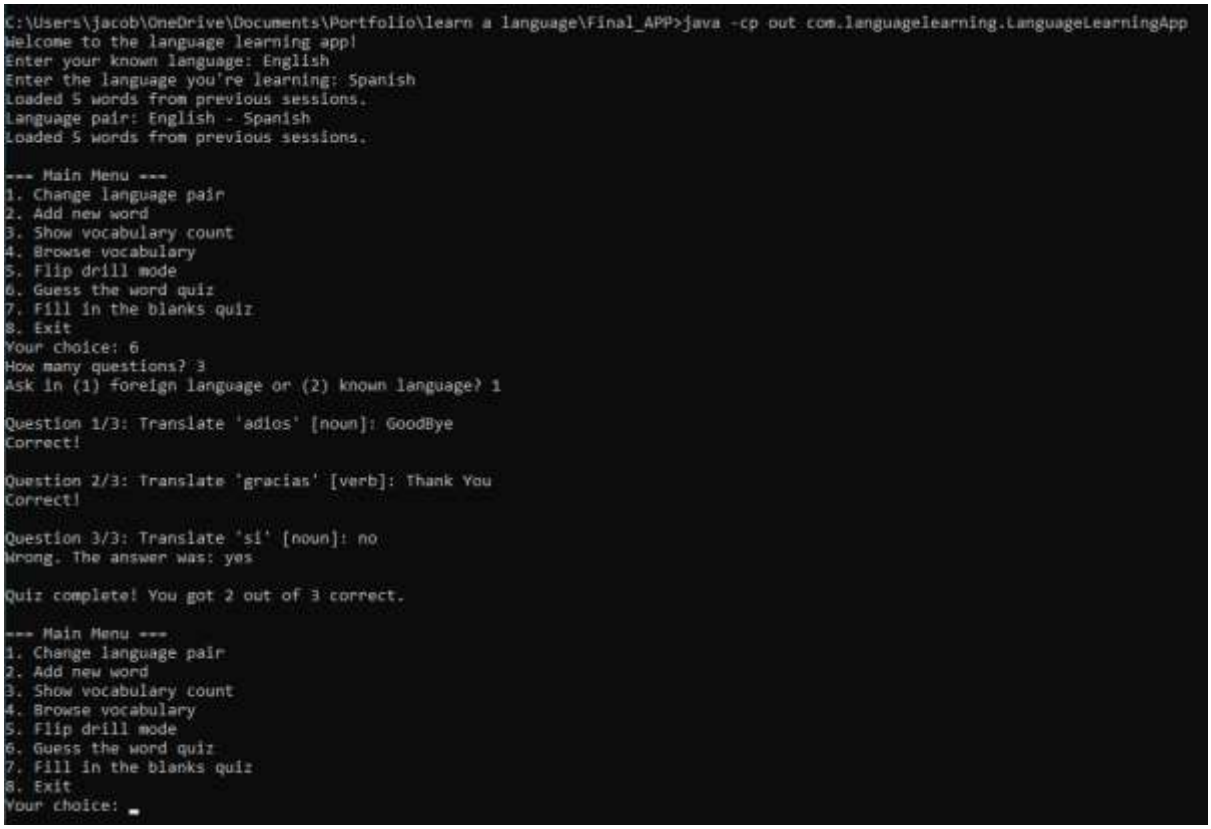
Screenshot of test 11



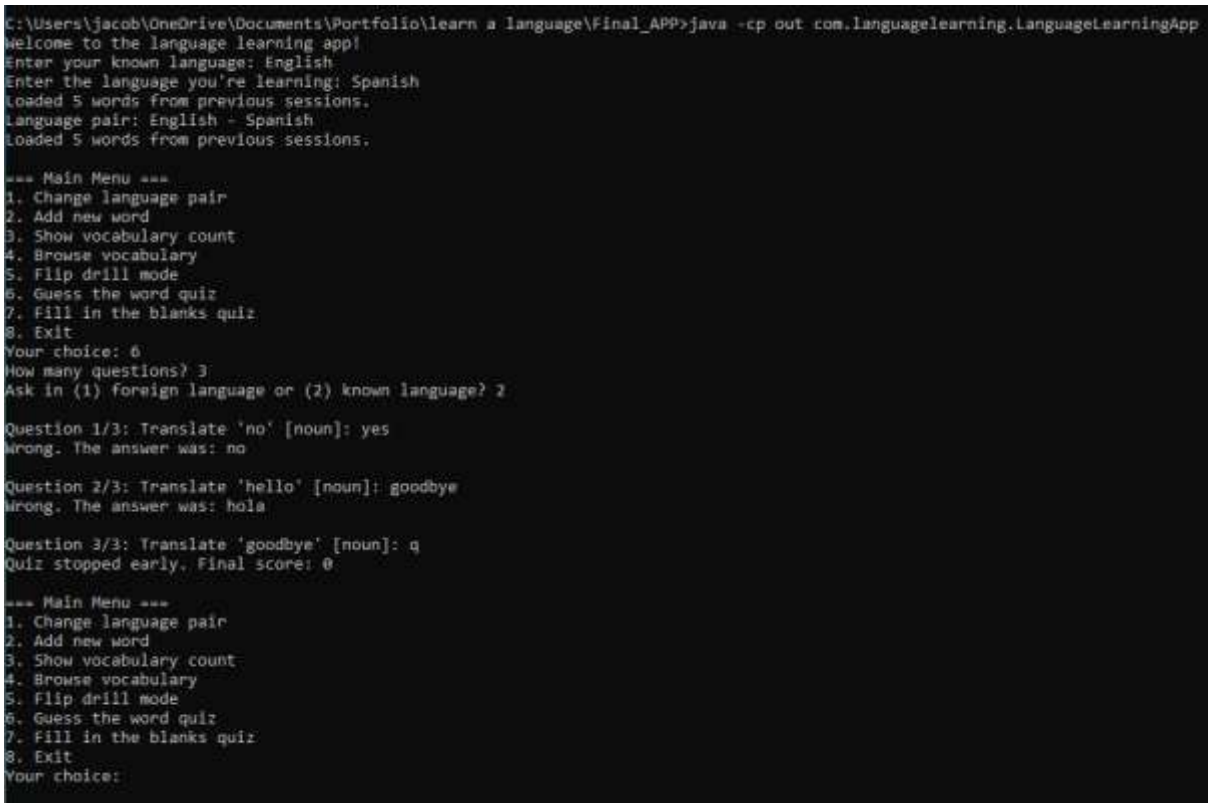
Screenshot of test 12



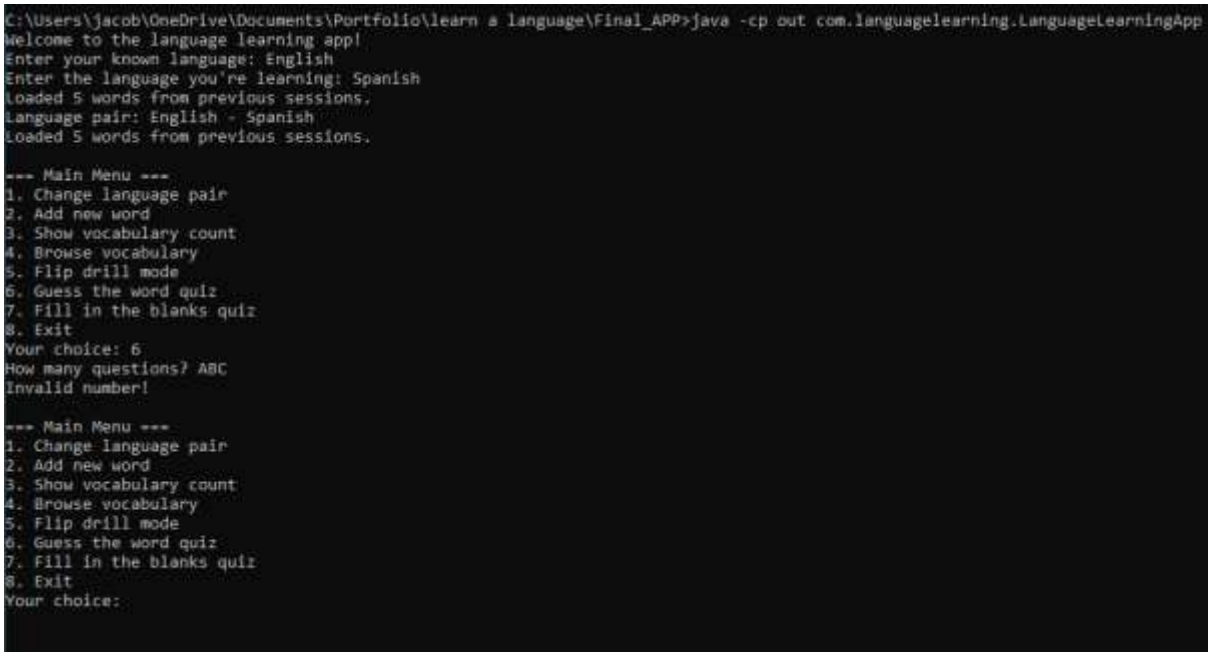
Screenshot of test 13



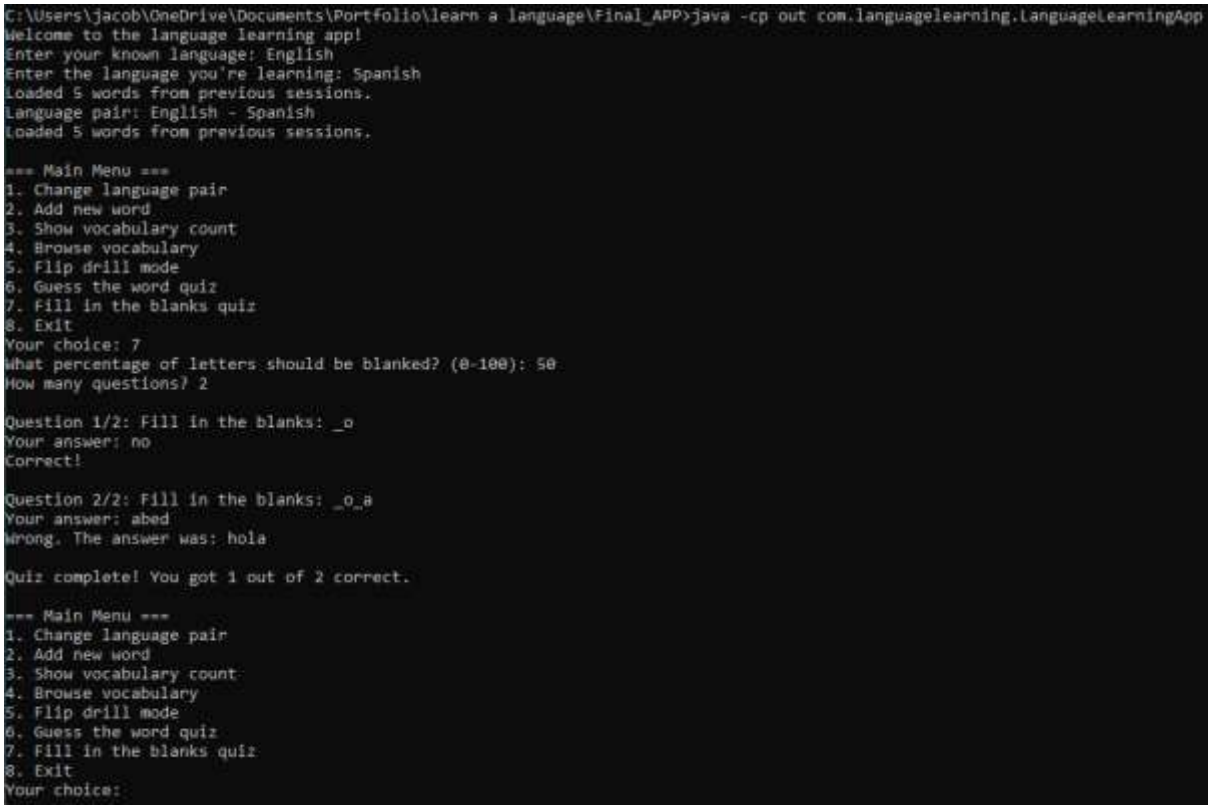
Screenshot of test 14



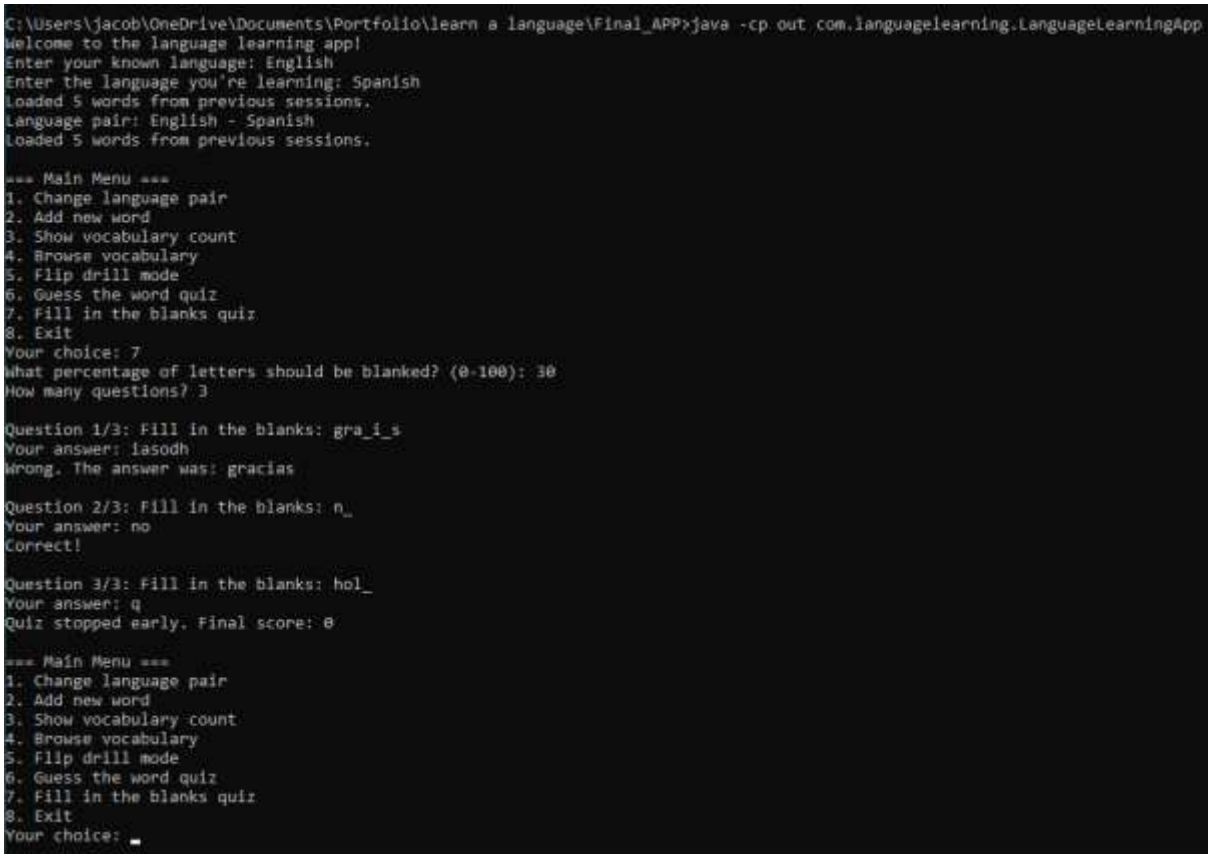
Screenshot of test 15



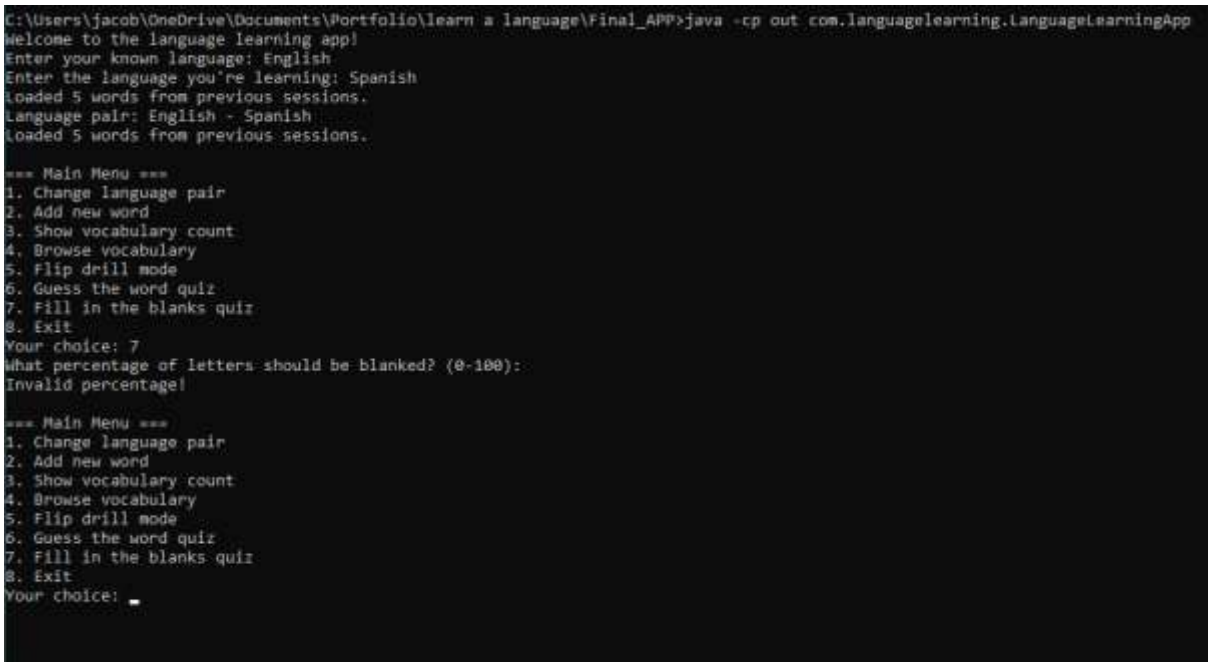
Screenshot of test 16



Screenshot of test 17

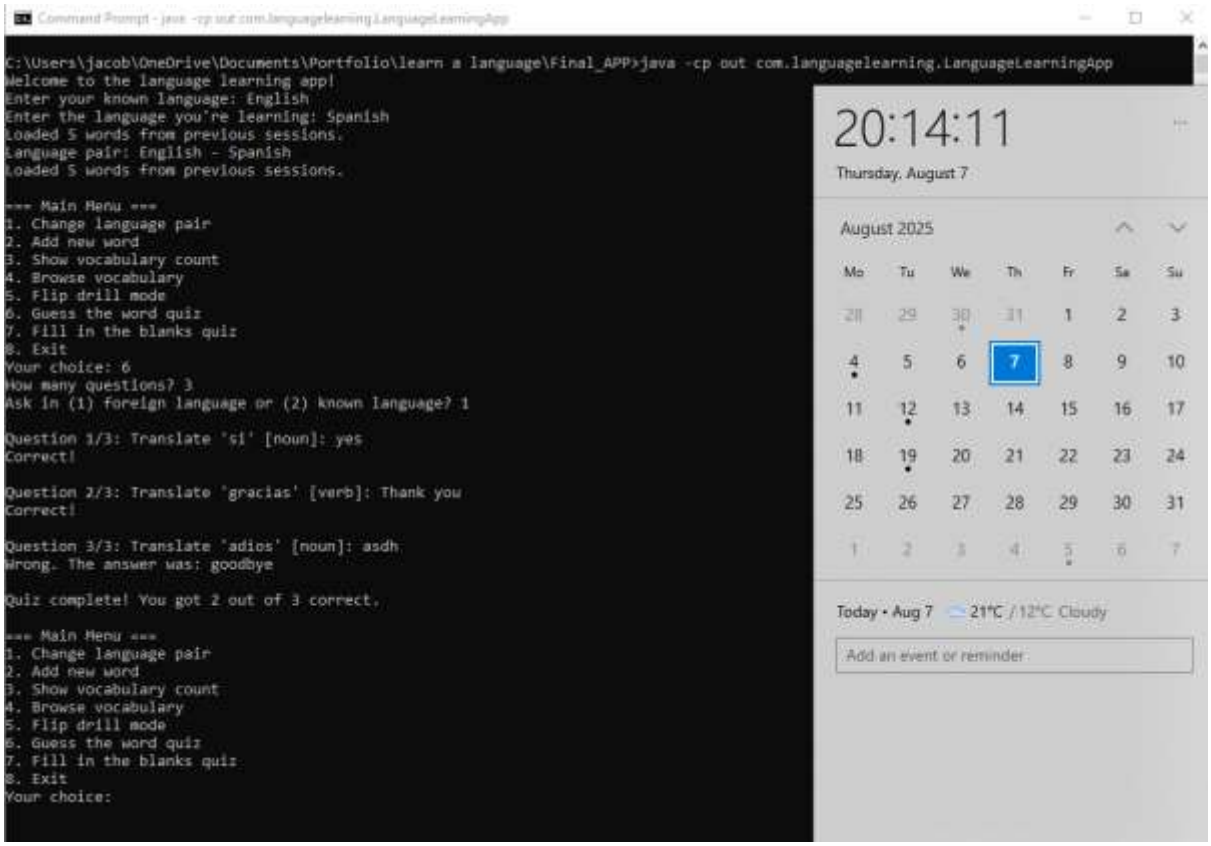


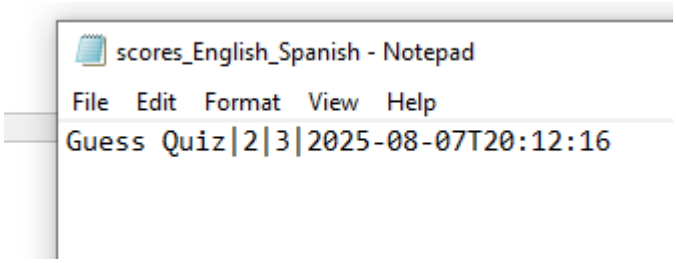
Screenshot of test 18



Screenshot of test 19

Screenshot of test 20





4. Evaluation

4.1 Difficulties encountered

During development, the most challenging aspect was designing a clean, extensible framework for multiple test types (NFR3). Defining the abstract Test base class and ensuring each subclass (FlipDrillTest, GuessTest, FillBlankTest) adhered to the same run() signature required careful planning of shared fields and return values (TestResult). Additionally, handling file I/O (NFR1) robustly—creating directories, reading/writing plain-text without data loss, and formatting timestamps—introduced error-handling complexities that took extra time to debug.

4.2 Remaining work

I believe that I have hit all Functional requirements and non-Functional requirements

4.3 What you learned

This project deepened my understanding of:

- **Inheritance & Polymorphism**: abstracting common behaviour in Test and sharing logic via runTest().
- **Java I/O APIs**: using java.nio.file for directory management, buffered readers/writers for performance.
- **Collections & Streams**: sorting, filtering, and paginating the vocabulary list with minimal code.
- **Robust CLI design**: validating user input, handling invalid entries, and providing clear prompts.

4.4 Self-assigned marks

5. **Functional implementation (20/20)**: All FR1–FR9 fully met and demonstrable via tests.
6. **Persistence & UI (15/15)**: Plain-text vocab/scores, menu-driven CLI, extensible test framework implemented.
7. **Code quality & OOP (14/15)**: Well-structured classes and methods; minor area for improved exception feedback.
8. **Inheritance usage (10/10)**: Abstract `Test` class and three concrete subclasses.
9. **Testing (10/10)**: Comprehensive test table and screenshots for each requirement.
10. **Documentation & Design (14/15)**: Clear UML, pseudocode, and rationale; could include a full sequence diagram for every flow.
11. **Evaluation (5/5)**: Honest reflection and roadmap for future work.

4.5 Creativity and Innovation

In a more advanced design, the app could incorporate a Text-to-Speech engine so that, whenever a learner encounters a new word or takes a quiz, they might hear an accurate pronunciation in the target language—thereby reinforcing listening and speaking skills alongside visual recognition. Likewise, the system could feature an achievement framework that tracks progress and awards badges for milestones such as “First 50 Words Learned,” “Perfect Quiz Score,” or “Seven Consecutive Days of Practice.” Such a gamified layer would not only acknowledge incremental success but could also display a simple progress chart in the CLI, providing ongoing motivation and a clear sense of advancement.