JAVA FLASHCARD LEARNING SYSTEM

AN INTERNSHIP PROJECT REPORT

By

POTNOORU MADHURI DEEKSHIT

322103211057

Under the esteemed guidance of

Mr. G. APPAGI

Assistant Professor

IT Department

Department of INFORMATION TECHNOLOGY

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

[Approved by AICTE NEW DELHI, Affiliated to Andhra University]

[Accredited by National Board of Accreditation (**NBA**) for B.Tech. CSE, ECE & IT – Valid from 2019-22 and 2022-25]

[Accredited by National Assesment and Accreditation Council(**NAAC**)– Valid from 2022-27]

Kommadi , Madhurawada, Visakhapatnam–530048

2024–2025

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

DEPARTMENT OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the internship project report titled **"JAVA FLASHCARD LEARNING SYSTEM"** is a bonafide work of following IV B.Tech. student in the Department of Information Technology, Gayatri Vidya Parishad College of Engineering for Women affiliated to Andhra University, Visakhapatnam. This project was done during the academic year 2024-2025(III rd year Semester-2 summer vacation) and it is evaluating in III-I Semester of the academic year 2024-2025).

POTNOORU MADHURI DEEKSHIT

322103211057

Mr. G. Appaji
Assistant Professor
(Internal Guide)

Dr. M. Bhanu Sridhar
Assistance Professor & HOD
(Head of the department)

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We feel elated to extend our sincere gratitude to **Mr G. Appaji,** Assistant Professor for encouragement all the way during analysis of the project. His annotations, insinuations and criticisms are the key behind the successful completion of the thesis and for providing us all the required facilities.

We express our deep sense of gratitude and thanks to **Dr. M. Bhanu Sridhar**, Professor and Head of the Department of Information Technology for his guidance and valuable and grateful opinions in the project for its development and for providing lab sessions and extra hours to complete the project.

We would like to take this opportunity to express our profound sense of gratitude to Vice Principal, **Dr. G. Sudheer** for allowing us to utilize the college resources thereby facilitating the successful completion of our project. We are also thankful to both teaching and non-teaching faculty of the Department of Information Technology and Engineering for giving valuable suggestions for our project.

We would like to take the opportunity to express our profound sense of gratitude to the revered Principal, **Dr. R. K. Goswami** for all the help and support towards the successful completion of our project.

TABLE OF CONTENTS

	TOPICS	PAGENO.
	Abstract	**5**
1.	**INTRODUCTION**	**6**
	1.1 Problem Statement	
	1.2 Objectives	
2.	**RELATED WORK**	**7**
3.	**SYSTEM ANALYSIS**	**8**
	3.1 Software Requirement Specification	
	3.2 System requirements	
	3.2.1 Software requirements	
	3.2.2 Hardware requirements	
4.	**SYSTEM DESIGN**	**10**
	4.1 Introduction	
	4.2 UML diagrams	
	4.2.1 Use case diagram	
	4.2.2 Class diagram	
	4.2.3 Sequence diagram	
	4.2.4 Activity diagram	
5.	**METHODOLOGY**	**14**
	5.1 Modules, Descriptions	
	5.2 Techniques / Algorithms Used	
	5.3 Packages, Libraries	
6.	**IMPLEMENTATION**	**16**
7.	**CONCLUSION**	**25**
8.	**FUTURE SCOPE**	**26**
9.	**REFERENCES**	**27**
10.	**CERTIFICATE**	**28**

ABSTRACT

The **Java Flashcard Learning System** is a console-based educational application designed to enhance memory retention and learning efficiency through the principles of **spaced repetition** and **active recall**. The project implements the **Leitner Flashcard Method**, a scientifically established learning technique that schedules revision based on user performance. Using this system, learners create flashcards containing concepts and explanations, study them in structured sessions, and receive performance-based feedback that determines how frequently each flashcard appears in subsequent sessions.

The system is developed using **core Java programming concepts**, including object-oriented design, collections, file handling, and modular architecture. Flashcards are stored persistently in a CSV file, ensuring that the learner's progress is maintained across multiple sessions. The study module presents each flashcard exactly once per cycle—eliminating redundancy and reducing cognitive overload. Correct answers promote a flashcard to a higher memory-strength level (Leitner box), while incorrect answers demote it, ensuring targeted reinforcement of weaker topics.

This project demonstrates how simple programming tools can support effective learning strategies and showcases Java's capability to build practical, real-world applications. The Java Flashcard Learning System serves as an accessible, platform-independent tool for students, enabling systematic revision and improved long-term retention.

Keywords:

Flashcard Learning System,

Spaced Repetition,

Leitner Method,

Active Recall,

Educational Technology,

Memory Retention,

Console Application,

Java Collections,

File Handling, CSV Storage,

Learning Enhancement, Study Tool,

Knowledge Management System.

1. INTRODUCTION

Effective learning requires active recall and structured revision, yet traditional study methods often lead to poor long-term retention. Flashcards offer an efficient way to break information into simple units and reinforce memory through repeated review. To enhance this process, the **Java Flashcard Learning System** applies the **Leitner Spaced Repetition Method**, where cards are promoted or demoted based on user performance, ensuring that difficult concepts appear more frequently while well-understood concepts require less frequent review.

Developed using core Java concepts such as object-oriented programming, collections, and file handling, the system provides a simple, menu-driven console interface. Flashcards are stored persistently in a CSV file, allowing users to build and revise a personalized knowledge base over multiple sessions. This project demonstrates how Java can be used to create practical educational tools that improve study efficiency and long-term memory retention.

1.1 Problem Statement

Students often face difficulty retaining information because traditional study methods lack structured revision and effective memory reinforcement. Without a system that supports active recall and spaced repetition, learners struggle to identify weak areas and revise efficiently. There is a need for a simple, offline Java-based tool that allows users to create flashcards, review them systematically, and improve long-term retention.

Input: Flashcard data (front: concept, back: explanation)**.** User responses during study (correct/incorrect/quit)

Output: Updated flashcard levels based on performance**.** Organized list of cards**.** Study statistics**.** Persistent CSV file storing all flashcards and their memory levels

User Interface: A menu-driven console interface that allows users to add cards, study them, view card lists, check statistics, and save progress.

1.2 Objectives

The primary objective of the **Java Flashcard Learning System** is to enhance learning efficiency by implementing a structured, spaced-repetition-based study tool. The system helps users retain information through active recall and repeat exposure to weak concepts.

Specific Objectives

1. To develop a Java-based flashcard system that allows users to create and manage study cards.
2. To implement the **Leitner Spaced Repetition Method** for promoting or demoting cards based on performance.
3. To present each flashcard exactly once per study cycle, ensuring organized and non-repetitive revision.
4. To store flashcards and their progress levels using a CSV file for long-term persistence.
5. To provide a simple, menu-driven console interface for ease of use.
6. To improve memory retention by reinforcing difficult concepts more frequently.

2. RELATED WORK

Flashcard learning has been recognized for decades as an effective method for reinforcing memory through repetition and active recall. Traditional paper-based flashcards offer a simple approach but lack adaptability, automatic scheduling, and progress tracking. As learning demands increased, digital flashcard systems emerged to provide enhanced features.

Anki, one of the most widely used flashcard applications, employs the SuperMemo SM-2 spaced repetition algorithm to automatically determine when each card should be reviewed. While highly effective, Anki's interface and complexity may be overwhelming for beginners, and its installation requirements reduce accessibility.

Quizlet offers a web-based platform with flashcards, tests, and collaborative study modes. It simplifies the learning experience but relies heavily on internet connectivity and provides limited control over spaced repetition compared to specialized tools like Anki.

Mobile-based learning platforms, such as **Brainscape** and **Memrise**, also incorporate spaced repetition and gamification to engage learners. These applications provide intuitive interfaces but often enforce subscription models or require online accounts, which may not be suitable for offline study environments.

Research in cognitive psychology strongly supports spaced repetition and active recall as key components of effective learning. Studies consistently show that revising information at increasing intervals helps transfer knowledge from short-term to long-term memory. The **Leitner Method**, introduced by Sebastian Leitner in 1972, remains one of the simplest and most practical systems for spaced repetition, organizing flashcards into levels based on learner performance.

The **Java Flashcard Learning System** builds upon these concepts by implementing the core principles of the Leitner System in a simple, offline, console-based tool. Unlike complex or commercial learning platforms, this system focuses on accessibility, minimal resource usage, and ease of understanding. By using core Java features such as object-oriented programming and file handling, the system provides a functional and lightweight alternative to larger flashcard applications while demonstrating sound software development practices.

3. SYSTEM ANALYSIS

System analysis identifies the requirements, constraints, and expected behavior of the **Java Flashcard Learning System**. It ensures that the system's design and implementation align with user needs and project objectives. The analysis covers functional behavior, performance expectations, and the necessary resources for running the application effectively.

3.1 Software Requirement Specification (SRS)

The SRS describes what the system should do (functional requirements) and the quality standards it must meet (non-functional requirements).

Functional Requirements

1. The system must allow users to create flashcards containing a concept (front) and explanation (back).
2. It must store all flashcards and load them automatically when the program starts.
3. The study module must implement the **Leitner Spaced Repetition Method**, promoting or demoting cards based on correct or incorrect responses.
4. Each flashcard should appear **once per study cycle**, preventing repetition within the same session.
5. The system must allow users to list all flashcards along with their box levels.
6. It must display statistics showing the number of cards in each box.
7. The system must save all updates to a CSV file to ensure persistent storage.
8. Users must be able to exit safely without losing data.

Non-Functional Requirements

1. **Usability:**
   The system should be simple, intuitive, and easy to use. A menu-driven console interface ensures clarity and minimizes the learning curve for new users.
2. **Performance:**
   Operations such as adding cards, studying, saving, and loading must execute quickly. The system must remain efficient even when handling a large number of flashcards.
3. **Reliability:**
   Data must be stored accurately without corruption. The program should handle invalid inputs safely and maintain consistent behavior across multiple sessions.
4. **Portability:**
   Since the system is developed in Java, it should run on any operating system with Java installed, including Windows, Linux, and macOS.
5. **Maintainability:**
   The system should support easy updates and extension. Modular object-oriented design ensures that changes to one module do not affect others. Features like GUI support or additional study modes can be added in the future with minimal restructuring.

3.2 System Requirements

The system requirements define the software and hardware environment necessary for developing and executing the **Java Flashcard Learning System**. These requirements ensure that the application performs smoothly, remains compatible across platforms, and supports efficient user interaction.

3.2.1 Software Requirements

1. **Operating System:**
   The system can run on any OS that supports Java, such as Windows, Linux, or macOS. No platform-specific features are used, ensuring high portability.
2. **Java Development Kit (JDK):**
   JDK 8 or above is required to compile and run the application. The project uses standard Java libraries, so any modern JDK version will work without additional installations.
3. **Integrated Development Environment (IDE):**
   Tools like IntelliJ IDEA, Eclipse, VS Code, or NetBeans are recommended for easier coding, debugging, and execution. However, the program can also be developed using simple editors like Notepad++ or the command-line interface.
4. **CSV Reader (Optional):**
   Applications such as Microsoft Excel or LibreOffice Calc may be used to view or edit the flashcards.csv file externally. This is optional but useful for verifying data storage.
5. **Java Runtime Environment (JRE):**
   Users only need a JRE installed to run the compiled program, making the application accessible even without a full development setup.

3.2.2 Hardware Requirements

1. **Processor:**
   A basic dual-core processor is sufficient, as the application performs lightweight operations such as text processing and file handling.
2. **Memory (RAM):**
   At least 4 GB of RAM is recommended. This ensures smooth functioning of Java and the development environment, especially when using IDEs.
3. **Storage:**
   The project requires minimal disk space—approximately 100 MB for Java installation, source files, and CSV storage. Flashcard data files occupy only a few kilobytes.
4. **Display:**
   A standard monitor or laptop screen is enough, as the program runs entirely in the console/terminal window. No graphical interface is required.
5. **Input Devices:**
   A keyboard is needed for interacting with the menu-driven interface, adding flashcards, and navigating study sessions.

4. SYSTEM DESIGN

System design focuses on defining the architecture of the system and describing how different components of the system interact to fulfil the project's requirements. It provides a blueprint for how the system will be developed and guides the implementation process.

4.1 Introduction

System design defines the structure and architecture of the **Java Flashcard Learning System** by identifying how different components interact and how the overall functionality is achieved. The system is built using **Object-Oriented Programming (OOP)** principles, allowing modularity, reusability, and clear separation of responsibilities.

The design ensures that the system is easy to understand, maintain, and extend. It follows a **layered architecture** consisting of:

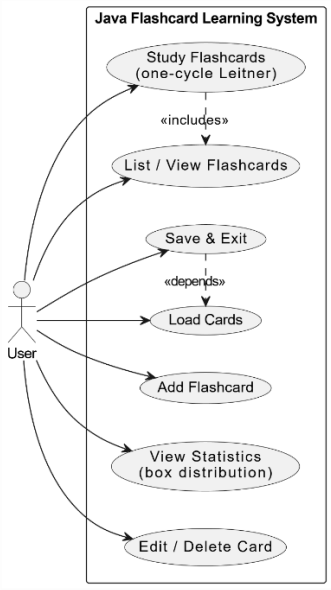
1. **User Interface Layer** – Handles user interaction through a simple console-based menu.
2. **Application Logic Layer** – Manages study sessions, spaced repetition logic (Leitner method), and card-level updates.
3. **Data Management Layer** – Handles reading and writing of flashcard data to a CSV file for persistent storage.

The design revolves around three core classes: **Flashcard** (individual card entity). **Deck** (collection and management of flashcards). **FlashcardApp** (controller for menu and study operations)

This structured approach ensures smooth workflow, efficient data handling, and accurate execution of spaced repetition, making the system both scalable and reliable.

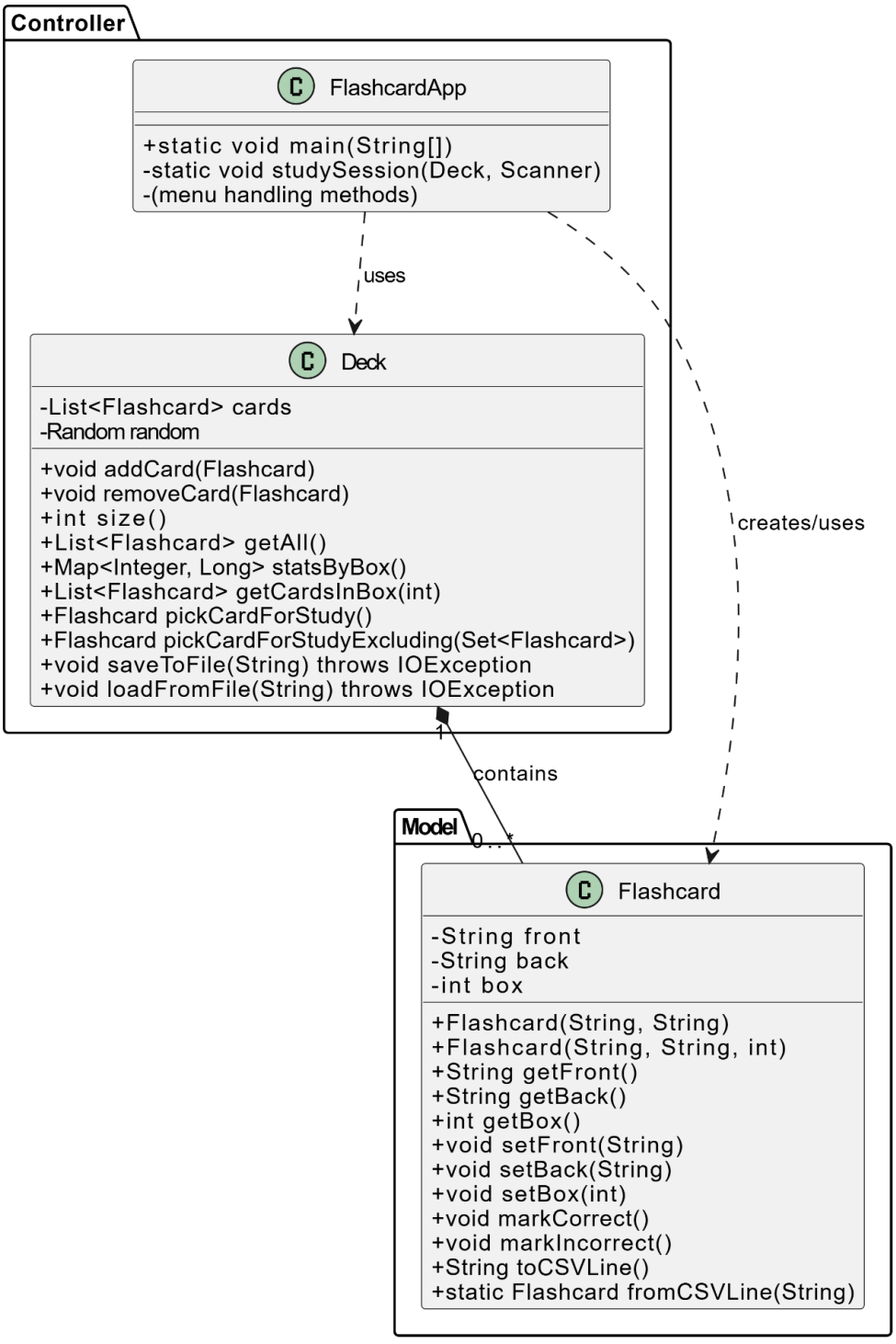
4.2 UML Diagrams: UML (Unified Modelling Language) diagrams are visual representations of the system's components and their interactions. The following UML diagrams describe the architecture of the handwritten digit recognition system:

4.2.1 Use Case Diagram: A Use Case Diagram represents the interactions between the system and external entities (users and systems) involved in the process. It shows the different use cases and how the user interacts with the system.



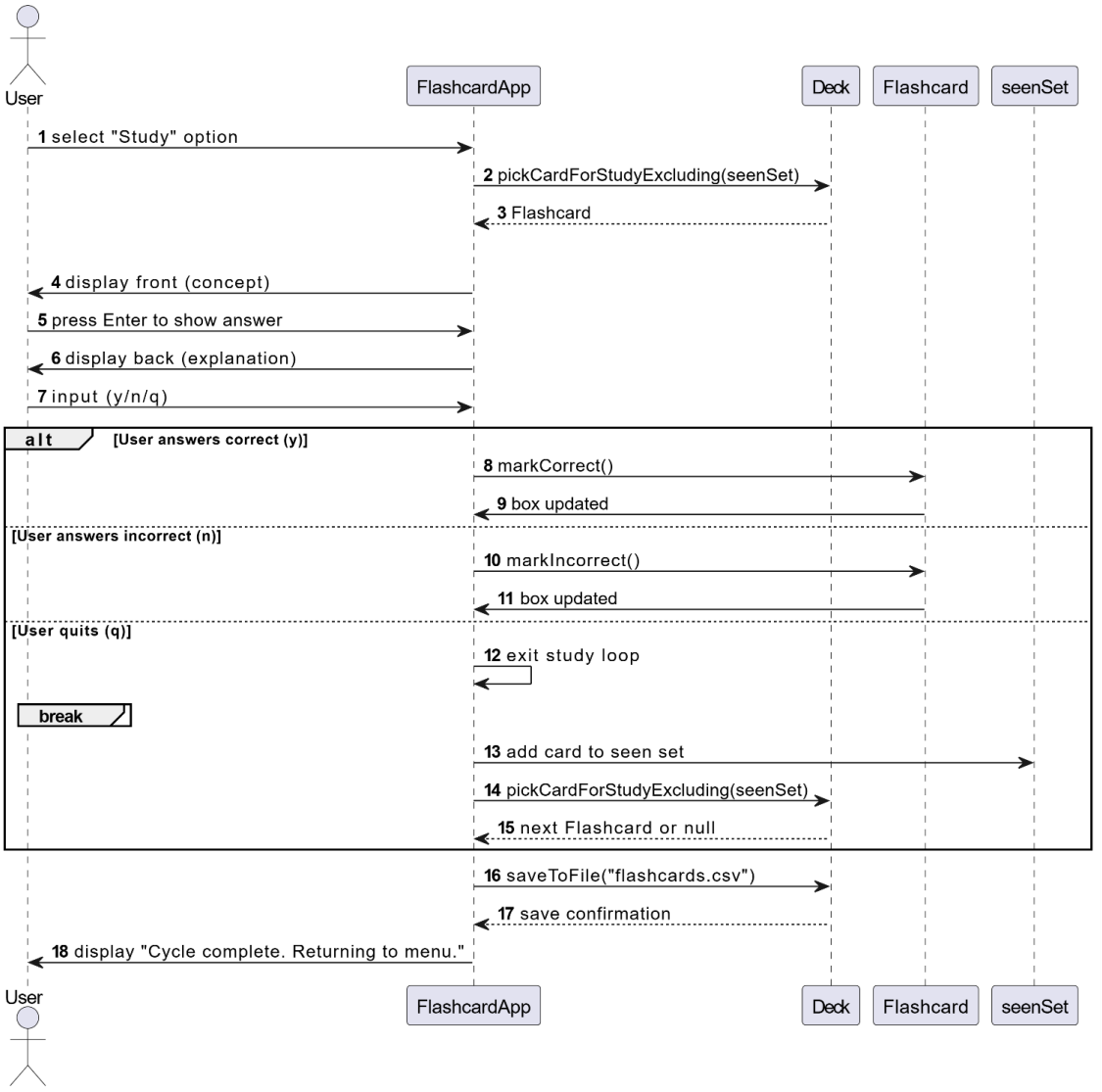
4.2.2 Class Diagram

A Class Diagram shows the structure of the system by detailing the different classes, their attributes, and methods. It describes how the system is divided into different components.



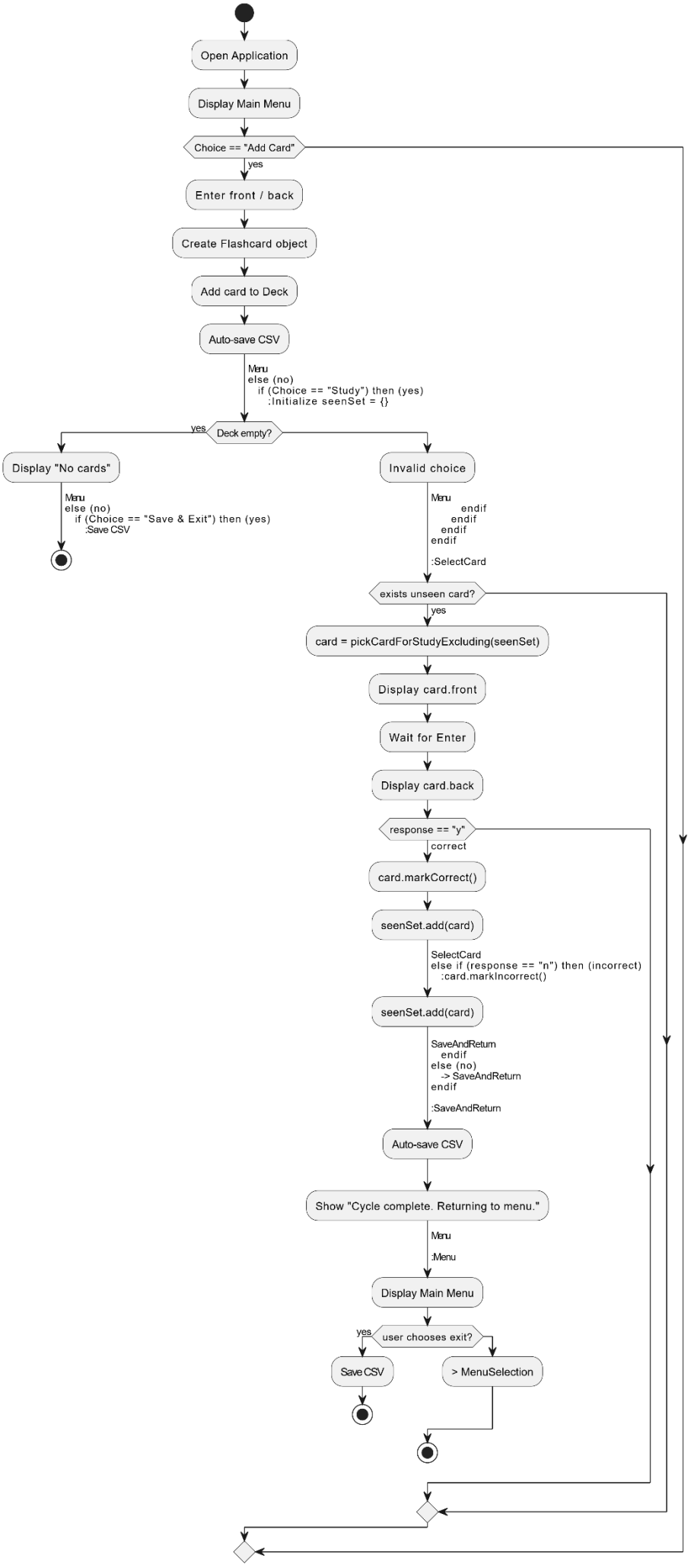
4.2.3 Sequence Diagram

A Sequence Diagram shows how different components of the system interact over time to complete a specific use case or process.



4.2.4 Activity Diagram

An Activity Diagram shows the workflow of the system and how different activities are performed in sequence to achieve the goal.


Open Application
Display Main Menu
Choice == "Add Card"
yes
Enter front / back
Create Flashcard object
Add card to Deck
Auto-save CSV
Menu
else (no)
if (Choice == "Study") then (yes)
:Initialize seenSet = {}
yes
Deck empty?
Display "No cards"
Menu
else (no)
if (Choice == "Save & Exit") then (yes)
:Save CSV
Invalid choice
Menu
endif
endif
endif
endif
:SelectCard
exists unseen card?
yes
card = pickCardForStudyExcluding(seenSet)
Display card.front
Wait for Enter
Display card.back
response == "y"
correct
card.markCorrect()
seenSet.add(card)
SelectCard
else if (response == "n") then (incorrect)
:card.markIncorrect()
seenSet.add(card)
SaveAndReturn
endif
else (no)
-> SaveAndReturn
endif
:SaveAndReturn
Auto-save CSV
Show "Cycle complete. Returning to menu."
Menu
:Menu
Display Main Menu
yes
user chooses exit?
Save CSV
> MenuSelection

5. METHODOLOGY

The methodology section describes the steps and techniques used to build and implement the Java Flashcard Learning system using Java. It includes the different components (modules) of the system and the algorithms used for training and classification.

5.1 Modules and Descriptions

1. Flashcard Module

This module defines the structure of a single flashcard. Each flashcard contains a *front* (concept), a *back* (explanation), and a *box level* indicating its position in the Leitner spaced repetition system. The module also provides methods to update the box level when the user marks an answer correct or incorrect. It includes functions for converting a card to and from CSV format for storage.

2. Deck Module

This module manages the entire collection of flashcards. It stores the list of cards, loads them from the CSV file, and saves changes back to the file. It also contains the logic for selecting a flashcard to study, including weighted random selection and exclusion of cards already shown in the current cycle. The module provides utilities for listing cards, getting statistics, and handling file operations.

3. Study Session Module

This module controls the study process. It manages the one-cycle study mechanism, where each flashcard appears exactly once per session. It displays the card front, reveals the back when the user presses Enter, accepts correctness feedback, and updates the flashcard's box level. It maintains a set of "seen" cards to avoid repetition and ends the cycle when all cards have been shown.

4. File Handling Module

This module handles reading and writing flashcards to the CSV file. It ensures data persistence between sessions. It uses Java's file I/O classes to safely store flashcards in the format front|back|box, ensuring that user progress is saved after each update or study session.

5. User Interface Module

The User Interface module provides a simple, menu-driven console interface. It allows the user to add cards, start a study session, view flashcards, check statistics, and save and exit. It ensures clear prompts, input validation, and smooth navigation.

5.2 Techniques / Algorithms Used

A. Leitner Spaced Repetition Technique

The Leitner method improves long-term memory by promoting correctly answered cards to higher boxes and demoting incorrectly answered cards to lower boxes. Higher-box cards appear less frequently, while lower-box cards appear more often, reinforcing difficult concepts.

B. One-Cycle Study Algorithm

This algorithm ensures that each card appears exactly once per study cycle. The system maintains a set of seen cards, selects only unseen cards during the cycle, and stops when all cards are reviewed. This avoids repetition and provides structured revision.

C. Weighted Random Selection Algorithm

To support spaced repetition more effectively, cards are selected with a weight based on their box level. Cards in lower boxes (weaker knowledge) receive higher weights and thus have a higher chance of appearing during study sessions.

D. CSV Serialization Algorithm

Flashcards are stored in a CSV file so that user data persists across sessions. The algorithm converts each flashcard to a string format (front|back|box) and reconstructs objects from this format when loading.

E. Input Validation and Error Handling

The system validates user choices and handles incorrect input gracefully. It ensures robust execution during file operations and study sessions by using try–catch blocks and fallback actions.

5.3 Packages and Libraries

Java Standard Libraries Used

- **java.util**
  Provides data structures such as ArrayList, List, HashSet, Map, and utilities like Random and Scanner for user interaction.
- **java.io**
  Used for reading and writing data through BufferedReader, BufferedWriter, and IOException.
- **java.nio.file**
  Provides modern file handling mechanisms using Files, Paths, and StandardOpenOption for reliable CSV storage.
- **java.lang**
  Core Java classes automatically imported, including essential types and exceptions.

External Libraries

- None.
  The entire project is implemented using **built-in Java libraries**, ensuring portability and ease of deployment.

6. IMPLEMENTATION

The implementation of the **Java Flashcard Learning System**. The program is implemented using three Java classes:

- Flashcard.java — data model for an individual card
- Deck.java — collection management, selection logic, and CSV persistence
- FlashcardApp.java — console UI and study session controller

All files are plain Java (no external libraries). Save them in the same folder and compile with javac *.java.

Flashcard.java

```
import java.util.Objects;

public class Flashcard {
   private String front;
   private String back;
   private int box; // Leitner box: 1..5
   public Flashcard(String front, String back) {
      this.front = front;
      this.back = back;
      this.box = 1;}
   public Flashcard(String front, String back, int box) {
      this.front = front;
      this.back = back;
      this.box = Math.max(1, Math.min(5, box)); }
   public String getFront() { return front; }
   public String getBack() { return back; }
   public int getBox() { return box; }
   public void markCorrect() {
      if (box < 5) box++;}
   public void markIncorrect() {
      if (box > 1) box--;}
   public void setFront(String front) { this.front = front; }
   public void setBack(String back) { this.back = back; }
```

```
    public void setBox(int box) { this.box = Math.max(1, Math.min(5, box)); }
    @Override
    public String toString() {
        return "Flashcard{" + "front='" + front + '\'' + ", back='" + back + '\'' + ", box=" + box + '}';}
    public String toCSVLine() {    // CSV helper: escape pipe and newline if needed
        String f = front.replace("|", "¦").replace("\n", " ");
        String b = back.replace("|", "¦").replace("\n", " ");
        return f + "|" + b + "|" + box;}
    public static Flashcard fromCSVLine(String line) {
        String[] parts = line.split("\\|", 3);
        if (parts.length < 3) return null;
        String f = parts[0].replace("¦", "|");
        String b = parts[1].replace("¦", "|");
        int box = 1;
        try { box = Integer.parseInt(parts[2]); } catch (NumberFormatException ignored) {}
        return new Flashcard(f, b, box);}
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Flashcard)) return false;
        Flashcard fc = (Flashcard) o;
        return Objects.equals(front, fc.front) && Objects.equals(back, fc.back);}
    @Override
    public int hashCode() {
        return Objects.hash(front, back);}}
```

Deck.java

```
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.stream.Collectors;
import java.nio.charset.StandardCharsets;
import java.nio.file.StandardOpenOption;
```

```
public class Deck {
    private List<Flashcard> cards = new ArrayList<>();
    private Random random = new Random();
    public void addCard(Flashcard card) {
        if (!cards.contains(card)) cards.add(card);}
    public void removeCard(Flashcard card) {
        cards.remove(card);}
    public int size() { return cards.size(); }
    public Map<Integer, Long> statsByBox() {
       return cards.stream().collect(Collectors.groupingBy(Flashcard::getBox, Collectors.counting()));
}
    public List<Flashcard> getCardsInBox(int box) {
        return cards.stream().filter(c -> c.getBox() == box).collect(Collectors.toList());  }
    public Flashcard pickCardForStudy() {// Weighted pick (lower box = higher chance)
        if (cards.isEmpty()) return null;
        List<Flashcard> weighted = new ArrayList<>();
        for (Flashcard c : cards) {
            int weight = 6 - c.getBox();
            for (int i = 0; i < weight; i++) weighted.add(c);}
        return weighted.get(random.nextInt(weighted.size()));} // pick excluding seen set
    public Flashcard pickCardForStudyExcluding(Set<Flashcard> exclude) {
        if (cards.isEmpty()) return null;
        List<Flashcard> weighted = new ArrayList<>();
        for (Flashcard c : cards) {
            if (exclude != null && exclude.contains(c)) continue;
            int weight = 6 - c.getBox();
            for (int i = 0; i < weight; i++) weighted.add(c);}
        if (weighted.isEmpty()) return null;
        return weighted.get(random.nextInt(weighted.size()));}
    // Robust save: create parent folder, use TRUNCATE_EXISTING and flush
    public void saveToFile(String path) throws IOException {
        Path p = Paths.get(path).toAbsolutePath();
```

```
        // ensure parent exists
        Path parent = p.getParent();
        if (parent != null && !Files.exists(parent)) {
            Files.createDirectories(parent);}// Open writer with create + truncate
        try (BufferedWriter writer = Files.newBufferedWriter(p,
                StandardCharsets.UTF_8,
                StandardOpenOption.CREATE,
                StandardOpenOption.TRUNCATE_EXISTING,
                StandardOpenOption.WRITE)) {
            for (Flashcard c : cards) {
                writer.write(c.toCSVLine());
                writer.newLine();}
            writer.flush();}}
    public void loadFromFile(String path) throws IOException {
        Path p = Paths.get(path).toAbsolutePath();
        cards.clear();
        if (!Files.exists(p)) return;
        try (BufferedReader br = Files.newBufferedReader(p, StandardCharsets.UTF_8)) {
            String line;
            while ((line = br.readLine()) != null) {
                Flashcard c = Flashcard.fromCSVLine(line);
                if (c != null) cards.add(c); }}}
    public List<Flashcard> getAll() { return new ArrayList<>(cards); }
}
```

FlashcardApp.java

```
import java.io.IOException;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.HashSet;
import java.nio.file.Paths;
```

```
public class FlashcardApp {
    private static final String DATA_FILE = "flashcards.csv";
    public static void main(String[] args) {
        Deck deck = new Deck(); // try load
        try {
            deck.loadFromFile(DATA_FILE);
                        System.out.println("Loaded " + deck.size() + " cards from " + Paths.get(DATA_FILE).toAbsolutePath());
        } catch (IOException e) {
                System.out.println("No saved deck found. Starting fresh. (Will save to " + Paths.get(DATA_FILE).toAbsolutePath() + ")");}
        Scanner sc = new Scanner(System.in);
        boolean running = true;
        while (running) {
            System.out.println("\n--- Flashcard App ---");
            System.out.println("1) Add card");
            System.out.println("2) Study");
            System.out.println("3) List cards");
            System.out.println("4) Stats");
            System.out.println("5) Save & Exit");
            System.out.print("Choose: ");
            String choice = sc.nextLine().trim();
            switch (choice) {
                case "1":
                    System.out.print("Front: ");
                    String front = sc.nextLine().trim();
                    System.out.print("Back: ");
                    String back = sc.nextLine().trim();
                    deck.addCard(new Flashcard(front, back));
                    System.out.println("Added.");                    // auto-save immediately after add
                    try {
                        deck.saveToFile(DATA_FILE);
```

```
                System.out.println("Auto-saved to " + Paths.get(DATA_FILE).toAbsolutePath());
            } catch (IOException e) {
                System.err.println("Auto-save failed: " + e.getMessage());}
            break;
        case "2":
            studySession(deck, sc);
            break;
        case "3":
            int i = 1;
            for (Flashcard c : deck.getAll()) {
             System.out.println(i++ + ") [" + c.getBox() + "] " + c.getFront() + " -> " + c.getBack());}
            break;
        case "4":
            Map<Integer, Long> stats = deck.statsByBox();
            for (int b = 1; b <= 5; b++) {
                System.out.println("Box " + b + ": " + stats.getOrDefault(b, 0L)); }
            break;
        case "5":
            try {
                deck.saveToFile(DATA_FILE);
                System.out.println("Saved to " + Paths.get(DATA_FILE).toAbsolutePath());
            } catch (IOException e) {
                System.err.println("Failed to save: " + e.getMessage());}
            running = false;
            break;
        default:
            System.out.println("Unknown choice."); }}
    sc.close();}    // studySession: shows each card once and then returns to menu
private static void studySession(Deck deck, Scanner sc) {
    if (deck.size() == 0) {
        System.out.println("No cards. Add some first.");
        return; }
    System.out.println("Study mode — press Enter to show answer, then y/n if you got it right.");
```

```
System.out.println("You can also press 'q' after seeing answer to quit early.");
        Set<Flashcard> seenThisCycle = new HashSet<>();
        boolean keepStudying = true;
        while (keepStudying) {
            Flashcard card = deck.pickCardForStudyExcluding(seenThisCycle);
            if (card == null) {
                System.out.println("\nCycle complete. Returning to main menu.");
                break; }
            seenThisCycle.add(card);
            System.out.println("\nFront: " + card.getFront());
            System.out.print("(press Enter to see answer) ");
            sc.nextLine();
            System.out.println("Back: " + card.getBack());
            System.out.print("Did you answer correctly? (y/n or q to quit study): ");
            String ans = sc.nextLine().trim().toLowerCase();
            if (ans.equals("y")) {
                card.markCorrect();
                System.out.println("Great! Moved to box " + card.getBox());
            } else if (ans.equals("n")) {
                card.markIncorrect();
                System.out.println("No worries. Moved to box " + card.getBox());
            } else if (ans.equals("q")) {
                System.out.println("Quitting study early. Returning to main menu.");
                break;
            } else {
                System.out.println("Unknown input, returning to menu.");
                break; }}
        // auto-save progress after study
        try {
            deck.saveToFile(DATA_FILE);
            System.out.println("Progress auto-saved to " + Paths.get(DATA_FILE).toAbsolutePath());
        } catch (IOException e) {
```

```
            System.err.println("Auto-save after study failed: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
D:\Madhuri\3-2 project>java FlashcardApp
Loaded 0 cards from D:\Madhuri\3-2 project\flashcards.csv

--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 1
Front: A program in execution, containing code, data, and its PCB.
Back: process
Added.
Auto-saved to D:\Madhuri\3-2 project\flashcards.csv
```

```
--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 1
Front: The smallest unit of scheduling inside a process.
Back: thread
Added.
Auto-saved to D:\Madhuri\3-2 project\flashcards.csv
```

```
--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 1
Front: Removing redundancy and improving data consistency.
Back: normalization
Added.
Auto-saved to D:\Madhuri\3-2 project\flashcards.csv

--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 1
Front: A blueprint for creating objects.
Back: class
Added.
Auto-saved to D:\Madhuri\3-2 project\flashcards.csv
```

```
--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 3
1) [1] A program in execution, containing code, data, and its PCB. -> process
2) [1] The smallest unit of scheduling inside a process. -> thread
3) [1] Removing redundancy and improving data consistency. -> normalization
4) [1] A blueprint for creating objects. -> class
5) [1] Restricting direct access to object data using private fields. -> encapsulation
6) [1] Converting readable data into protected, unreadable form. -> encryption

--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 4
Box 1: 6
Box 2: 0
Box 3: 0
Box 4: 0
Box 5: 0
```

```
--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 2
Study mode ? press Enter to show answer, then y/n if you got it right.
You can also press 'q' after seeing answer to quit early.

Front: The smallest unit of scheduling inside a process.
(press Enter to see answer) thread
Back: thread
Did you answer correctly? (y/n or q to quit study): y
Great! Moved to box 2

Front: Restricting direct access to object data using private fields.
(press Enter to see answer)
Back: encapsulation
Did you answer correctly? (y/n or q to quit study): n
No worries. Moved to box 1

Front: Converting readable data into protected, unreadable form.
(press Enter to see answer)
Back: encryption
Did you answer correctly? (y/n or q to quit study): n
No worries. Moved to box 1
```

```
Cycle complete. Returning to main menu.
Progress auto-saved to D:\Madhuri\3-2 project\flashcards.csv

--- Flashcard App ---
1) Add card
2) Study
3) List cards
4) Stats
5) Save & Exit
Choose: 5
Saved to D:\Madhuri\3-2 project\flashcards.csv
```

A9

	A	B	C	D	E	F
1	A program in execution	containing code	data	and its PCB.|process|2		
2	The smallest unit of scheduling inside a process.|thread|2					
3	Removing redundancy and improving data consistency.|normalization|2					
4	A blueprint for creating objects.|class|2					
5	Restricting direct access to object data using private fields.|encapsulation|1					
6	Converting readable data into protected	unreadable form.|encryption|1				
7						

7. CONCLUSION

The **Java Flashcard Learning System** effectively integrates spaced repetition principles with core Java concepts to create a simple and efficient learning tool. By using the Leitner method and a one-cycle study approach, the system helps users revise content in a structured and memory-enhancing manner.

1. Model Development

The system was built using modular object-oriented design, separating responsibilities into Flashcard, Deck, Study Session, File Handling, and User Interface modules. This structure ensures clarity, maintainability, and easy extension.

2. Performance

The application performs reliably with fast loading, saving, and card selection. Its lightweight data structures and CSV-based storage allow smooth operation even with large sets of flashcards.

3. User Interface

A simple, menu-driven console interface ensures ease of use. Clear prompts make it accessible to all users, requiring no prior technical experience.

4. Limitations and Improvements

Current limitations include the absence of card editing, deleting, multiple decks, or a graphical interface. Future enhancements may introduce:

- GUI support (JavaFX/Swing)
- Multi-deck management
- Advanced spaced repetition scheduling
- Progress visualization and analytics

8. FUTURE SCOPE

The Java Flashcard Learning System lays a strong foundation for interactive and adaptive learning. Although the current version provides the essential features required for effective revision, several enhancements can significantly improve usability, performance, and learner engagement. The following points highlight potential future developments:

1. Graphical User Interface (GUI)

A GUI built using JavaFX or Swing can replace the console-based interface to make the system more user-friendly. Visual cards, buttons, menus, animations, and color-coded Leitner boxes would greatly improve navigation and user engagement. A GUI also allows additional components like search bars, filters, and deck summaries to be displayed more effectively.

2. Multiple Deck Management

In future versions, users can be allowed to create multiple decks categorized by subject, chapter, or difficulty level. This helps learners organize their study material better and focus on specific topics. Options such as merging decks, importing deck files, and assigning colors or tags to decks can also be added.

3. Card Editing and Deletion

Providing the ability to edit or delete existing flashcards makes the system more flexible. Users can correct mistakes, update outdated content, and remove irrelevant cards. This ensures that the study material remains accurate and relevant over time.

4. Advanced Spaced Repetition Algorithms

The current Leitner-based spacing is effective but basic. Future upgrades can incorporate advanced algorithms like SM-2, which dynamically calculates review intervals based on memory strength. Time-based scheduling, difficulty ratings, and predictive review dates can significantly enhance learning efficiency and mimic professional tools like Anki and SuperMemo.

5. Progress Analytics and Visualization

Adding graphical analytics can help learners understand their progress in a more meaningful way. Features such as weekly improvement graphs, accuracy percentages, box distribution charts, and performance trends can motivate users and guide focused revision. Advanced analytics may even give personalized study recommendations.

6. Mobile and Cloud Synchronization

Developing Android/iOS versions would allow learners to revise on the go. Cloud synchronization ensures that all devices share the same data, enabling seamless transitions between desktop and mobile study sessions. Online backups also protect the user's data from accidental loss.

7. Audio/Image Support

Extending flashcards to support images, audio clips, and symbols makes the system useful for subjects that rely heavily on visuals or pronunciation—such as language learning, biology diagrams, or engineering symbols. This feature would significantly expand the range of topics the system can effectively teach.

9. REFERENCES

[1] Baeldung, “Guide to Java Collections Framework,” Available at: https://www.baeldung.com/java-collections

[2] GeeksforGeeks, “File Handling in Java – Overview and Examples,” Available at: https://www.geeksforgeeks.org/file-handling-in-java/

[3] Nicky Case, “The Leitner System and Spaced Repetition Explained,” Available at: https://ncase.me/remember/

[4] SuperMemo Research, “Spaced Repetition: The Learning Technique Explained,” Available at: https://www.supermemo.com/en/archives1990-2015/english/rep

[5] Stack Overflow, “How to Read and Write CSV Files in Java?” Available at: https://stackoverflow.com/questions/4961405/how-to-read-a-csv-file-in-java

[6] GitHub Projects, “Open-Source Java Flashcard Applications,” Available at: https://github.com/topics/flashcards-java

[7] Cult of Pedagogy, “Active Recall and Spaced Practice: Effective Learning Strategies,” Available at: https://www.cultofpedagogy.com/learning-strategies/

10.CERTIFICATE