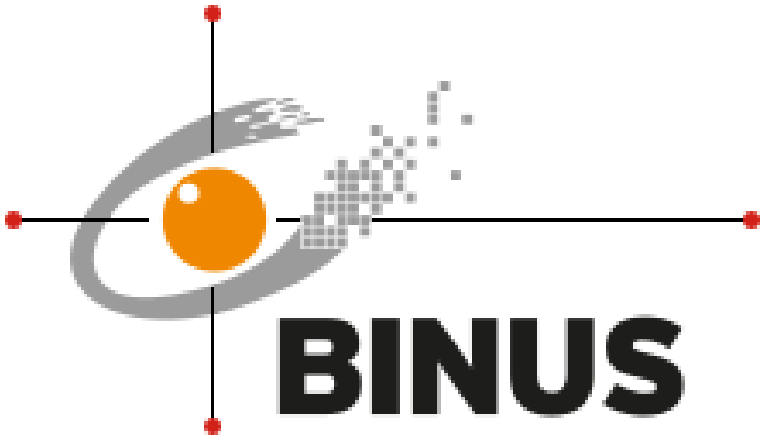
Final Project Report



BINUS
UNIVERSITY
INTERNATIONAL

Object Oriented Programming

COMP6699001

Lecturer

Jude Joseph Lamug Martinez, MCS

Report by

Joelliane Anggra (2802466322)

L2AC

Type of Assignment : Final Project Report

Submission Pattern:

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offences which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Joelliane Anggra

Table of Contents

I. Project Specification

a. Background

For this Object-Oriented Programming course final project, it is expected that we apply the main concepts of OOP as well as expand upon the topics covered in class whilst solving a particular problem. This led me to brainstorm the problems I have encountered as an individual that may allow me to encapsulate these ideas effectively. There are two widely-known flashcard applications that I know of, both at opposite sides of the spectrum in terms of customization and user-friendliness, that is Anki and Quizlet. Whilst Quizlet makes the process of creating and learning flashcards simple, it lacks the flexibility of Anki's learning algorithm in terms of being able to provide more varied feedback when studying the flashcards. Hence, the problem I intend to solve is the lack of a user-friendly flashcard application with an effective spaced repetition algorithm.

b. Solution Description

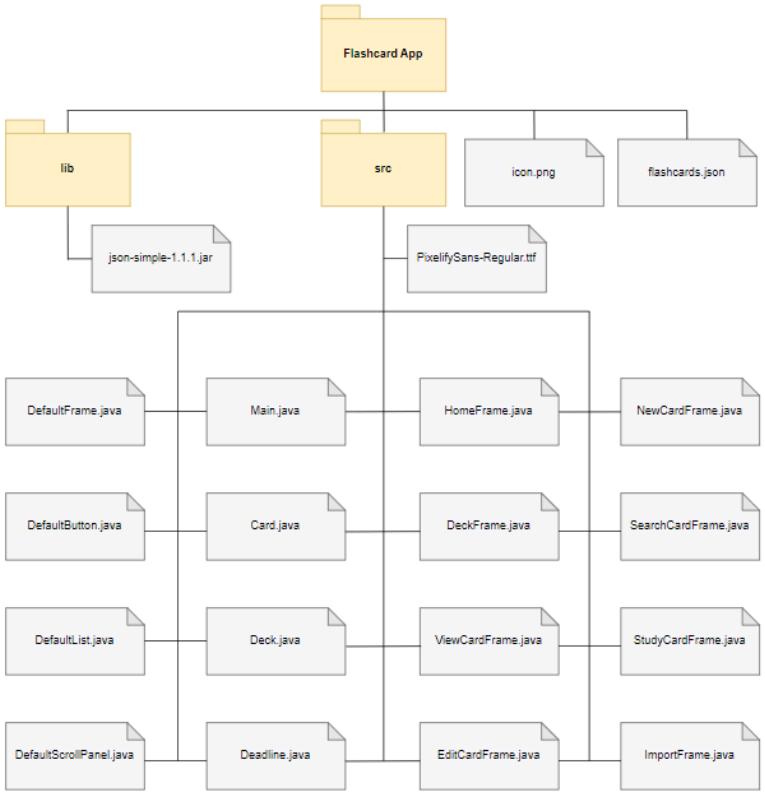
To tackle this problem, I decided to create a flashcard application using JAVA. It will have the main features of the average flashcard program, a way of adding, removing and editing flashcards, but also a study mode that utilizes a more complex system that will maximize long-term memory. There will also be the added feature of a deadline list where the user can add their personal schedule for upcoming dates they want to keep track of and also an import function, specifically for importing from Quizlet due to its extensive existing catalog of flashcards, to make for convenient data inputting.

The user will first be directed to the Home window where there will be a menu to do the basic functionalities, including a search feature to select a particular card. The list of on-going deadlines will be displayed as well as a list of decks which can redirect the user to the Deck window. Additionally, a Study All function is also available. The Deck window includes the options to add a new card, view cards and study cards in the particular deck whilst displaying the cards in the deck.

c. Libraries Used

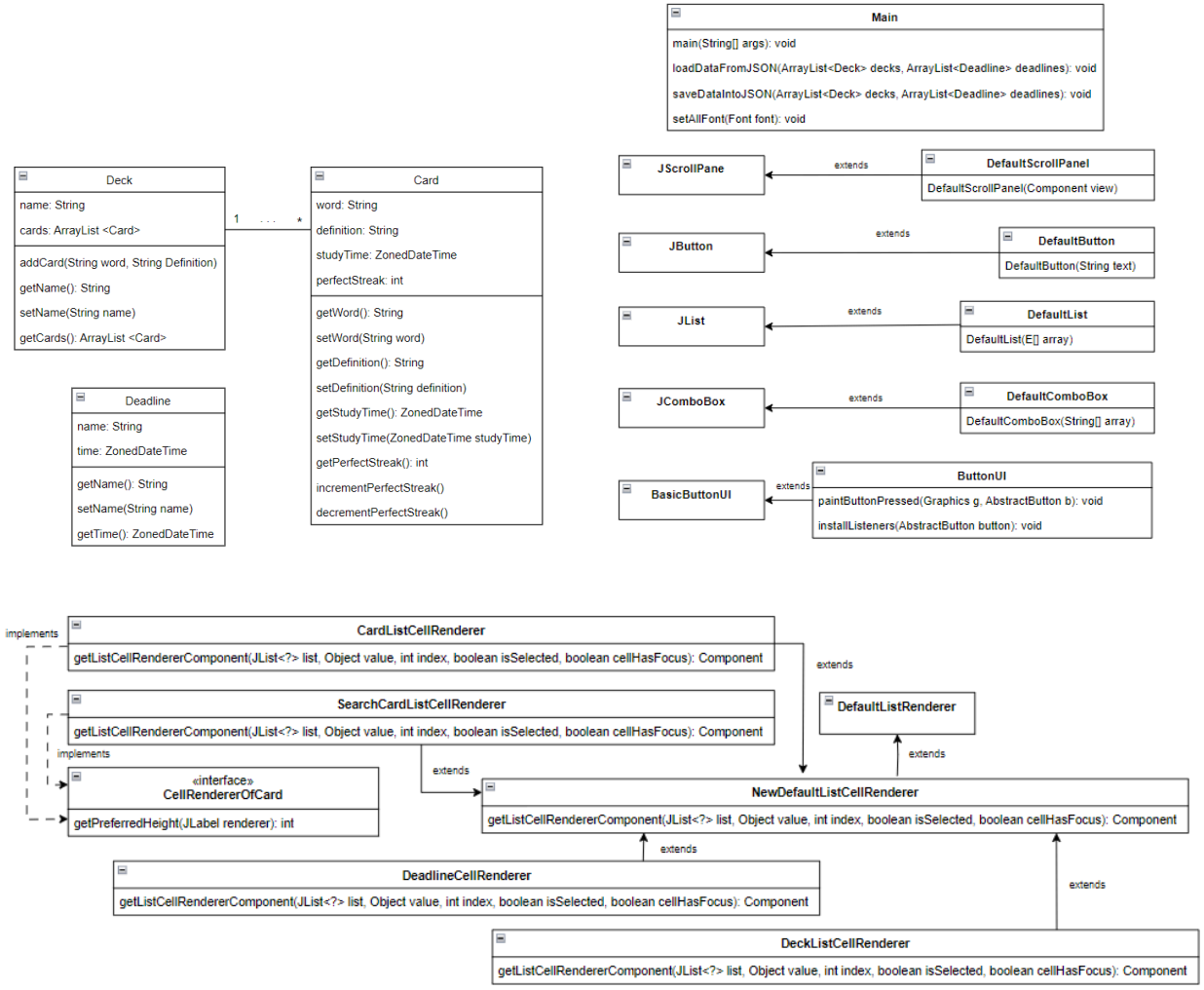
1. Java Standard Library (Java SE):
    - javax.swing: For creating the GUI components.
    - java.awt: For layouts, colors, fonts, and other GUI-related functionalities.
    - java.util: For data structures like ArrayList and other utility classes.
2. Json-simple ToolKit:
    - org.json.simple.JSONArray: For creating and manipulating JSON arrays.
    - org.json.simple.JSONObject: For creating and manipulating JSON objects.
    - org.json.simple.parser.JSONParser: For parsing JSON text into Java objects.
    - org.json.simple.parser.ParseException: For handling exceptions that occur during parsing.

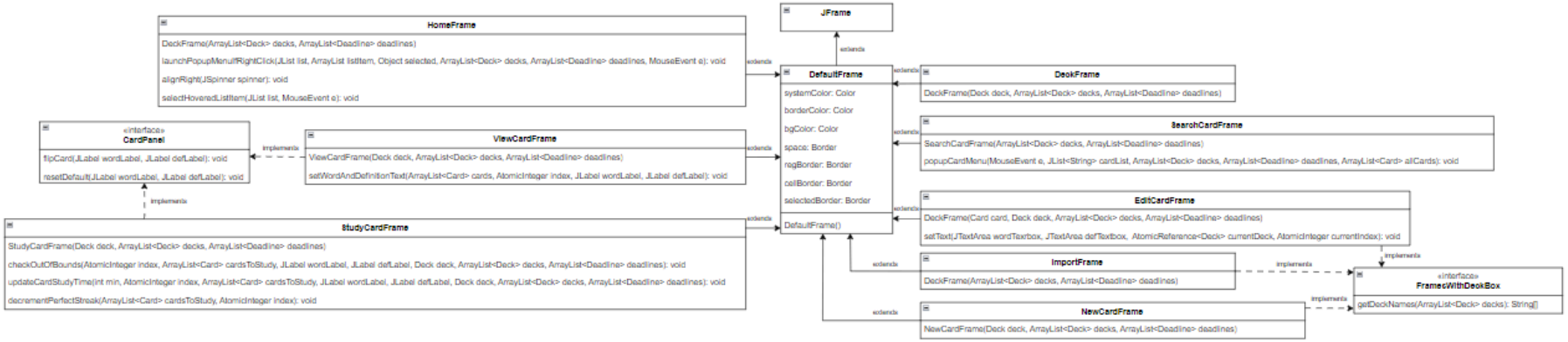
d. File Structure



II. Solution Design

a. Class Diagram

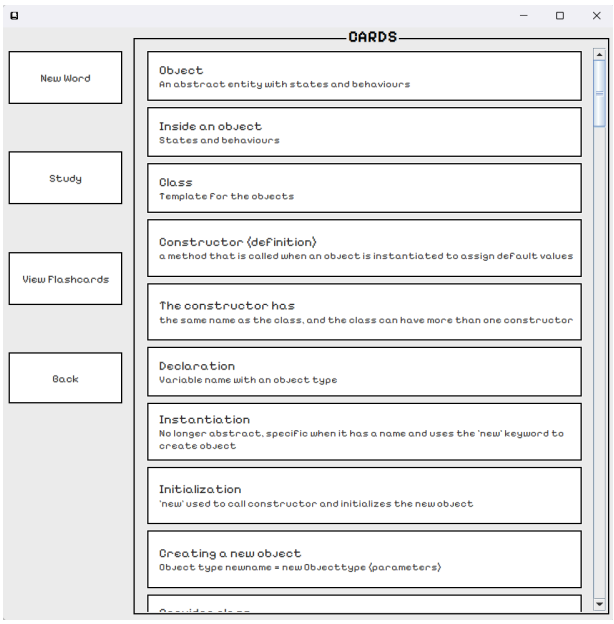



JFrame
HomeFrame
DeckFrame(ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
launchPopupMenuIfRightClick(JList list, ArrayList listItem, Object selected, ArrayList<Deck> decks, ArrayList<Deadline> deadlines, MouseEvent e): void
alignRight(JSpinner spinner): void
selectHoveredListItem(JList list, MouseEvent e): void
extends
DefaultFrame
systemColor: Color
borderColor: Color
bgColor: Color
space: Border
regBorder: Border
selBorder: Border
selectedBorder: Border
DefaultFrame()
DeckFrame
DeckFrame(Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
SearchCardFrame
SearchCardFrame(ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
popupCardMenu(MouseEvent e, JList<String> cardList, ArrayList<Deck> decks, ArrayList<Deadline> deadlines, ArrayList<Card> allCards): void
EditCardFrame
DeckFrame(Card card, Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
setText(JTextArea wordTextbox, JTextArea defTextbox, AtomicReference<Deck> currentDeck, AtomicInteger currentIndex): void
ImportFrame
DeckFrame(ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
NewCardFrame
NewCardFrame(Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
implements
«interface»
FramesWithDeckBox
getDeckNames(ArrayList<Deck> decks): String[]
«interface»
CardPanel
flipCard(JLabel wordLabel, JLabel defLabel): void
resetDefault(JLabel wordLabel, JLabel defLabel): void
ViewCardFrame
ViewCardFrame(Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
setWordAndDefinitionText(ArrayList<Card> cards, AtomicInteger index, JLabel wordLabel, JLabel defLabel): void
StudyCardFrame
StudyCardFrame(Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines)
checkOutOfBounds(AtomicInteger index, ArrayList<Card> cardsToStudy, JLabel wordLabel, JLabel defLabel, Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines): void
updateCardStudyTime(int min, AtomicInteger index, ArrayList<Card> cardsToStudy, JLabel wordLabel, JLabel defLabel, Deck deck, ArrayList<Deck> decks, ArrayList<Deadline> deadlines): void
decrementPerfectStreak(ArrayList<Card> cardsToStudy, AtomicInteger index): void

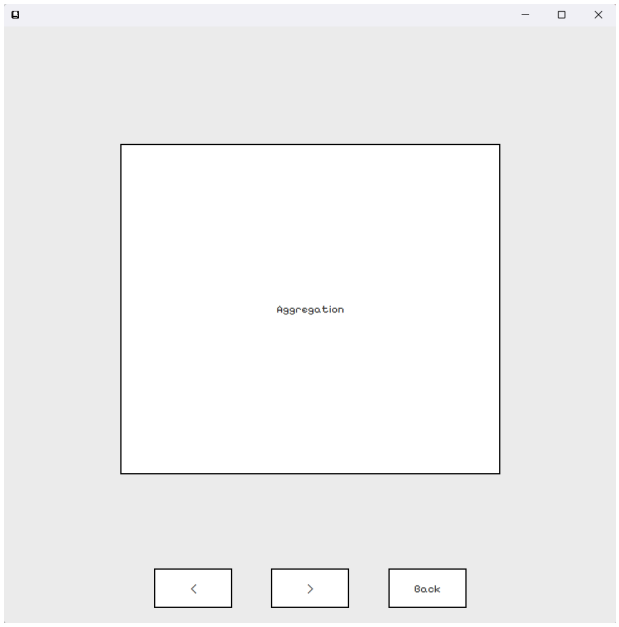
b. Program Design

Home Window

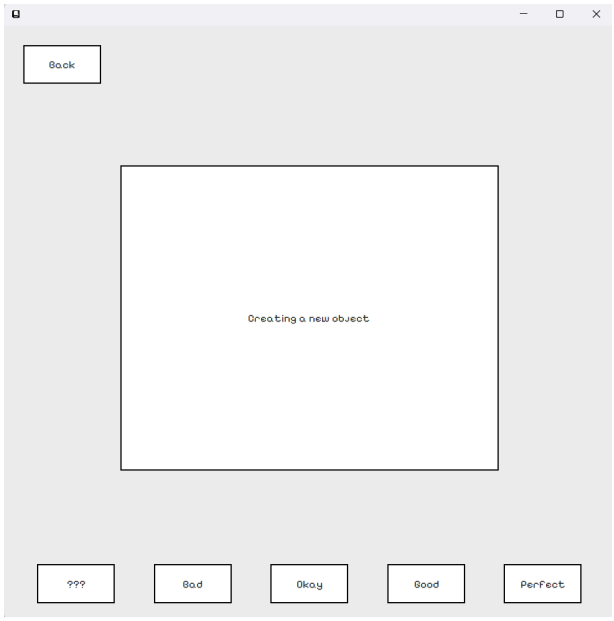
Deck WIndow



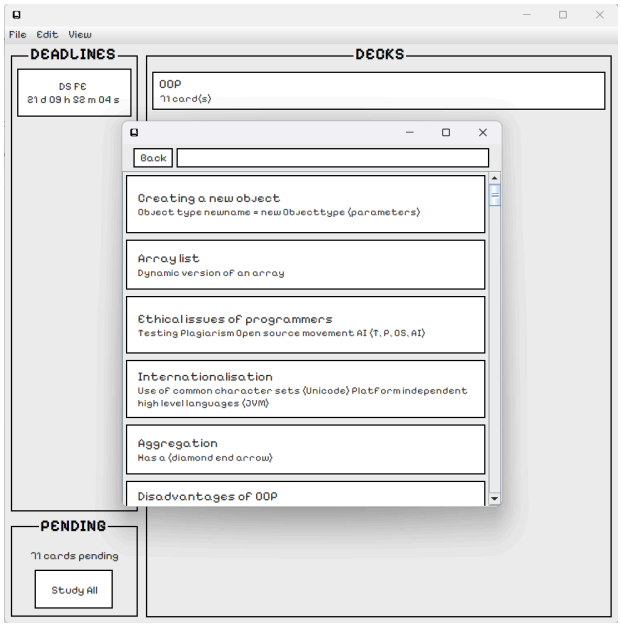
View Card Window



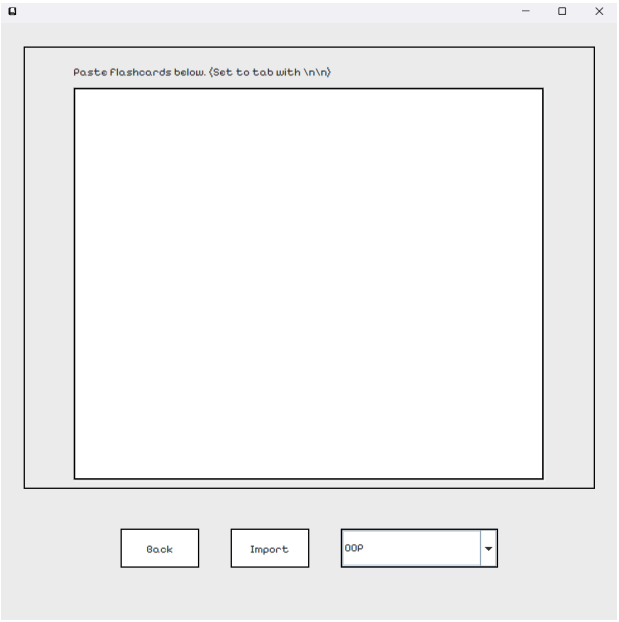
Study Card Window



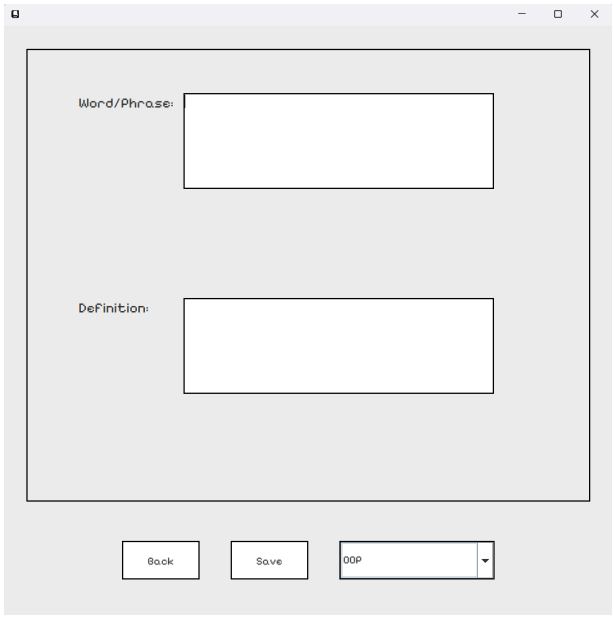
Search Card Window



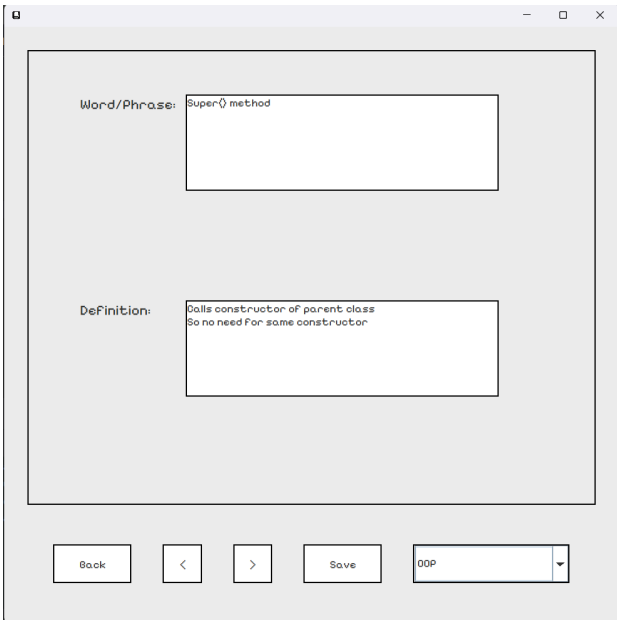
Import Card Window



Add New Card Window



Edit Card Window



c. Important Algorithms

loadDataFromJSON(ArrayList<Deck> decks, ArrayList<Deadline> deadlines)

```
//Loads data from JSON file into program
public static void loadDataFromJSON(ArrayList<Deck> decks, ArrayList<Deadline>
deadlines) {
   JSONParser parser = new JSONParser();
   try (FileReader reader = new FileReader("flashcards.json")) { //Reading
JSON file
       JSONObject jsonObject = (JSONObject) parser.parse(reader);
       JSONArray decksArray = (JSONArray) jsonObject.get("decks"); //Puts all
deck JSONObjects into JSONArray
       JSONArray deadlinesArray = (JSONArray) jsonObject.get("deadlines");
//Puts all deadline JSONObjects into JSONArray
       if (decksArray ≠ null) { //Checks if a deck exists in JSON file
           for (Object deckObj : decksArray) {
               JSONObject deckJson = (JSONObject) deckObj;
               String deckName = (String) deckJson.get("name");
               ArrayList<Card> cards = new ArrayList<>();
               JSONArray cardsArray = (JSONArray) deckJson.get("cards");
               for (Object cardObj : cardsArray) {
                   JSONObject cardJson = (JSONObject) cardObj;
                   String word = (String) cardJson.get("word");
                   String definition = (String) cardJson.get("definition");
                   ZonedDateTime studyTime = ZonedDateTime.parse((String)
cardJson.get("studyTime"), DateTimeFormatter.ISO_ZONED_DATE_TIME);
                   int perfectStreak = Math.toIntExact((Long)
cardJson.get("perfectStreak"));
                   cards.add(new Card(word, definition, studyTime,
perfectStreak)); //Create new Card for every instance of JSONObject card in
deck
```

```
                }
                decks.add(new Deck(deckName, cards)); //Create new Deck for
every instance of JSONObject deck
            }
        }
        if (deadlinesArray ≠ null) { //Checks if a deadline exists in JSON
file
            for (Object deadlinesObj : deadlinesArray) {
                JSONObject deadlinesJson = (JSONObject) deadlinesObj;
                String deadlineName = (String) deadlinesJson.get("name");
                ZonedDateTime deadlineTime = ZonedDateTime.parse((String)
deadlinesJson.get("time"), DateTimeFormatter.ISO_ZONED_DATE_TIME);
                deadlines.add(new Deadline(deadlineName, deadlineTime));
            }
        }
    } catch (IOException | ParseException e) {
        e.printStackTrace(); //If file cannot be read
    }
}
```

saveDataIntoJSON(ArrayList<Deck> decks, ArrayList<Deadline> deadlines)

```
//Rewrites data into JSON file
public static void saveDataIntoJSON(ArrayList<Deck> decks, ArrayList<Deadline>
deadlines) {
    Runtime.getRuntime().addShutdownHook(new Thread(() → {
        JSONObject jsonObject = new JSONObject();
        JSONArray decksArray = new JSONArray();
        JSONArray deadlinesArray = new JSONArray();
        for (Deck deck : decks) {
            JSONObject deckObject = new JSONObject();
```

```
            deckObject.put("name", deck.getName());
            JSONArray cardsArray = new JSONArray();
            for (Card card : deck.getCards()) {
                //Conversion of arraylist to JSONArray
                JSONObject cardObject = new JSONObject();
                cardObject.put("word", card.getWord());
                cardObject.put("definition", card.getDefinition());
                cardObject.put("studyTime",
card.getStudyTime().format(DateTimeFormatter.ISO_ZONED_DATE_TIME));
                cardObject.put("perfectStreak", card.getPerfectStreak());
                cardsArray.add(cardObject);
            }
            deckObject.put("cards", cardsArray);
            decksArray.add(deckObject);
        }
        for (Deadline deadline : deadlines) {
            //Conversion of arraylist to JSONArray
            JSONObject deadlineObject = new JSONObject();
            deadlineObject.put("name", deadline.getName());
            deadlineObject.put("time",
deadline.getTime().format(DateTimeFormatter.ISO_ZONED_DATE_TIME));
            deadlinesArray.add(deadlineObject);
        }
        jsonObject.put("decks", decksArray);
        jsonObject.put("deadlines", deadlinesArray);
        try (FileWriter writer = new FileWriter("flashcards.json")) {
            writer.write(jsonObject.toJSONString());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }));
}
```

I made use of the JSON-simple ToolKit to save and load data into a JSON file. How it works is that it converts the data to be saved into JSON compatible objects so that it can be stored into the JSON file and the opposite is true for loading the data. In this case, I converted the decks arraylist and deadlines arraylist to JSON arrays, while converting each instance variable into JSON objects.

getDeckNames(ArrayList <Deck> decks)

```
    default String[] getDeckNames(ArrayList<Deck> decks) {
        String[] deckNames = new String[decks.size()];
        for (int i = 0; i < decks.size(); i++) {
            deckNames[i] = decks.get(i).getName();
        }
        return deckNames;
    }
}
```

This function iterates through all the decks in the decks arraylist and stores each of their names into the deckNames array. It is one of the functions that is implemented from the FramesWithDeckBox interface and usually used when it is required to be displayed in a combobox, which appears in multiple different frames/windows.

flipCard(JLabel wordLabel, JLabel defLabel)

```
default void flipCard(JLabel wordLabel, JLabel defLabel) {
    if (defLabel.isVisible()) {
        resetDefault(wordLabel, defLabel);
    } else {
        defLabel.setVisible(true);
        wordLabel.setVisible(false);
    }
}
```

Another method that is implemented, this time from the CardPanel interface, is flipCard(), which has the function of a ‘flipping’ effect of the average flashcard, showing the definition when it previously showed the word and vice versa.

DeckListRenderer class

```
static class DeckListCellRenderer extends NewDefaultListCellRenderer {
   @Override
   public Component getListCellRendererComponent(JList<?> list, Object value,
int index, boolean isSelected, boolean cellHasFocus) {
       super.getListCellRendererComponent(list, value, index, isSelected,
cellHasFocus);
       JLabel renderer = (JLabel) super.getListCellRendererComponent(list,
value, index, isSelected, cellHasFocus);
       String[] values = value.toString().split("avalidseparator");
       setText("<html><body><div style='display: flex; justify-content:
space-between; align-items: center; font-size: 12px;'>  " +
values[0] + "</div><div style='font-size: 10px'>   " +
values[1] + " card(s)</div></body></html>");
       renderer.setPreferredSize(new Dimension(500,
renderer.getFontMetrics(renderer.getFont()).getHeight() + 40));
       if (cellHasFocus) {
           renderer.setBorder(selectedBorder);
       }
       return renderer;
   }
}
```

One of the ways I utilized inheritance is in customizing the cell renderer of the JSwing JList component. In this case, the getListCellRendererComponent function is overridden by replacing the text to be displayed as well as its font and overall aesthetics. I also added the feature of the list actively showing which item has been selected by changing the border to a thicker one.

d. Further Development

After creating the program, I stumbled upon different ways it can be improved. The following features may be added in upcoming iterations of the program:

1. Customizable themes – Users may select from a number of different theme options (or maybe select their own color palette and fonts) to apply to the program
2. Added game mechanics – There may be a feature to incentivise studying flashcards that mimics game mechanics such as the ability to level up after studying a number of flashcards (which in turn unlocks different rewards)
3. More user intuitive features – When flipping cards, it is more intuitive to indicate the side that is being displayed (showing whether it is word/definition). Additionally, the number of cards left to be studied, and how many the user has studied may be indicated during the study mode itself.
4. Relating deadline to study time – Users may be given the option to relate a deadline to a deck, and the program will be able to curate a specific learning curve for the deck in consideration of the deadline.

III. Resources

- **ChatGPT –** chatgpt.com
- **JAVA GUI YouTube Course –** https://youtu.be/Kmgo00avvEw?si=FGBOcJeeUS9bnp1L
- **json-simple toolkit download –** https://code.google.com/archive/p/json-simple/
- **Dall-E image generator –** https://www.bing.com/images/create/?ref=hn