Documentation for OOP(Object Oriented Programming) Final Project: “Memory Game”

Lecturer:
Mr. Jude Joseph Lamug Martinez, MCS.

Report done by member of Class L2CC:

Micky Malvino Kusandiwinata **2602174522**

[illegible]

....

Binus School of Computer Science Undergraduate Program
Universitas Bina Nusantara
Jakarta
2023

Project Documentation

Table of Contents

*** Link to demo video: https://drive.google.com/file/d/1s-VP8l0b40Da41_v5g_Nwbi5AIihzlwd/view?usp=sharing

*** Link to Github (contains all necessary files): https://github.com/mmalvino/MemoryGame_OOP_FinalProject

Brief Description

This Memory Game is a Java-based project that simulates a classic memory game. This version of the game implements the terminal as an aid for visual representation. The objective of the game is to match pairs of cards with the same fruit names by flipping them over. By default, the game consists of multiple levels going up to level 5, each increasing in difficulty. Each level has 4 cards in total aka 2 pairs of cards you have to match with each other. So 1st level has 4, 2nd has 8, and so on and so forth until the final 5th level which has a total of 20 cards or 10 pairs you need to match. The player's goal is to complete all levels and achieve the highest score possible. There are some changes you can make to the game manually through the code and these are: changing which level you start at, changing the number of cards in a deck at every level (instead of 4 you can do 2 or 6 or 8 or 10, etc), changing the number of levels(instead of 5 you can add or subtract the total number of levels in the game)

The project utilizes object-oriented programming concepts to implement various classes and interfaces. Here's an overview of the key components:

Card: Represents a card in the memory game. Each card has a fruit name and can be face up or face down. The Card class implements the Displayable interface, allowing it to be displayed on the screen.

Deck: Represents a deck of cards in the memory game. The Deck class generates a specified number of pairs of cards with random fruit names. The cards are shuffled to ensure a random order.

Player: Represents a player in the memory game. Each player has a name and a score. The score increases whenever a matching pair of cards is found.

Game: Manages the game logic and flow. The Game class prompts the player for their name and starts the game. It handles the gameplay, including displaying cards, getting user input, and checking for matches. The game progresses through multiple levels, with the number of cards and difficulty increasing with each level.

The MemoryGame class contains the main method, which creates an instance of the Game class and starts the memory game.

To play the game, the player enters their name and proceeds through each level. They will be prompted to enter the index of the card they want to flip. If a match is found, the player's score increases, and the matched cards will turn to a “-“ symbol to specify they are matched. Any attempts to match matched cards will trigger an invalid input message. The game continues until all cards are matched or until the player completes all levels.

Project Specifications

Project Description:
a. The purpose of the project is to create a Memory Game that challenges players to improve their memory skills and concentration. It provides an entertaining way to exercise cognitive abilities by matching pairs of cards with the same fruit. The game promotes mental agility and can be enjoyed by people of all ages.

b. The project utilizes various data structures, such as lists and arrays, to store and manage the cards in the deck. Algorithms are employed for shuffling the cards and checking for matches between flipped cards. Additionally, loops and conditional statements are used to control the game flow and player interactions.

c. The expected outcomes of the project are a functional and enjoyable Memory Game that provides an immersive user experience. The game facilitates cognitive development by offering increasing levels of difficulty, tracking, and displaying player scores.

User Interface (Choose either GUI or CLI):
Command-Line Interface (CLI): A text-based interface that prompts the player for inputs and displays the game state using text-based representations through the terminal. The CLI provides clear instructions(giving the range of input), allows players to enter card indices, shows updates as cards are flipped or matched, and shows different messages when different errors are made to tell the user what to do.

Object-Oriented Design:
a. The project will include objects and classes such as Card, Deck, Player, and Game. The Card class represents a single card in the game, the Deck class manages a collection of cards, the Player class tracks the player's name and score, and the Game class controls the overall game logic.

b. Inheritance and polymorphism to create a hierarchy of classes, such as a base Card class and derived classes for specific types of cards. For example, the Card class can be extended to create different themed decks (e.g., car brands, video game characters, animals, numbers, colors) with additional properties or behaviors.

c. Interfaces, like the Displayable interface in the provided code used to define common behaviors that participating classes implement. This allows for code reusability and ensures that objects can be displayed uniformly.

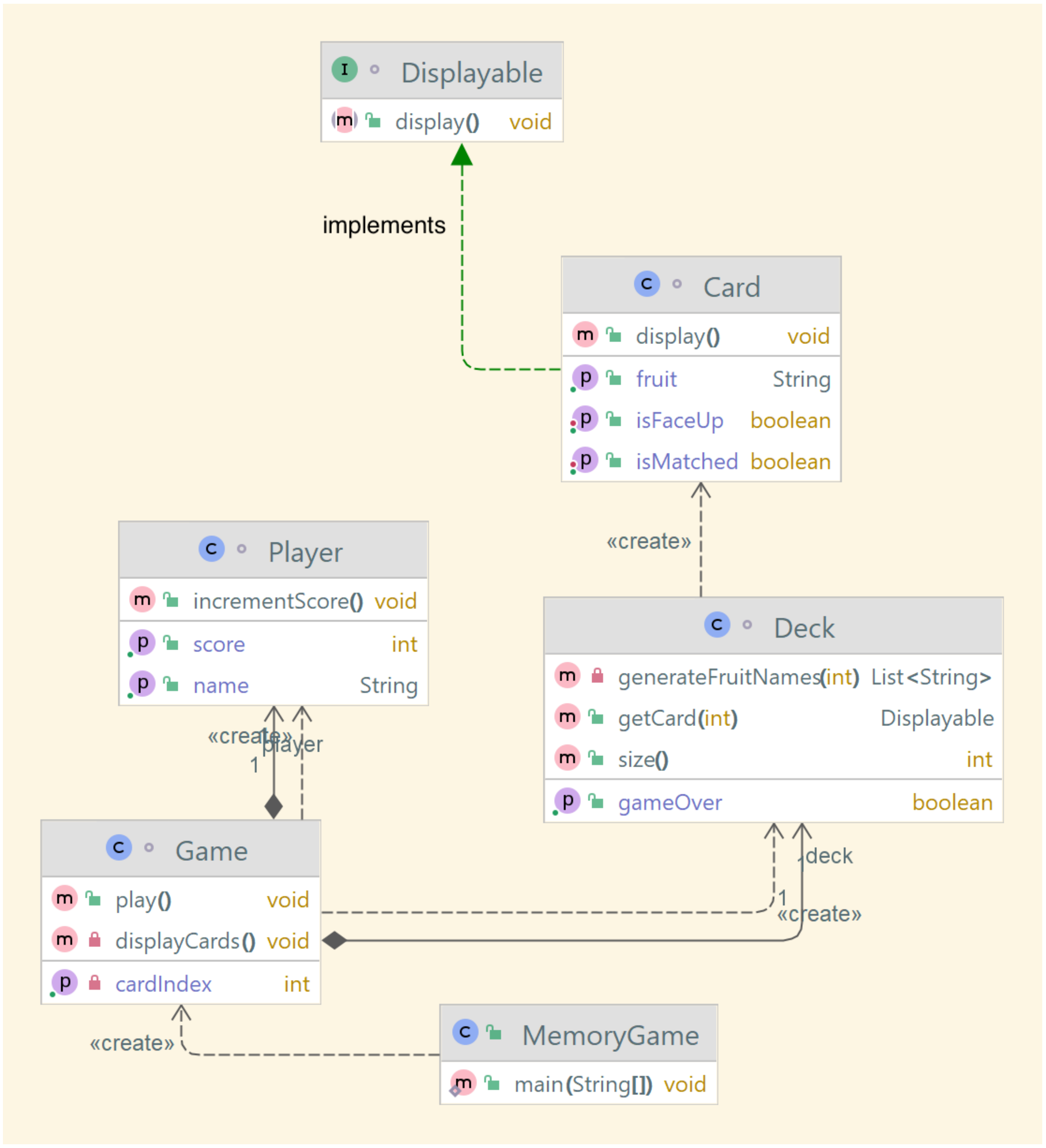
Data Persistence:
The data includes information about players(the name which is entered at the beginning of the game), their scores, and the game state.

Testing and Debugging(Error handling):
Invalid inputs: The code includes some checks to handle invalid user inputs. For example, when the player is prompted to enter a card index, the code validates whether the input is a valid number within the range of available cards. If the input is outside the range, appropriate error messages are displayed, and the user is prompted to enter a valid input. Also, if the user inputs the same number twice hoping to get a match, the system will output an appropriate message saying how the same card has been selected, please try again. If the input is in negative numbers, symbols, or letters, the system will also display error messages and prompts the user to enter a valid input. Lastly, the code checks if all cards in the deck are matched to determine if the game is over. If it's not the game continues and if all cards are matched it will output the congratulatory message along with the user's name and score.

Documentation:
a. Clear and concise documentation has been provided, describing the purpose, functionality, and usage of the program.
b. Comments have been included in the source code to explain the purpose and functionality of each method, class, and important code block.
c. A user manual has been posted on README file in Github to guide users on how to play the game, navigate the user interface, and understand its features.

Project Management:
Version control system, Github, is used to track changes to the codebase. This allows for collaboration, code review, and other necessary actions. It also facilitates project organization and documentation of changes made over time.

Solution Design (Class Diagram):



The Card class implements the Displayable interface, indicating that it provides an implementation for the display() method defined in the interface. This relationship ensures that any Card object can be displayed according to the rules specified in the Displayable interface.

The Deck class has an association relationship with the Card class. It uses a List of Displayable objects to represent a collection of cards in the deck. Since the Card class implements the Displayable interface, it can be added to the list. This association allows the Deck class to manage and manipulate the Card objects within the deck.

The Game class has a composition relationship with the Deck class. This means that the Game class contains and manages a Deck object as part of its internal state. It creates a new instance of the Deck class and uses it to represent the deck of cards in the memory game. This composition relationship allows the Game class to control the lifecycle of the Deck object.

The Game class has an association relationship with the Deck class. This relationship enables the Game class to interact with the Deck object to play the game. It can access and manipulate the cards in the deck, such as flipping them and checking for matches.

The Game class has a composition relationship with the Player class. It contains and manages a Player object as part of its internal state. The Player object represents a player in the memory game and stores the player's name and score. This composition relationship allows the Game class to control the lifecycle of the Player object.

The Game class has an association relationship with the Player class. This relationship enables the Game class to interact with the Player object to track the player's score. It can access the player's score and increment it as the game progresses.

The MemoryGame class has an association relationship with the Game class. This relationship indicates that the MemoryGame class interacts with the Game class to start and play the memory game. It creates an instance of the Game class and calls its play() method to initiate the game.

Modules, essential algorithms, data structures, and additional features used in the program:

Modules:

- MemoryGame: The main class that starts the memory game and controls the game flow.
- Game: Represents the game logic and flow, including playing levels, managing the deck, and handling player input.
- Deck: Represents a deck of cards in the memory game, including generating pairs of cards, shuffling them, and checking if the game is over.
- Card: Represents a card in the memory game, including its fruit name, face-up/face-down state, and matched status.
- Player: Represents a player in the memory game, including their name and score.
- Displayable (interface): Provides the display() method that objects can implement to display themselves.

Essential Algorithms:

- Shuffle: The Collections.shuffle() method is used to shuffle the cards in the deck. It randomly rearranges the order of the cards to provide a more challenging and varied game experience.

Data Structures:

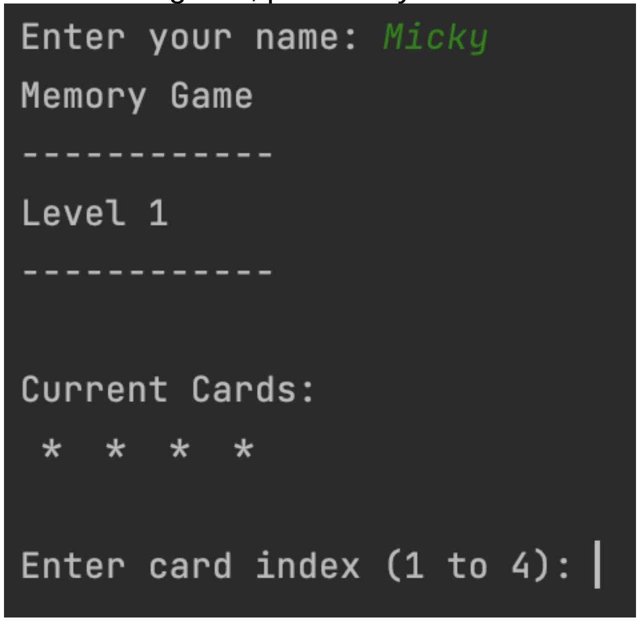
- List: The java.util.List interface is used to represent a collection of cards in the deck. An ArrayList implementation is used to store and manipulate the cards efficiently.
- String Array: An array of strings is used to store the available fruit names. It is used to generate pairs of cards with the same fruit.

Additional Features:

- User Input: The Scanner class is used to get user input for player names and card selections. It allows the player to interact with the game by entering their name and choosing card indices.
- Level System: The game implements a level system where the player progresses through multiple levels. Each level increases the difficulty by increasing the number of card pairs to match.
- Scoring System: The Player class keeps track of the player's score, which is incremented when a match is found. The score represents the number of successful matches made by the player.
- Display: The Displayable interface is used to provide a consistent way to display various objects, including cards. It allows objects to define their display behavior and facilitates the rendering of the game's current state**.**

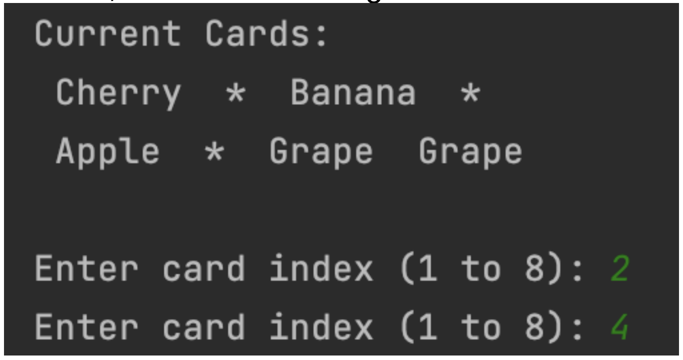
Screenshots of the application along with information and evidence

Start of the game, punch in your name then hit enter, it will bring you to level 1



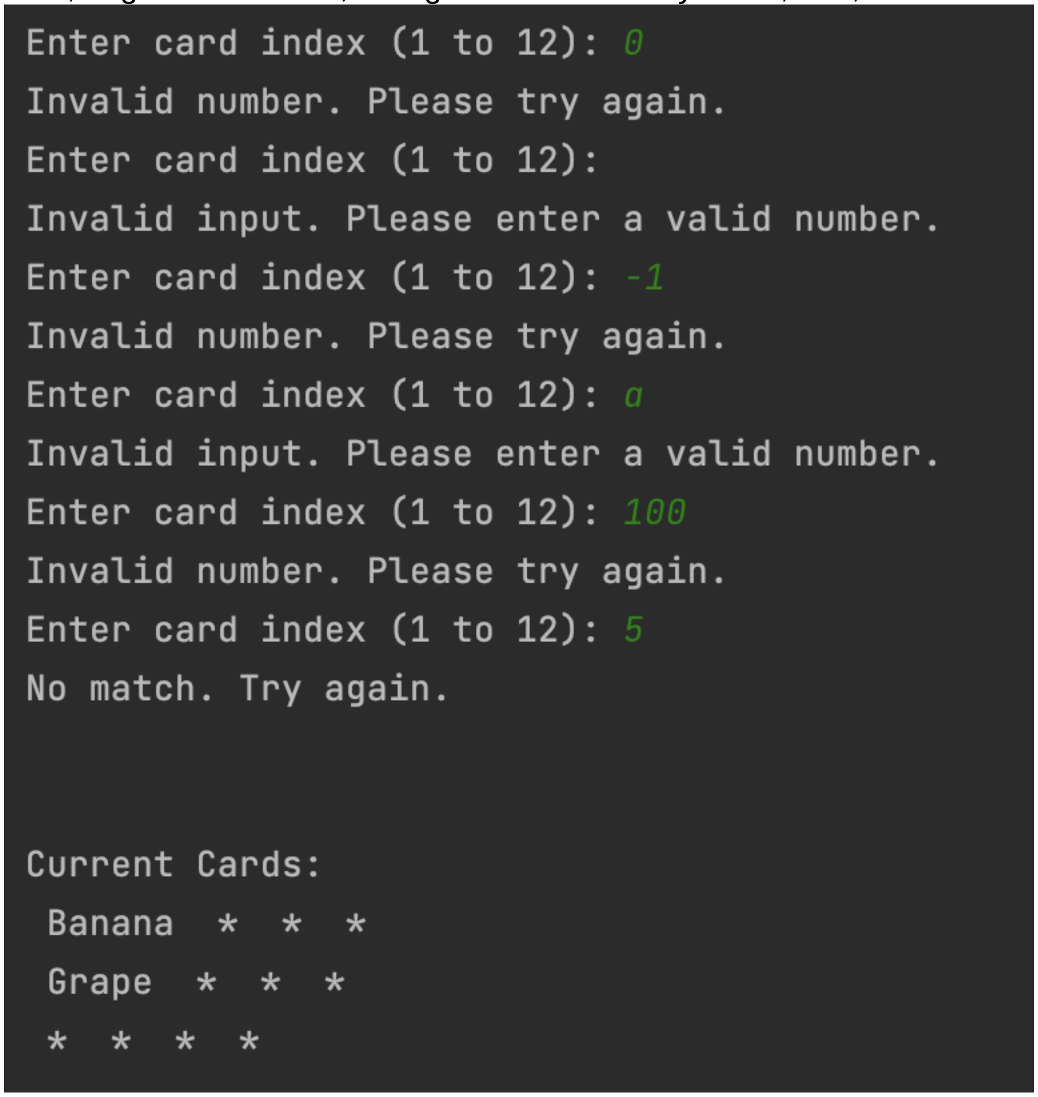
```
Enter your name: Micky
Memory Game
------------
Level 1
------------

Current Cards:
 *  *  *  *

Enter card index (1 to 4): 
```

Level 2, number of cards goes from 4 to 8



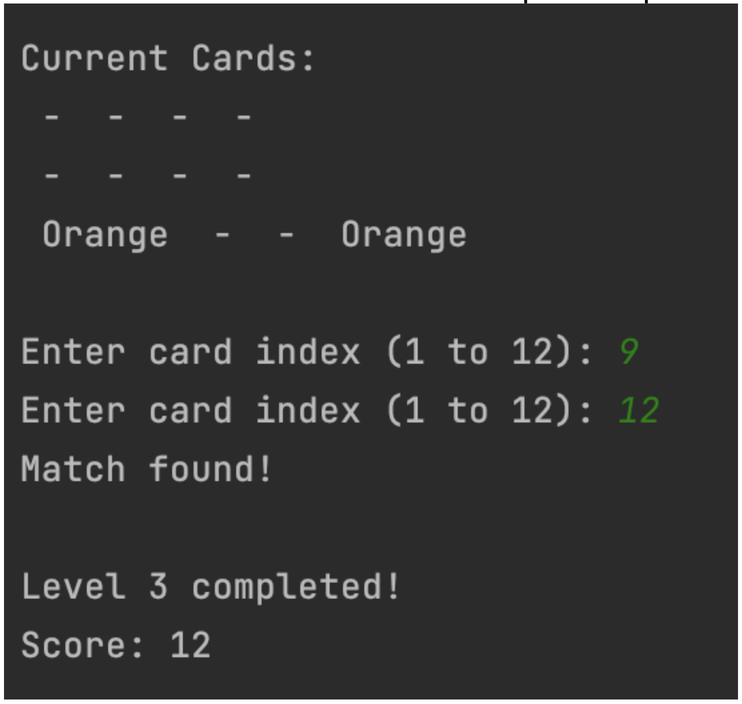
```
Current Cards:
 Cherry  *  Banana  *
 Apple  *  Grape  Grape

Enter card index (1 to 8): 2
Enter card index (1 to 8): 4
```

Some error handling, appropriate error messages will pop up depending on the Situation. Anything outside the range of appropriate values will be denied, whether it be 0, negative numbers, hitting enter without any value, 100, etc.



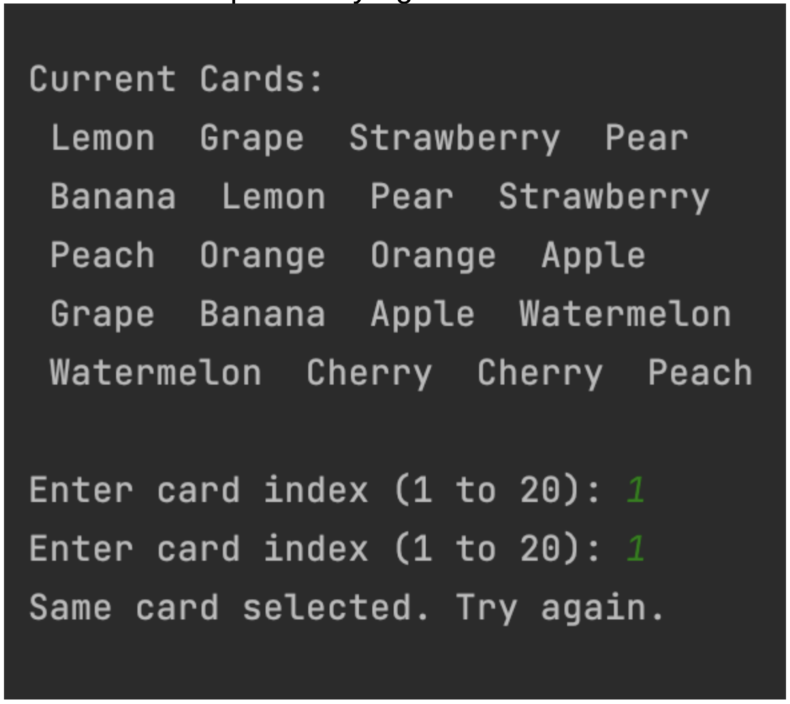
```
Enter card index (1 to 12): 0
Invalid number. Please try again.
Enter card index (1 to 12):
Invalid input. Please enter a valid number.
Enter card index (1 to 12): -1
Invalid number. Please try again.
Enter card index (1 to 12): a
Invalid input. Please enter a valid number.
Enter card index (1 to 12): 100
Invalid number. Please try again.
Enter card index (1 to 12): 5
No match. Try again.



Current Cards:
 Banana  *  *  *
 Grape  *  *  *
 *  *  *  *
```

Message pops up when a level is completed and it tallies up the score, in this case level 3 was cleared and total score up to this point accumulates to 12 points



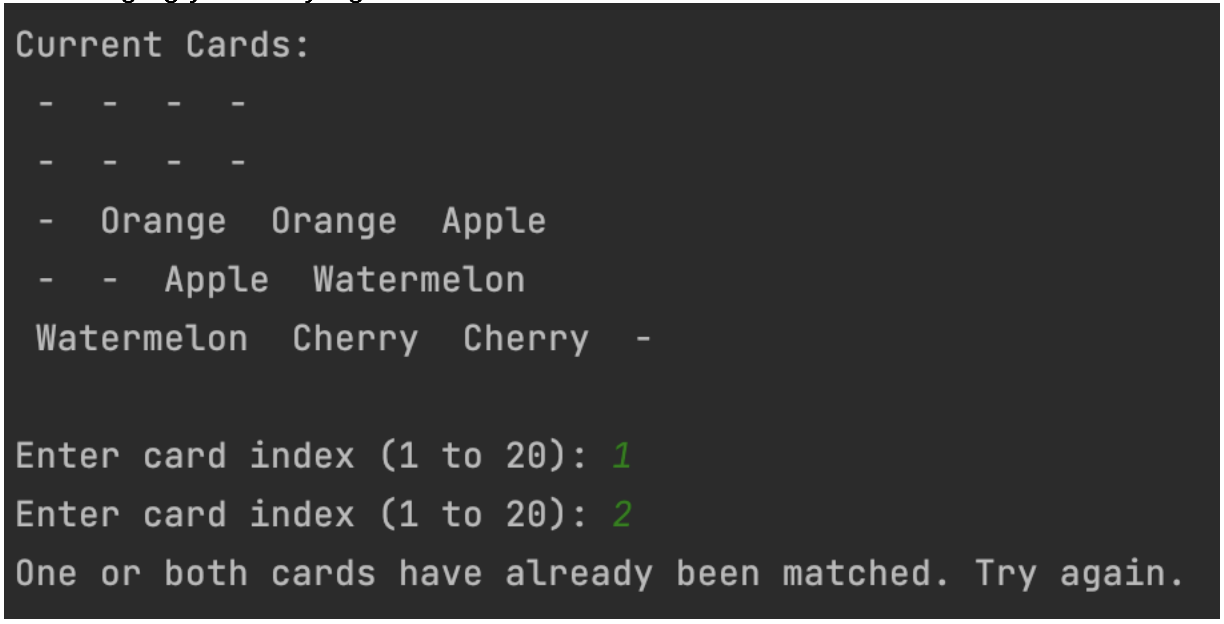
```
Current Cards:
 -   -   -   -
 -   -   -   -
 Orange  -   -  Orange

Enter card index (1 to 12): 9
Enter card index (1 to 12): 12
Match found!

Level 3 completed!
Score: 12
```

If same card is selected, it will pop up an error message saying the same card is selected and to please try again



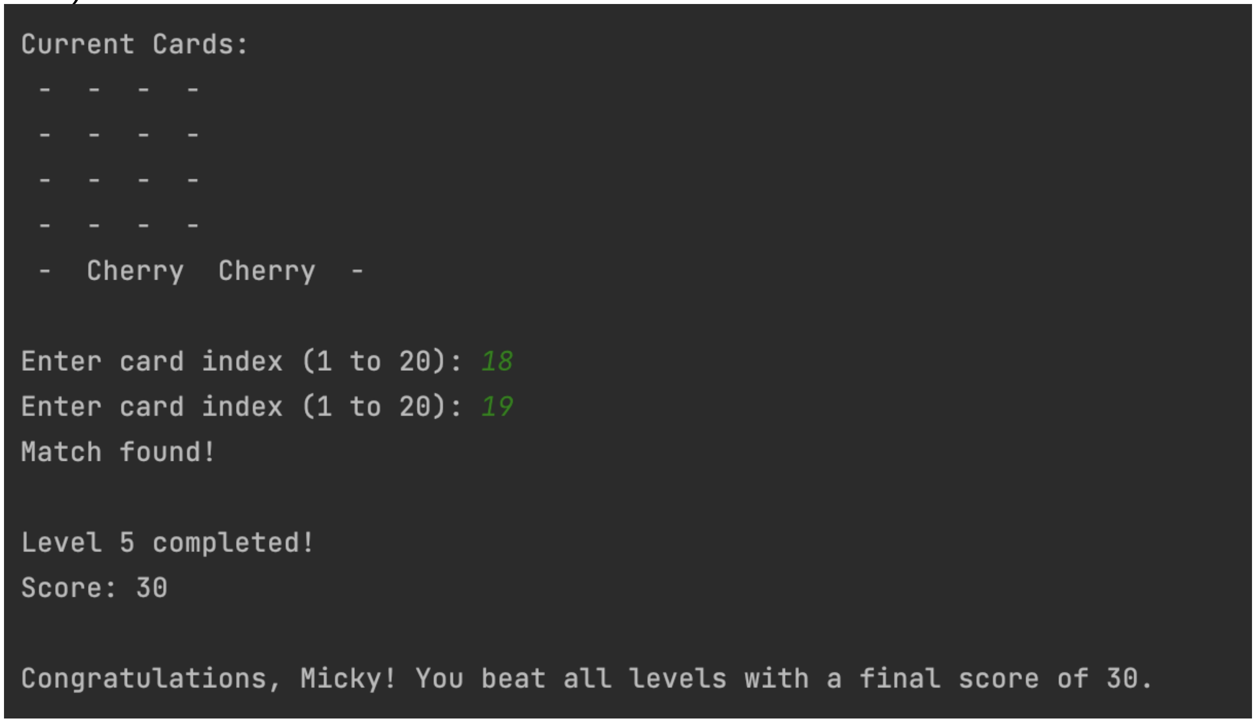
```
Current Cards:
 Lemon  Grape  Strawberry  Pear
 Banana  Lemon  Pear  Strawberry
 Peach  Orange  Orange  Apple
 Grape  Banana  Apple  Watermelon
 Watermelon  Cherry  Cherry  Peach

Enter card index (1 to 20): 1
Enter card index (1 to 20): 1
Same card selected. Try again.
```

If you try to match cards that are already matched, this message will pop up encouraging you to try again!



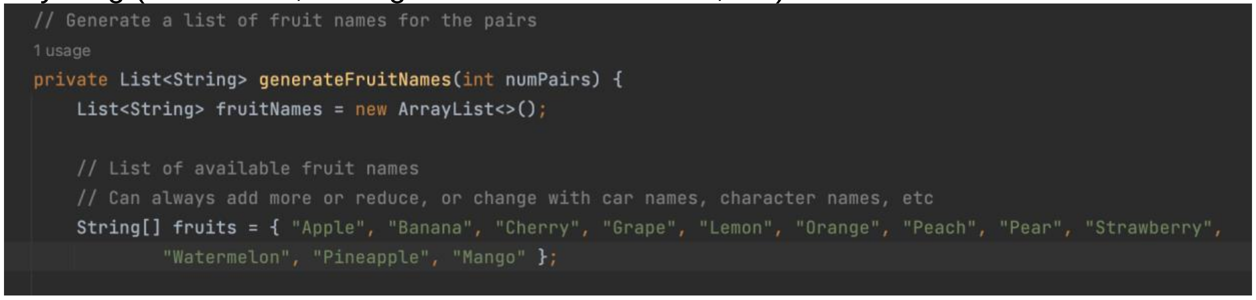
```
Current Cards:
 -   -   -   -
 -   -   -   -
 -   Orange  Orange  Apple
 -   -   Apple  Watermelon
 Watermelon  Cherry  Cherry  -

Enter card index (1 to 20): 1
Enter card index (1 to 20): 2
One or both cards have already been matched. Try again.
```

A congratulatory message with your name you input in the beginning along with the final score. In this case, level 5 is the final level (you can change this to however you want).



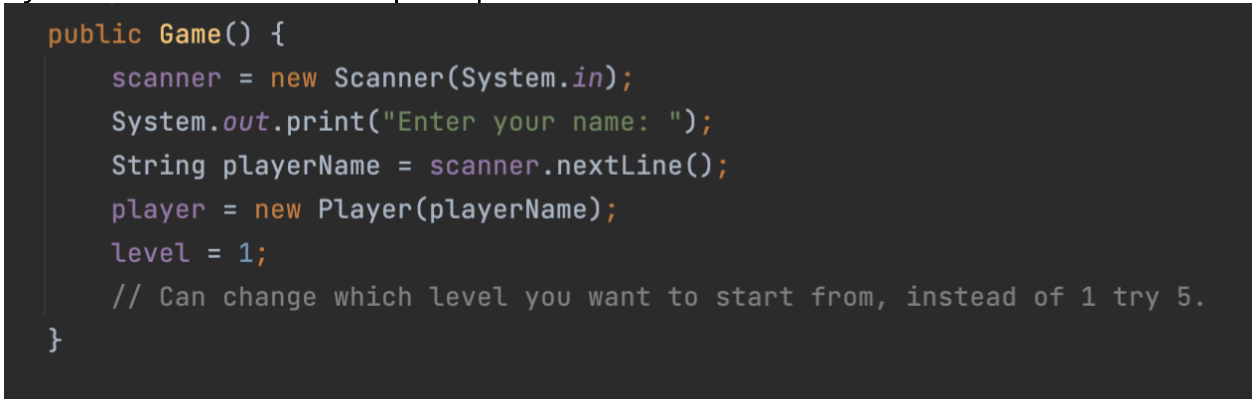
```
Current Cards:
 -   -   -   -
 -   -   -   -
 -   -   -   -
 -   -   -   -
 -   Cherry  Cherry  -

Enter card index (1 to 20): 18
Enter card index (1 to 20): 19
Match found!

Level 5 completed!
Score: 30

Congratulations, Micky! You beat all levels with a final score of 30.
```

You can add more fruit names or remove some, you can also switch fruit names with anything (car names, videogame character names, etc)



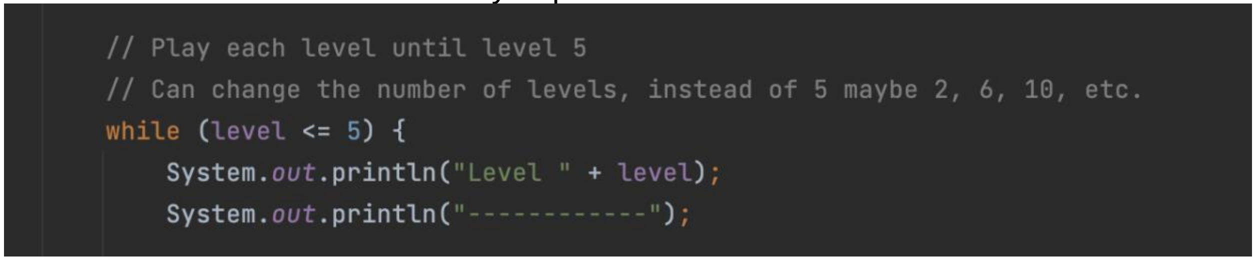
```java
// Generate a list of fruit names for the pairs
1 usage
private List<String> generateFruitNames(int numPairs) {
    List<String> fruitNames = new ArrayList<>();

    // List of available fruit names
    // Can always add more or reduce, or change with car names, character names, etc
    String[] fruits = { "Apple", "Banana", "Cherry", "Grape", "Lemon", "Orange", "Peach", "Pear", "Strawberry",
            "Watermelon", "Pineapple", "Mango" };
```

You can choose which level you start from, instead of starting from level 1, you can try to start from level 3 or 5 perhaps



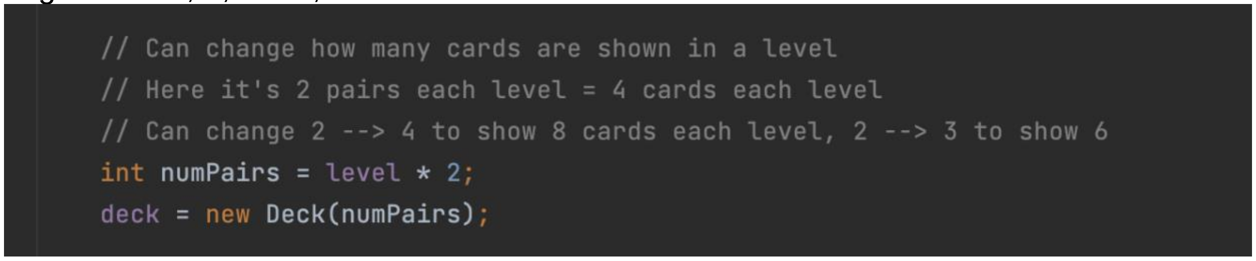
```java
public Game() {
    scanner = new Scanner(System.in);
    System.out.print("Enter your name: ");
    String playerName = scanner.nextLine();
    player = new Player(playerName);
    level = 1;
    // Can change which level you want to start from, instead of 1 try 5.
}
```

You can also change the number of levels in the game, instead of 5 levels you may want to increase or decrease it as you please



```java
    // Play each level until level 5
    // Can change the number of levels, instead of 5 maybe 2, 6, 10, etc.
    while (level <= 5) {
        System.out.println("Level " + level);
        System.out.println("------------");
```

You can change how many cards are shown in a level, instead of 4 cards shown you might want 6, 8, or 16, etc.



```java
    // Can change how many cards are shown in a level
    // Here it's 2 pairs each level = 4 cards each level
    // Can change 2 --> 4 to show 8 cards each level, 2 --> 3 to show 6
    int numPairs = level * 2;
    deck = new Deck(numPairs);
```