**Documentation – Matching cards Game Project**

**Abhigna Kappagantula**

**SE22UCSE003**

**1. Project Overview**

The Pair Match Game is a memory-based matching game where players are presented with a grid of cards. The objective is to match pairs of cards with the same image by flipping them over two at a time. The game ends when all pairs are matched, and errors are tracked during the game. The project is implemented using Java with a graphical user interface (GUI) built with Swing and AWT.

**2. Features**

* A 4x6 grid of cards is used for the game board.
* Cards have images representing various food items.
* Cards are flipped when clicked, and players must match pairs of cards.
* Errors are counted, and feedback is shown to the player.
* The game can be restarted at any time.
* The cards briefly show their images at the start of the game for better memorization.

**3. Tools and Technologies**

**3.1 Programming Languages and Platforms**

* **Java**: Used to write the core logic and implement the game mechanics, such as handling card selection, checking for matches, and maintaining game state.
* **Swing (javax.swing)**: Used to create the graphical user interface, including buttons, labels, and panels.
* **AWT (Abstract Window Toolkit - java.awt)**: Provides layouts, events, and rendering support for the graphical interface.
* **Operating System**: Windows was used for development and execution of the program.

**3.2 Libraries and Frameworks**

* **Swing (javax.swing)**: Used for creating GUI components like buttons (JButton), labels (JLabel), and panels (JPanel).
* **AWT (Abstract Window Toolkit - java.awt)**: Used for handling layout management and event handling. It is essential for creating and managing the game’s visual layout.

**4. Project Structure**

The project is divided into the following classes and components:

**4.1 PairMatchGame.java**

* **Core Game Logic**: Contains the logic for handling card flips, matching, error counting, and updating the user interface. It uses Swing components for visual elements.
* **Card Class**: A nested class that represents each card in the game. Each card has a name and an associated image.

**4.2 Launcher.java**

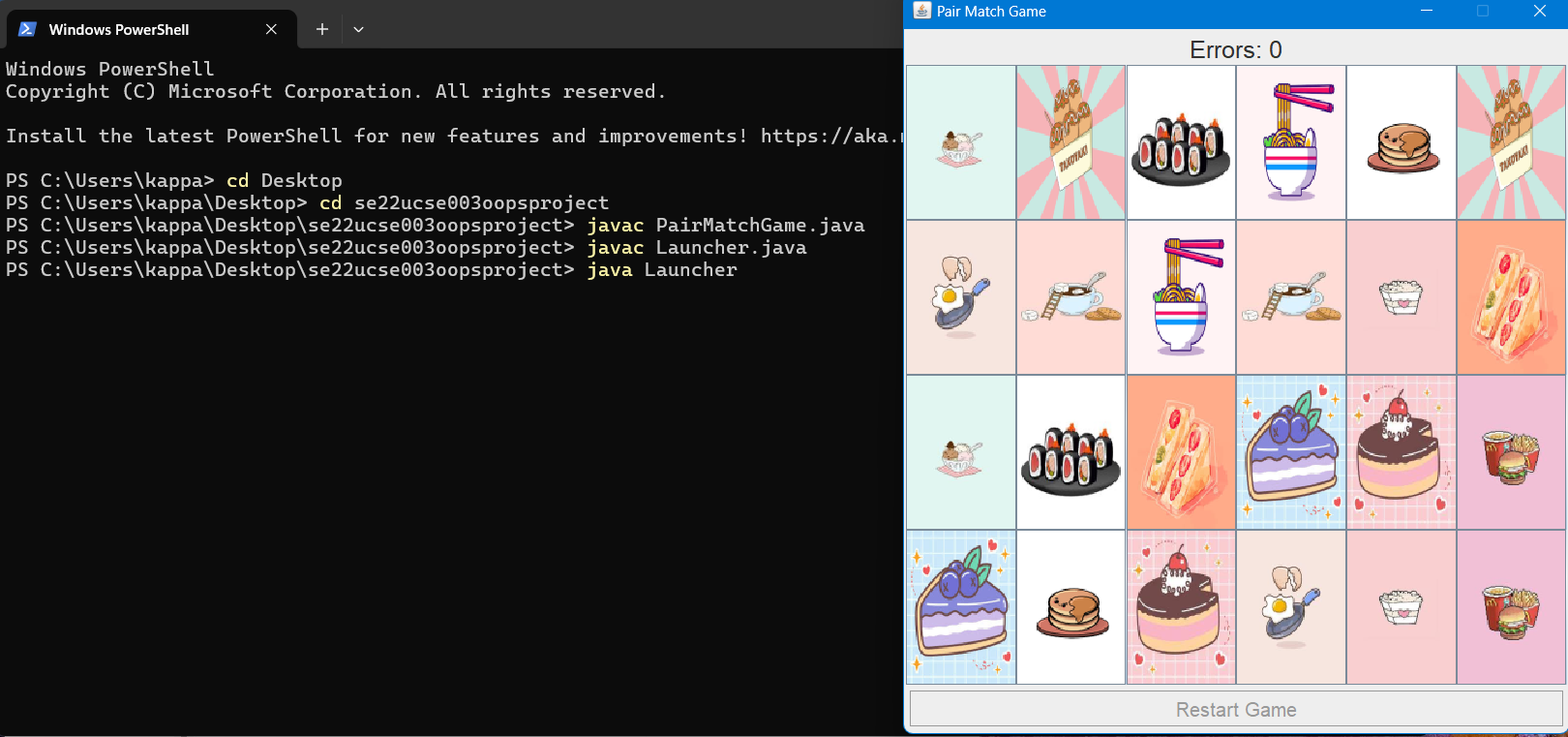
* **Entry Point**: This file contains the main function, which initializes the game by creating an instance of PairMatchGame.

**4.3 Image Files**

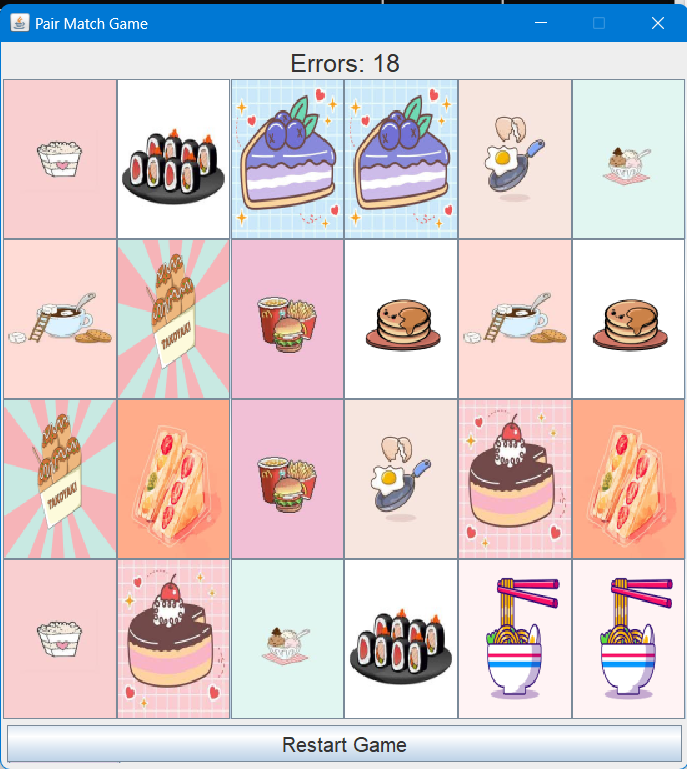
* **img**: Stores the card images (e.g., burgerfries.jpg, cake.jpg, etc.) and back.jpg for the back face of the cards. These images are used for the front and back of each card displayed in the game. The images are fetched from the img folder where they are stored in their original format without resizing.

**5. Game Flow and Main Logic**

1. **Card Initialization**:
   * A list of card names and their corresponding images is created and used to initialize the cards. The images are loaded into ImageIcon objects and associated with each card.
   * The list of cards is then duplicated to create pairs, and they are shuffled randomly.
2. **Game Board Setup**:
   * A grid layout (GridLayout) is used to arrange the cards. Each card is represented by a JButton, which initially displays the back face of the card (backImage).
   * When a player clicks on a card, its front face (image) is shown. The first and second cards selected are compared to check for a match.
3. **Error Handling**:
   * When two cards are selected that do not match, the error count is incremented, and the cards are hidden after a brief delay using a Timer.
4. **Winning Condition**:
   * The game ends when all pairs are matched. The restart button becomes available for the player to restart the game.
5. **Output**

****





**7. Code Explanation**

**PairMatchGame.java:**

**Card Class**

* Represents a card with a name and a front-facing image (face). The toString method is overridden to return the card’s name, which is used when displaying the card’s label.

**initializeCards()**

* Initializes the cards with names and images. The cards list holds a pair of each card by adding the list to itself. The back image (backImage) is set for each card initially.

**shuffleCards()**

* Shuffles the cards using a random index to ensure the game is not predictable.

**Game Setup and Action Handling**

* A grid of JButton components is created, each representing a card. The cards are arranged in a GridLayout with a setPreferredSize for each button to match the size of the images.
* The ActionListener for each button flips the card when clicked. If two cards are selected and they do not match, an error is logged and the cards are hidden after a brief pause using a Timer.

**hideCards()**

* If two cards are not a match, this method hides both cards after a brief delay, using a Timer to schedule their reset.

**showCardsBriefly()**

* At the beginning of the game, all cards are shown for a brief period (1 second) to allow the player to memorize their positions before they are hidden again.

**Game Restart**

* The restart button allows the game to be reset. All cards are shuffled, and the error count is reset to zero.

**8. OOP Principles in the Project**

The project follows the core Object-Oriented Programming (OOP) principles:

**1. Encapsulation**

* The class Card encapsulates all properties and methods related to the card, such as its name, image, and behavior (e.g., flipping). The game logic in PairMatchGame accesses these properties and methods without directly manipulating them, providing a clean and manageable interface for working with cards.

**2. Abstraction**

* The internal details of card matching and game logic are hidden within the PairMatchGame class, and the user interacts only with the buttons and the visible GUI elements. The game mechanics, such as card selection, shuffling, and matching, are abstracted away from the user, allowing for a clean and focused interaction with the game.

**3. Inheritance**

* Though this project doesn’t heavily use complex inheritance hierarchies, the Card class could be extended or modified in the future to introduce more specialized card types (e.g., special cards with different rules) without affecting the rest of the game logic.

**4. Polymorphism**

* Polymorphism is applied where the ActionListener interface is used in PairMatchGame to handle the card flip actions. The same ActionListener can handle multiple JButton objects (each representing a card) with different behaviors, making the game more flexible and scalable.

**9. Known Issues**

* The game does not have an option to adjust the grid size or change the difficulty level.

**10. Conclusion**

The Pair Match Game provides a simple yet fun memory challenge. It demonstrates fundamental concepts of Java GUI development, including event handling, randomization, and state management. It also provides a good learning experience in working with Swing and AWT components while following OOP principles such as encapsulation, abstraction, inheritance, and polymorphism.