Old Maid Game Implementation and Multi-Threading

Introduction:

In this report, I’ll discuss the implementation of an Old Maid card game using Java. The game involves multiple players who take turns to play and follow specific rules to eliminate pairs of cards. We'll delve into how multi-threading was utilized to enable players to play concurrently and manage the game's flow.

Old Maid Game Overview:

The Old Maid game is a classic card game played with a standard deck of cards. The objective is to form pairs of matching cards and discard them, with the goal of being the first player to empty their hand completely.

1. Object-Oriented Design:

The essence of my code lies in its adherence to object-oriented design principles, which fosters modularity, reusability, and maintainability. Each class in my code represents a distinct entity, encapsulating its attributes and behavior. The `Card` class acts as a foundation for various card types, embodying polymorphism by allowing specialized cards to inherit common attributes. Interfaces like `Deck` define abstract behaviors, enabling multiple implementations to cater to varying deck and discard pile structures.

Furthermore, the Singleton pattern is leveraged through the `OldMaid` class, ensuring a singular instance of the game. In contrast, the Strategy pattern is applied via the `Card` class and its concrete subclass `StandardCard`, allowing different card types to exhibit specific play behaviors. This design fosters flexibility and extensibility for potential future enhancements.

1. Multi-Threading Approach:

I make sure Multi-threading is used to allow players to take their turns concurrently, simulating real-time gameplay. Each player's actions are encapsulated within a separate thread, enabling them to play without waiting for other players. This parallel execution enhances the game's interactivity and realism.

**PlayerThread Class**

The PlayerThread class extends the Thread class and encapsulates a player's actions in a separate thread. This class controls the player's turn, ensuring synchronization and proper execution. The thread performs the following steps in a loop:

Checks if it's the player's turn by comparing the current player index with the thread's index.

Displays the game status and the player's hand.

The player draws a card from the previous player's hand, ensuring proper synchronization.

The player throws pairs of cards from their hand.

If the player's hand is empty, they are removed from the game, and the current player index is updated.

Notifies the next player to play their turn.

**OldMaid Class**

The OldMaid class orchestrates the entire game. It initializes players, manages the deck, and controls the game's flow. It implements multi-threading to allow players to play concurrently.

Initializes players, creates their threads, and starts them.

In a loop, players take turns one by one using synchronization.

After all players finish their turns, remaining players are removed from the game, and indexes are adjusted accordingly.

1. Clean Code Principles:

My code strives to adhere to clean code principles, as advocated by Uncle Bob. One key principle is the Single Responsibility Principle (SRP), wherein each class serves a specific purpose. For instance, the `OldMaidDeck` class focuses solely on deck management, while the `OldMaidPlayers` class is dedicated to player actions.

The Open/Closed Principle is also upheld through extensible design. By utilizing abstract classes and interfaces, my code can accommodate new card types, decks, and game variations without altering existing code. Moreover, my code emphasizes readability, with meaningful variable and method names, informative comments, and structured organization.

1. Pseudocode Design and Logic Overview:

As I developed the Old Maid card game code, I found it beneficial to outline the logic and flow using pseudocode before delving into actual coding. This approach allowed me to plan and organize the gameplay's sequence of actions without getting bogged down in specific programming syntax. Below, I've provided a high-level pseudocode representation of how a player's turn is orchestrated within the PlayerThread class:

**method run():**

**while true:**

**synchronized(player):**

**if only one player left:**

**announce the winner**

**add winner to the result list**

**return // Exit the game loop**

**if it's not current player's turn:**

**wait for notification**

**continue // Skip this iteration of the loop**

**displayGameStatus()**

**displayPlayerHand()**

**draw a card from the previous player**

**throw a pair of cards if possible**

**synchronized(player):**

**if player's hand is empty:**

**remove player from the game**

**update next player's turn**

**notify all players**

**continue // Skip the rest of the loop**

**slesep for a short time**

This pseudocode provides a clear outline of the steps a player takes during their turn and how the thread synchronization mechanisms ensure orderly execution. By having this pseudocode, I was able to visualize the entire process and ensure that the multi-threaded gameplay maintains data consistency and a seamless experience.

1. Conclusion:

My Old Maid card game code is crafted with meticulous attention to object-oriented design principles and thread synchronization mechanisms. This design encapsulates the essence of clean code principles, fostering a modular, maintainable, and immersive gaming experience. By adhering to these principles, I've established a foundation that can be expanded upon, enhanced, and refined in the spirit of continuous improvement.