The Old Maid Card Game

- Card Class: Represents a playing card with face value and suit.
  - **Functionality**:
    - Initializes card with specified face and suit values.
    - Provides methods to get face, suit, and a string representation of the card.

- CardList Class: **:** a collection of cards.
  - **Functionality:**
    - Manages a list of cards, allowing addition, removal, dealing, and shuffling.
    - Includes methods to get the size, retrieve specific cards, and check if more cards are available.

- Deck Class (inherits from CardList): a deck of playing cards.
  - **Functionality:**
    - Initializes a full deck and shuffles it.
    - Inherits from CardList (extends its functionality)

- OldMaidHand Class (inherits from CardList):
  - **Functionality:**
    - Extends CardList and includes methods specific to the Old Maid game, such as removing pairs and displaying the hand.

- OldMaidPlayer Class (inherits from Player):
  - **Functionality:**
    - Inherits from the generic Player class.
    - Implements Old Maid specific card removal logic.

- OldMaid Class: Main class
  - **Functionality:**
    - Manages player input, game flow, and scoring.
    - Utilizes threads to synchronize player turns and ensure fair gameplay.

- Player Class: the player in a card game and his info
  - **Functionality:**
    - Tracks player statistics such as wins, losses, and points.
    - Provides methods to update and retrieve player information

a. Object-Oriented Design in the Code:

- encapsulated related functionalities within classes.
- i used inheritance to create specialized classes like OldMaidHand and OldMaidPlayer.
- **SOLID principles**

Single Responsibility Principle (SRP):

- Classes like Card, CardList, OldMaidHand, OldMaidPlayer, and Player each have distinct responsibilities, such as managing cards, players, and game logic.

open/Closed Principle (OCP):

- The OldMaidHand and OldMaidPlayer classes can be extended to add new functionalities without modifying the existing code. This promotes maintainability and extensibility.

b. thread synchronization mechanisms used in my code

- The OldMaid class utilizes threads to manage player turns in the game.
- The synchronized block ensures that only one player can perform their turn at a time, preventing conflicts in accessing and modifying shared game data.
- The use of wait(), notifyAll(), and synchronized blocks enables orderly turn-taking among players, avoiding race conditions.

c. clean code principles (Uncle Bob)

avoided using switch statements because they were challenging to read and modify instead, I used enums.

I kept it simple, stupid. (Simpler always better). I reduced complexity as much as possible. I wrote meaningful comments. If a comment was necessary.

I try as hard as possible to minimize dependencies and avoid duplicating of code , Use as few elements as possible, and ensure it is readable and easyto test by make it maintainable.

used the DRI principle.