Cover Page

Name	Student ID	Contribution
Udit Tripathi	720029757	50
Victor Rusu	710029903	50

Development Log

Date and time	Session Duration	ID 720029757	ID 710029903
01/11/2023, 7PM	1.5 hours	Driver	Observer
8/11/2023, 6PM	2 hours	Observer	Driver
12/11/2023, 8PM	1 hours	Observer	Driver
15/11/2023, 7PM	1.5 hours	Driver	Observer
18/11/2023, 7PM	2 hours	Observer	Driver
20/11/2023, 8PM	3 hours	Driver	Observer
24/11/2023, 6PM	3.5 hours	Observer	Driver
25/11/2023, 8PM	3 hours	Driver	Observer
26/11/2023, 6PM	4 hours	Observer	Driver

Design Choices	**reasoning**
Object-Oriented Design	Each class (Card, Player, CardGame) summarises related functionality, making the code more maintainable and extendable.
Thread Safety with Locks:	Ensuring thread safety is crucial in a multi-threaded environment. The use of locks, such as synchronized and ReentrantLock, helps prevent data races and maintains the integrity of shared data structures like player hands and decks.
Thread Communication:	The Object (gameLock) is used for thread communication to coordinate player turns. This avoids busy waiting and ensures that players take turns in a synchronized manner. wait() and notifyAll() are used for effective communication between threads.

Pack Initialization and Distribution:	Distributing cards in a round-robin fashion ensures fairness among players. Each player receives an equal share of cards from the pack, promoting a balanced and competitive game.
Player Actions and Discards:	The simple game strategy of drawing from the left and discarding to the right ensures that players actively participate in the game, providing an interesting and dynamic gameplay experience.
Error Handling:	Robust error handling is implemented to check for invalid pack files and ensure the game starts with the correct number of players. This helps catch issues early and provides informative error messages to users for troubleshooting.
Scalability:	The design allows for scalability by separating concerns into distinct classes. Additional players or enhancements can be easily accommodated without significant changes to the existing codebase, making the system scalable and adaptable.
Ring topology	design incorporates a ring topology for players and decks, distributing cards and forming a circular connection. This adds an interesting structure to the game.

Design pattern

Singleton Pattern (Partial): The CardGame class has a static constant gameLock used for synchronization.

Factory Method Pattern (Partial): The constructors in the Card, Player, and CardGame classes can be seen as factory methods. They are responsible for creating instances of their respective classes.

Encapsulation Principle: The code adheres to the principle of encapsulation by encapsulating the internal state of the classes and providing controlled access through methods.

Separation of Concerns: The code follows the principle of separating concerns by having distinct classes (Card, Player, CardGame) each responsible for a specific aspect of the system.

Ring Topology Pattern : While not a standard design pattern, the decision to organize players and decks in a ring topology adds a specific structure to the game logic.