Introduction

The **HiFive** game is a multiplayer card game built using the JCardGame and JGameGrid libraries. The game facilitates interaction among multiple players—each utilizing unique strategies to discard cards and accumulate points. This report evaluates the game's design and implementation, with a focus on refactoring decisions, extensibility, and adherence to well-established design principles such as GRASP, SOLID, and GoF patterns.

Thought Process Before Coding

Before writing any code, I started by analyzing the problem and identifying the core components and responsibilities that the game would need. I knew that the game had to manage different player behaviors (human and AI), handle card interactions, and track scores across rounds. With this in mind, I began by conceptualizing the **domain model**.

- I identified the main objects, such as HiFive (responsible for the game flow), Player (for different player types), and Card, Rank, and Suit to represent the deck. I also considered how each player would have a unique strategy for selecting cards, which led me to create the Player interface and different classes that implement it (e.g., CleverPlayer, BasicPlayer).
- Once I had a clear understanding of the relationships between these objects, I created the **domain model**, which depicted how the objects would interact with each other. This included defining associations between players and the game and outlining methods for core functionalities like card selection and score calculation.
- Next, I created the system sequence model, which detailed the flow of actions during a game. This sequence model helped me visualize how different components would interact, such as how the HiFive class would delegate actions to different player types, handle user input, and manage the overall game state.

Only after establishing a clear design did I begin writing the code. This approach ensured that the game's architecture was well-planned and capable of supporting future extensions.

Design Analysis Based on GRASP and SOLID Principles

The design of **HiFive** reflects a strong foundation in object-oriented principles, particularly GRASP and SOLID.

1. **Single Responsibility Principle (SRP)**

Each class in the **HiFive** game has been carefully designed to ensure that it has one, and only one, reason to change:

- HiFive class: Manages the overall game logic, score calculation, and game progression.

- Player classes: Handle individual player strategies for card discarding. The CleverPlayer, BasicPlayer, and RandomPlayer implement their respective strategies through the selectCardToDiscard method.
- **PropertiesLoader**: Focuses solely on loading configuration properties from the external file.

By adhering to SRP, the codebase remains modular and maintainable, allowing future enhancements (e.g., introducing new player types) without requiring significant changes.

2. **Open/Closed Principle (OCP)**

The game's design adheres to the OCP by allowing extensions in functionality without modifying existing code. For instance:

- Player Strategy: New player strategies can be introduced by simply adding new classes that implement the Player interface.
- Score Calculation: The scoreForHiFive method in HiFive could be extended to introduce additional scoring rules without altering existing logic.

3. **Liskov Substitution Principle (LSP)**

All player types (CleverPlayer, BasicPlayer, RandomPlayer, and HumanPlayer) adhere to the Player interface and can be used interchangeably in the game logic. This is ensured by designing the interface in a way that all players fulfill their contract without requiring changes in the game flow.

4. **Interface Segregation Principle (ISP)**

The Player interface is minimal and focused, including only the selectCardToDiscard and playCard methods. This design ensures that implementing classes are not forced to define unnecessary methods, adhering to ISP.

5. **Dependency Inversion Principle (DIP)**

Higher-level modules like HiFive depend on abstractions (Player interface) rather than specific implementations (CleverPlayer, RandomPlayer). This allows flexibility in swapping out player behaviors as required, adhering to DIP.

GoF Design Patterns

The **HiFive** game leverages several Gang of Four design patterns, ensuring flexibility and reducing tight coupling between components.

1. Strategy Pattern

The **Strategy Pattern** is used to define different player behaviors:

- The Player interface defines the general strategy for all player types, and the selectCardToDiscard method is implemented differently by CleverPlayer, BasicPlayer, RandomPlayer, and HumanPlayer. This allows flexibility in changing player behavior without modifying the core game logic.

2. Factory Pattern
In the initializePlayers method, player objects are instantiated based on the game properties. This dynamic player creation is achieved using a factory-like approach, abstracting the instantiation logic from the main game flow.

3 Observer Pattern (Implicit)

The event-driven nature of the game's user interaction employs an implicit **Observer Pattern**. For instance, human player actions (e.g., selecting a card) trigger updates in the game state, such as scoring and card display updates.

Refactoring and Design Decisions

In order to enhance the extensibility and modularity of the game, several refactoring decisions were made:

1 Separation of Concerns

Initially, game logic and player behaviors were intertwined, resulting in tight coupling between game mechanics and player actions. The refactored design separates these concerns:

- HiFive now handles game flow, while player-specific behaviors are delegated to individual classes like CleverPlayer, RandomPlayer, and BasicPlayer.

2 Improved Extensibility

The design is structured to accommodate future extensions:

- Player Strategies: New strategies can be added without modifying existing code. The use of the Player interface ensures that all player types follow a consistent contract.
- Game Modes: The game's rules and scoring system can be extended by modifying the methods within HiFive (e.g., by adding more complex scoring rules).

Domain Class Model and Sequence Diagram

Domain Class Model
The domain class model clearly depicts the relationships between the game components. The HiFive class manages the game state, interacts with player objects, and controls the flow of the game. Each player implements the Player interface, allowing flexibility in behavior.

- **Key Classes**:
  - HiFive: Manages the game flow, score calculation, and interactions.
  - Player: Interface for player behavior.
  - CleverPlayer, BasicPlayer, RandomPlayer, HumanPlayer: Implementations of the Player interface with unique strategies.
  - Rank and Suit: Represent the properties of cards, encapsulating card values and suits.

Sequence Diagram

The **sequence diagram** illustrates the flow of actions during a game round. It shows how the class interacts with players (both human and AI) to progress through each round. Key points in the sequence include:

- **Player Initialization**: Players are dynamically instantiated based on game settings.
- **Card Selection**: The sequence diagram details how each player selects a card to discard, with different strategies implemented for AI and human players.
- **Scoring**: After each player's turn, the game updates the score based on the cards played.

Note: Due to the dimensions of the diagram, I could not paste a clear image hence I have provided a link.

Link to - System Sequence Diagram

Extensibility and Future Development

The game's design is inherently extendable. Future enhancements could include:

- New Player Types: Introducing new player strategies (e.g., an AI player that learns from past games) would only require implementing the Player interface.
- **Advanced Game Rules**: New game modes or additional scoring rules could be added by modifying the score calculation methods without affecting the game's core logic.
- **Multiplayer Support**: The design could easily be adapted to support remote multiplayer games by introducing a networking layer, as the core game logic is independent of player input mechanisms.

Conclusion

The **HiFive** game is a well-architected system that adheres to the principles of modularity, extendability, and low coupling. By leveraging the Strategy and Factory patterns, the design ensures that player behavior can be easily modified or extended. The adherence to SOLID principles further strengthens the game's architecture, making it both maintainable and adaptable to future requirements. Through this careful design, the game remains flexible for further extensions and modifications.