**1. Introduction**

Project Overview:

The purpose of this project is to demonstrate the effective application of design patterns in software development through the implementation of two distinct games: Poker and Word Guess Game. Design patterns are essential in software engineering as they provide proven solutions to recurring design problems, promoting code reuse, maintainability, and scalability.

Objectives:

* Implement and showcase six design patterns: Singleton, Factory , Observer, Bridge, Decorator and Adapter patterns.
* Create comprehensive documentation explaining each feature and design pattern.
* Develop UML diagrams to illustrate the structure of the implemented design patterns.
* Build a user-friendly command-line interface (CLI) or graphical user interface (GUI) for interacting with the games.
* Demonstrate how design patterns enhance code clarity, maintainability, and extensibility.

**2. Project Requirements**

Comprehensive Documentation:

The Word Guess Game is a simple console-based game where the computer attempts to guess a target word, and the user can also make guesses. The computer's guessing logic is encapsulated in the ComputerGuesser class, and we've introduced design patterns such as Adapter and Decorator to enhance the functionality.

1. Adapter Pattern (Adapter Pattern):

The code contains the GuesserAdapter class, which adapts an object of the ComputerGuesser type to the Guesser interface. This allows you to use ComputerGuesser as a Guesser type object, which simplifies integration with other parts of the code that are waiting for objects that implement the Guesser interface.

1. Decorator Pattern (Decorator Pattern):

The code contains the GuessCounterDecorator class, which decorates an object of the Guesser type. The decorator adds functionality for tracking the number of guessing attempts without changing the basic behavior of the object implementing the Guesser interface. This allows you to dynamically add functionality without changing existing code.

1. Bridge Pattern (Bridge Pattern):

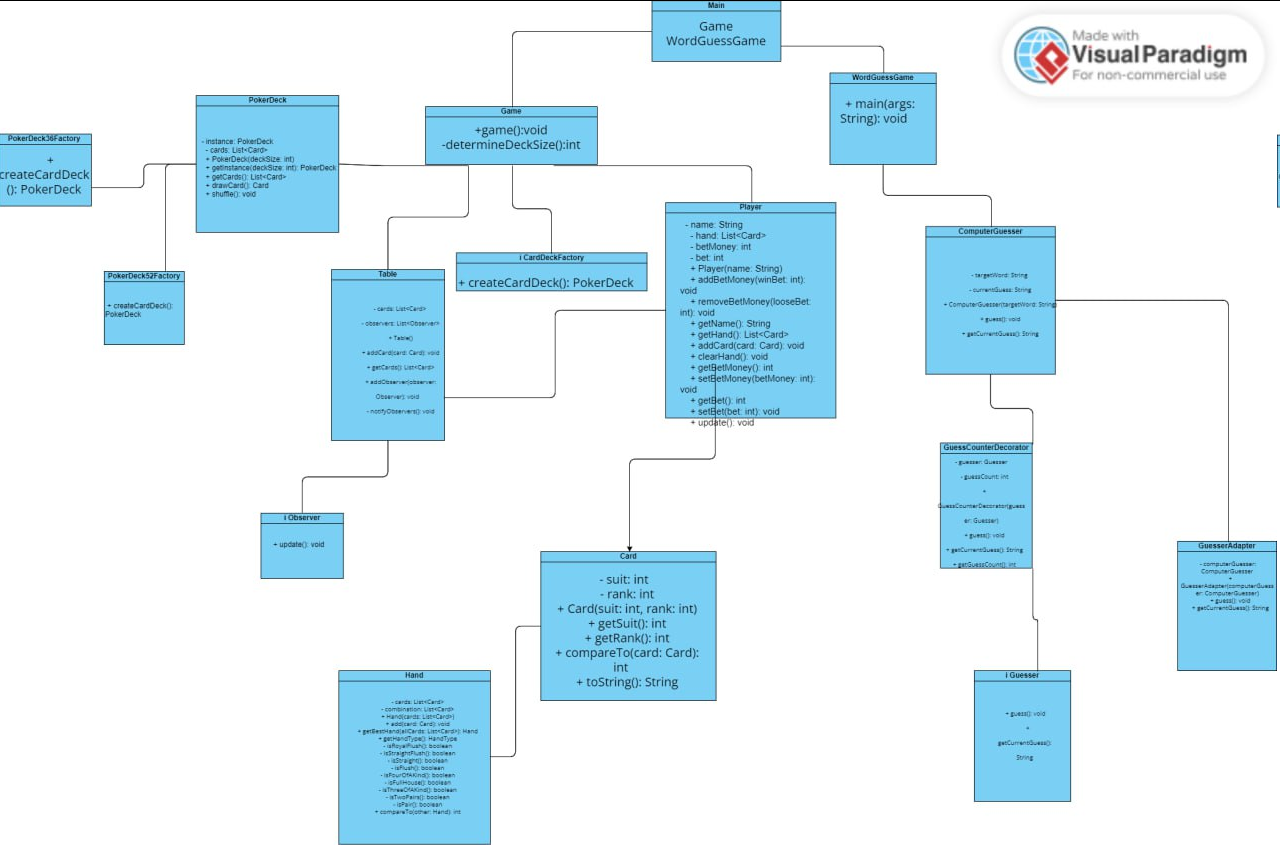
Although there is no explicit use of the "Bridge" pattern in the code, the GuesserAdapter scheme can also be interpreted as a simplified implementation of the "Bridge" pattern. The "bridge" allows you to separate abstraction from its implementation. In this case, the adapter separates the Guesser interface from a specific ComputerGuesser implementation, which makes it easier to replace or extend this implementation without changing the interface.

Poker patterns:

1. Observer - notifies you when a deck of cards changes. In our case 3 times - when the cards are dealt to the first and second player and when the main 5 cards are dealt.
2. Factory - Used for the possibility of choosing a 36 or 52 card playing deck.
3. Singleton - Used to initialize a single deck for the game.

The common use of these patterns in code allows for flexibility and extensibility, making it easier to add new features and change behavior without having to change existing code.

UML Diagrams:



User-Friendly Interface:

In poker, the user can choose the number of cards in the deck, as well as a convenient game design, in the letter guessing game, the user enters a word at the beginning that the computer must guess, then he sees a smooth animation of the selection of letters for the word

**4. Conclusion**

Summary:

In this project, a variety of design patterns were strategically employed to enhance the structure and functionality of two Java-based games: poker and word guessing. The following design patterns were utilized - factory, adapter, bridge, decorator, singleton, and observer.

The factory pattern was leveraged to create objects in a centralized manner, promoting a more modular and scalable codebase. This facilitated the efficient creation of diverse game elements.

The adapter pattern played a crucial role in ensuring compatibility between different interfaces, allowing components to work seamlessly together. This increased the flexibility and interoperability of the code.

By incorporating the bridge pattern, a clear separation between abstraction and implementation was achieved. This design choice simplified future modifications and expansions by minimizing the impact of changes on existing code.

The decorator pattern was implemented to dynamically add or alter behavior of objects, contributing to a more flexible and extensible design. This allowed for the incorporation of new features without modifying the existing codebase.

A singleton pattern was applied to restrict the instantiation of a class to a single object, ensuring a global point of access. This was particularly beneficial in managing resources and maintaining a consistent state across the games.

Finally, the observer pattern facilitated a robust event-handling mechanism. It enabled efficient communication between different components of the games, ensuring that changes in one part of the system were appropriately reflected elsewhere.

The impact of these design patterns on the codebase was significant. They collectively contributed to a more modular, maintainable, and extensible architecture. The use of these patterns not only enhanced the overall design quality but also simplified future development and modifications.

In terms of outcomes, the games, poker, and word guessing, demonstrated improved flexibility, scalability, and maintainability. The strategic application of design patterns resulted in a codebase that is not only robust but also poised for seamless expansion and adaptation to future requirements

Future Improvements

**Poker Game:**

1. **Multiplayer Support:** Introduce a multiplayer feature so that users can play poker against each other online. This could involve setting up a server-client architecture where players connect to a central server to play together.
2. **Improved AI:** Enhance the computer players' AI for a more challenging and dynamic gaming experience. You can explore different AI algorithms to make the opponents behave more realistically, adapting to the player's strategies.
3. **Variations of Poker:** Implement different variations of poker, such as Texas Hold'em, Omaha, or Seven Card Stud. This would add variety and depth to the game, appealing to a broader audience.
4. **Graphics and User Interface:** Enhance the game's visuals and user interface to make it more engaging. Consider using graphics libraries to create a more visually appealing representation of the cards and the overall game.
5. **Tournaments and Leaderboards:** Introduce tournament modes and leaderboards to encourage competition among players. This can add a competitive element to the game and keep players coming back.

**Word Guessing Game:**

1. **Categories and Levels:** Implement different categories for word guessing, such as movies, animals, or countries. Additionally, add difficulty levels so players can choose the complexity of the words they want to guess.
2. **User-Created Content:** Allow users to create their own word lists or categories, fostering community engagement. You can implement a system to review and share user-generated content, ensuring quality and appropriateness.
3. **Time Challenges:** Add a time-based mode where players must guess the word within a set time limit. This can add an element of urgency and excitement to the game.
4. **Hints and Power-Ups:** Introduce a hint system or power-ups that players can earn or purchase to make the game more forgiving or challenging, depending on their preferences.
5. **Multiplayer Mode:** Similar to the poker game, consider adding a multiplayer mode where players can compete to guess words against each other. This could be turn-based or real-time, depending on the nature of the game.

Problems encountered:

As we worked on the project, there were problems with the implementation of all the patterns in 1 work, there were also difficulties with the part of inventing the logic and essence of games. But we decided to delegate responsibilities and separate patterns and work to improve our overall efficiency. This helped us to create 2 Poker games and Word Guessing, with the subsequent implementation of patterns in this project

**5. References**

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
2. Refactoring Guru. (URL:<https://refactoring.guru/>)
3. Astana IT University, Software Design Pattern Lectures and Presentations, [2023].

**6. Submission and Deadline**

Timely submission of the project is crucial for the successful completion of this assignment. The deadline for project submission is [Final week]. Adhering to this deadline is of utmost importance to maintain the integrity of the academic/professional process and to ensure fair and consistent evaluation.

Late submissions may incur penalties as specified in the course guidelines. It is essential to be aware of and adhere to these guidelines to avoid any negative impact on grades or project evaluation.

If you anticipate challenges in meeting the deadline due to unforeseen circumstances, we encourage you to communicate proactively. Please use the designated channels for communication and request an extension in advance, if needed.

In professional settings, meeting deadlines is a critical aspect of maintaining a high standard of work. Developing this discipline during your academic journey will serve you well in your future career.

Thank you for your commitment to the timely submission of your project.