**Deliverable- 1**

**Git Repository URL:** <https://github.com/kau15021/Deliverable-1.git>

**1. Project Background:**

- Project Goals: The project's main purpose is to make a computerized version of the card game "War" using programming methods. The main goal is to develop a fun and interactive game while demonstrating coding skills and design patterns.

- Final Vision: The final vision is to have a complete and visually appealing version of the "War" card game that can be played by two people. The game should follow the rules of "War" and provide an easy-to-use interface.

**2. Game Description:**

- "War" is a classic card game that you can play with a regular deck of 52 cards. The goal of the game is to collect all the cards and become the winner.

The rules of the game:

1. The deck of cards is shuffled, and each player gets the same number of cards.

2. Both players reveal the top card from their deck at the same time.

3. The player with the higher-ranked card wins and takes both cards, adding them to the bottom of their deck.

4. If both players reveal cards with the same rank, a "war" happens:

- Each player puts three more cards facedown and one card face-up.

- The player with the higher-ranked face-up card wins the war and collects all the cards.

- If there's another tie during a war, the process repeats.

5. The game continues until one player has gathered all the cards and is declared the winner.

**3. Starting Base Code:**

- Programming Language: The initial code is written in Java.

- Patterns Observed: The code uses common programming techniques like inheritance and abstraction.

- Coding Conventions: The code follows standard Java coding conventions, such as using clear variable and method names and proper formatting.

- Provided Classes: The code includes several classes:

- `Card`: Represents a card in a card game. It has a method called `toString ()` that child classes will implement to show specific card details.

- `Game`: Serves as the base class for the game. It has methods for managing players and determining the winner. Child classes will implement the `play ()` and `declareWinner()` methods for the specific game.

- `GroupOfCards`: Represents a group of cards in a game. It provides methods for shuffling the cards and managing the group's size.

- `Player`: Represents a player in the game. It has a method called `play ()` that child classes will override with their own logic.

**DESCRIPTION-**

The main place where developers work together and keep track of changes for War Game projects is a Git repository. Depending on the decided schedule, each developer needs to submit their code either every day or by the end of the week. To keep things organized, the repository has separate folders for code, text files, and UML diagrams. UML diagrams have their own special folders to make it easier to see project plans visually. All the source code files, along with any related text files or documentation, are stored in the code folders.

**STANDARDS SET FOR CODING-**

The project will take the following standards into consideration-

**RULES FOR NAMING-** we are taking into the consideration of PascalCase naming convention For a class in a Java codebase. This means that the first letter of each word in the class name should be capitalized, without any spaces or underscores.However In Java code, CamelCase is widely used for variables and functions. The first letter of the identifier is lower cased in camelCase, and the first letter of each word that follows is capitalised.

**FORMATTING-** Use spaces are consistently used to have indent lines in your code. The purpose of the code and any complex logic is being described through the appropriate comments that provides the complex code and the type of methods used to execute the java class.

**BEST POSSIBLE COMMENTING PRACTICES-**

Use clear, succinct comments to make your code easier to read. Describe any unusual choices or presumptions that were made throughout development.

**PROGRAMMING BEST PRACTISES:**

Aim towards modular, reusable code. Create functions with a clear purpose. Encourage the use of object-oriented design patterns and principles where appropriate.

**Tools:**

IDE: NetBeans

Version Control: Git

UML Diagrams: Visual Paradigm

Testing Framework: Junit

**DESIGN CONSIDERATIONS-**

**Encapsulation:** The following code demonstrates some encapsulation by using private access modifiers for the instance variables name and password in the User class. It provides public getter and setter methods, respectively, for gaining access to and altering these variables. This encapsulates the internal state of the User object and allows controlled access to the User object's contents.

Improvements to encapsulation may be possible:

To further enhance encapsulation, it would be good to make the User class immutable by making the name and password variables final and removing the associated setter methods. This would ensure that once a User object is created, its status cannot be modified.

**DELEGATION:**

The current code does not explicitly show delegation.

Room for improvement in delegation:

One situation where delegation might be used is the UnoOnline class. The run() method is currently in charge of checking the user's password. It would be preferable to delegate this responsibility to a distinct PasswordValidator class, which may include the code necessary to validate passwords. This would promote the separation of concerns and improve modularity.

**FLEXIBILITY AND ACCESSIBILTY-**

The current code is unflexible and difficult to maintain for the following reasons:

Magic figures Depending on future demand, the array's fixed size of 100 users may be inadequate or excessive. To accommodate variable numbers of online players, a dynamic data structure like ArrayList would be the best choice.

**UNEASE TO HANDLE THE ERROR-**

The programming does not handle potential errors or exceptions that might occur when handling user input or checking passwords. A appropriate error management and exception handling method would improve the code's robustness and maintainability.

**Functionality restrictions**: The current code just addresses adding a new user to the list of prospective users. Additional features like user authentication, user deactivation, or permanent user data storage may be advantageous to it.

**SOME SUGGESTIONS TO IMPROVE THE CODE READABILITY AND USABILITY-**

**Use the appropriate data structures:** Rather than hardcoding certain sizes or data types, it is preferable to use data structures that can adapt dynamically to accommodate changing data sizes. Instead of using fixed-size arrays, which cannot be stretched or contracted as needed, think about using dynamic data structures like lists or vectors. This will ensure that your code can operate on a variety of input sizes without running into any limitations.

**IMPLEMENT ERROR MANAGEMENT PROCEDURES:**

Error handling is a crucial component of reliable software development. It is crucial to integrate error handling techniques so that mistakes or unexpected input won't cause the code to malfunction or give inaccurate results. This may entail catching and elegantly handling errors using try-catch blocks or other similar constructions. You may improve user experience by gracefully recovering from failures and directing users towards problem-solving by implementing error handling methods.

**ADD NEW FEATURES TO ENHANCE USER EXPERIENCE AND SATISFY FUTURE EXPECTATIONS:**

It's crucial to take future requirements into account as well as existing requirements. This might entail adding features or functionalities that enhance the user experience overall, such clear error messages, engaging user interfaces, or more functionality. By foreseeing future demands, you may make your codebase more adaptable and flexible and lessen the need for substantial rewrites in the future.

**ENCOURAGE ENCAPSULATION AND DELEGATE WORK:**

Encapsulation is the division of code into smaller, easier-to-manage pieces, each containing data and connected functions. Code that is properly encapsulated has a distinct separation of duties, which makes it simpler to comprehend and update. Each component has a clear purpose when duties are given to various modules or classes based on their unique capabilities, making the codebase easier to manage and maintain.

**TO ENSURE MAINTAINABILITY AND FLEXIBILITY, USE GOOD CODING PRACTISES:**

By using descriptive names, developing modular, reusable code, and sticking to coding standards, you may boost the flexibility and maintainability of your codebase.It is necessary for us to follow each and every guidelines when writing the code . We can look into consideration that everyone, including you, can better understand what an object, such as a variable, function, or class, if it has a clear and meaningful name. If you use descriptive names, your code will be simpler to comprehend and require fewer comments and it would be even more better if we break the whole code into smaller parts with each module having to be self reliant and have an responsibility to do their own task execution. This will help us to reuse the code and we can modify the code as per the future requirements .Likely, each class should have only one responsibility this makes the code easier to understand, test and modify the code.

We can use these suggestions to create a more trustworthy, flexible, and manageable architecture for your codebase. This will enhance user experience and make updating or altering your code to meet evolving requirements easier as per the future needs.

**War Card Game Implementation**

**Overview**

The design and implementation strategy for creating a digital War card game are laid out in this proposal. The project's goal is to develop a playable, interactive game that follows the guidelines and goals of the conventional War card game. The Java programming language will be used for the implementation, and the provided base code will be expanded and customised to meet the unique needs of the game.

**Project Scope**

The project team consists of the following members:

- [Jasmine Saini- Developer]

- [Harleen Kaur - Tester]

- [Priyanka Priyanka - Developer]

- [Arman Sharma- Tester]

The technical scope of the project includes:

- Developing a GUI (graphical user interface) to simplify user interactions.

- Executing the rules of the game, such as deck construction, card comparisons, battles, and wars.

- enabling two players to engage in simultaneous competition.

- Monitoring and showing each player's score as the game progresses.

- Selecting a winner when one player collects every card or when the game reaches its designated end time.

The completion of the project will be defined by:

- A GUI that is both practical and aesthetically pleasing and enables fluid gameplay.

- Correct application of the game's mechanics and rules.

- The choice to launch a new game or close the current one.

- Players' scores and game progress are displayed during play.

**High-Level Requirements**

The new system must include the following features:

- Players must register before the game begins.

- The sharing of game results, including victories and defeats.

- The score of each player is shown in real-time.

- A user interface that is simple and easy to utilise for fluid gameplay.

**Implementation Plan**

You may access the project's Git repository at <https://github.com/kau15021/Deliverable-1.git>. At the end of each day, developers will be required to check their code, assuring consistent updates and teamwork. Code, text files, and UML diagrams will all have their directories in the project's file structure.

The project will follow the following code guidelines:

- Consistent indentation and naming practices.

- For clarity, meaningful variable and method names.

- Use of comments in documentation appropriately.

- Effective use of the concepts of object-oriented programming.

The following equipment will be used by the development team:

- Integrated Development Environment (Name of IDE)

- Unit testing: [Name of the testing framework]

- Git for version control

**Design Considerations**

Some Object-Oriented (OO) ideas are demonstrated in the existing base code. Here are a few illustrations involving the base code:

1. Encapsulation:

Child classes must implement the abstract method "toString()" provided by the "Card" class. This captures the card-to-string conversion behaviour, which varies based on the particular card game implementation.

2. Delegation:

- The 'Game' class delegated to its subclasses the duty of implementing the game's behaviour. For example, the 'play()' method will be created in a child class to provide the War gaming mechanics.

3. Flexibility/Maintainability:

- The 'GroupOfCards' class offers a versatile framework for controlling a collection of cards in a game. By subclassing, it is possible to adapt and reuse this class to the needs of other card games.

To ensure a well-structured and maintainable codebase, the team will work to strengthen these principles during project execution while also adhering to other crucial design concerns.

**Conclusion**

The goals, parameters, specifications, and strategy for developing a digital version of the War card game are described in this proposal. Following this suggestion will provide the development team with a clear road map for making an entertaining and useful game that captures the spirit of the original card game. To create a top-notch gaming experience, the project will build upon the base code that has been provided.