Design Document Template For Final Project

**1. Project Background and Description**

The goal of the project is to implement a working version of the card game, War. This game is a two-player game. As such, the final implementation of the game will see the user play against the “computer.”

The rules of the game are as follows: the deck of 52 playing card is divided evenly between the two players. Each player turns up a card at the same time as the opponent. The player with the high card takes both cards and puts them face down on the bottom of his stack of cards. If the cards are of the same rank, each player plays two cards – one face up and one face down. The player with the highest cards will claim the piles. If still tied, the process is repeated until someone has a winning hand. The game ends when one player runs out of cards. For more information, please see <https://bicyclecards.com/how-to-play/war>.

The base code given provides the basic framework for the game. The base code contains four classes – Card.java, GroupOfCards.java, Game.java, and Player.java. The Card class models the cards used in the game. GroupsOfCards models the deck. Game serves as the game controller. Finally, Player models the player.

The game is to be implemented using the Java programming language.

**2. Project Scope**

Manraj Garg, Brian Silvera, and Paul Tracz will implement the game. All three members of the team will share equal responsibility, committing an equal amount of work. The game will interface with the user using the console as standard output. The project will be deemed complete once a fully working version of the game (one that follows all the rules) is successfully implemented.

High Level Requirements

The game will begin by asking the players name. Upon the user entering the name, the game will run the constructor in the player class and initialize the player object. Once complete, the next step requires the deck to be initialized. The constructor in the GroupOfCards class will created 52 card objects and store them in an array (in accordance with a standard deck of cards). Once the 52-card deck is initialized, it is to be shuffled and evenly divided between the players.

At this point, the game can begin. A card will be selected from the players deck and compared against the opponent’s card. The game will compare the values of the card. The player with the highest card value will win the turn and shall have both cards placed in their deck. In case of a draw, both players shall put down two additional cards. The game will compare the value of the second of two-card (the first is face down) and decide a winner. The process shall be repeated until a player wins the turn.

At the end of each turn, the user is to input a command that will instruct the game to begin the next turn.

The game should keep track of how many cards both players have and update it at the end of every turn. The game is to end when one player no longer has any cards in their deck. The game is to clearly indicate that the game is over and indicate which https://github.com/briansilvera/SYST-17796-Project.gitplayer has won.

**3. Impementation Plan**

A github repository has been created to ease the implementation of this project. The repository can be found with the following url; <https://github.com/briansilvera/SYST-17796-Project.git>. Github will allow all of the collaborators to work on the code remotely, make changes to the code and verify the accuracy and completeness of other’s work.

We expect to use the Netbeans IDE to write the code for the game. Netbeans will allow for an easier coding experience and its team functionality will allow for seamless use of the project’s GitHub repository.

**4. Design Consideration**

The game’s base code currently uses encapsulation in the GroupOfCards class. The class is responsible for the deck of the game. It initializes the cards used in the game and is also responsible for other game functions that relate to the cards (ex. show cards, shuffle, etc.). How the class does this is hidden from the Game class that controls the game and the class guarantees that the desired result will happen.

Once implemented, the Card class will also be an example of encapsulation. The card class will be responsible for creating the card objects that are to be stored in the deck of cards. The exact implementation will be hidden from the GroupOfCards class. The card class will simple guarantee that it will build the card object when called with the values passed to it.

The Game class will control the game and delegate the tasks to the other classes. The Game class will delegate the instantiation for the card deck to the GroupOfCards class, who will in turn delegate the creating of the card objects to the Card class. The Game class will also delegate any functions that involve the deck of cards to the GroupOfCards class (ex. shuffle, etc.).

The Game class will also delegate the instantiation for the user(s) to the player to the Player class. The player class will be responsible for the creation of the player object.

This code will be both highly flexible and maintainable. Once complete, it will not be difficult to add some code that will alter the user experience. Each class has a designated and specific role. The code will be highly cohesive, with each class having only variables and methods necessary to perform its designated task.

The code is loosely coupled, as the implementation of each class’ task remains hidden form the other classes. The other classes know about the code, but as long as the code does what it guarantees, the implementation is not important. For example, if more cards needed to be introduced into the deck, only the GroupOfCards class needs to be modified (an example of reusability and extensibility – signs of loose coupling).