**Design Doc of Trianta Ena**

1. **Object Design Specification:**

For each class, we listed 4 essential points of our design:

**1.** purpose of the class

**2.** benefit of this class to the existing design

**3.** benefit of this class, if any to a future game

**4.** changes be made to our original object model to implement this variation of the Card game.

* ***Bet:***

1. It records the bet of player in each round.
2. It clearly defines the bet of each player which is easy to calculate.
3. The bet can be further applied to other card games.
4. No change from the BlackJack version, since it can be commonly used in all games with a bet.

* ***TriantaEna (changed from BlackJack):***

1. It's the entrance of the whole program.
2. It helps set an entrance to start the game.
3. In the future, the class can be changed into other game version, which is easier to control.
4. Only changed the name of the class

* ***TriantaEnaPlayer (changed from BlackJackPlayer):***

1. To set the role of player with their attributes and behaviors in this game.
2. player is an essential element in this game, it contains some variables and some operations the player might take.
3. The player can be easily changed to adjust other games.
4. Some changes are as follows:
   1. Add a variable named “role”, so we can tell player from banker and change role easily.
   2. Delete some cues that different from BlackJack’s
   3. Add a function that let the player choose if they want to make bet
   4. Change the rule while end game, because banker also has a wallet and should settle accounts at the end of game.

* ***TriantaEnaRules (changed from BlackJackRules):***

1. It defines the rules of this game.
2. It helps check whether the player wins or loses.
3. For future, the rules can be updated.
4. Defines some new rules in Trianta Ena, such as new ways to check the winner and to check if the hand card is natural Trianta Ena.

* ***TriantaEnaTable (changed from BlackJackTable):***

1. It defines where the participants play the Black Jack game.
2. Table is the controller of the game, it calls API from some instances and gathers needed information and then sends them to other API from some instances to let the game goes smoothly. Moreover, the table supports multi-players’ game by entering a number bigger than 1 when asked the number of player at the start of this game.
3. For future, some for features may be added to Black Jack table.

* ***Card:***

1. The class defines cards with suit, number and value.
2. It's the basic element of the card game which is essential to the whole program.
3. The cards can be further applied to other card games.

* ***CardPlayer:***

1. It defines all participants in the game, and is the super class of BlackJackPlayer and BlackJackDealer.
2. It defines some variables and methods that all card players have in common.
3. For future, we can add more features to card player and this class can be extends by other card game’s roles.
4. Change the print card method which is different from BlackJack

* ***Config:***

1. It contains some configuration of the system.
2. It provides some constant or static variables in the program.
3. For future, other static variables can be added to this class.
4. Add some extra global variables

* ***HandCard:***

1. To define the cards that the player currently holds.
2. It maintains a List that allows player to split his handcard
3. For future, the handcard can support player to hold more than 2 hands of cards

* ***Name:***

1. To define a person’s name.
2. It includes the full name of a person and easy to print out.
3. For future, we can easily change the structure of names.
4. No change from the BlackJack version, since it can be commonly used in all card games.

* ***Person:***

1. To define a normal person with name.
2. It is the super class of CardPlayer.
3. For future, the more details such as date of birth and some common methods and some new roles can be added.
4. No change from the BlackJack version, since it can be commonly used in all card games.

* ***Rules:***

1. An interface of rules for the card game.
2. It defines one method ‘checkWin()’ for all specific rules related to some card games, this method is used to check the winner of the game.
3. For future, we can add much more required methods for all card games to this interface and it can be implemented by other games.
4. No change from the BlackJack version, since it can be commonly used in all card games.

* ***Shuffle:***

1. It's the machine which controls all cards and gives card randomly.
2. With the Shuffle class, the player and dealer can get card randomly, the Shuffle class controls all the cards.
3. For future, the Shuffle can support more than 1 set of cards, just call Shuffle(int n) is ok.

* ***Table:***

1. An interface of table for the card game.
2. It defines two methods ‘playGame()’ and ‘printResult()’ for all specific table relating with some card games. The method playGame() is the logic controller of the game and the method printResult() prints the result of that game.
3. For future, we can add much more required methods for all card games to this interface.

* ***Utils:***

1. It contains some common methods in this program.
2. It includes all common operations of the system, so that it could be easily managed in the correct way.
3. For future, the utility methods can be added here.

* ***Wallet:***

1. To save or change the money of players.
2. It defines some methods for the game to get or edit the money of players.
3. For future, we can define different wallets for multiple players.
4. No change from the BlackJack version, since it can be commonly used in all card games.
5. **Self -Assessment**

* *A review of the object model changes. Would your original implementation of BlackJack be different if you knew about this game in advance?*

Our original implementation of BlackJack should be kind of different if we knew about this game in advance. We would extract more attributes or behaviors from the original class, in order to separate BlackJack from the overall card games. In this way, when we are making changes to implement Trianta Ena, we can clearly identify the part we should change, or create new classes to extend the overall card games.

* *Would the changes made to implement Trianta Ena have made your original implementation of BlackJack easier and/or better?*

The changes made to implement Trianta Ena have made our original implementation of BlackJack little bit easier. In the Trianta Ena version, we extract more attributes and behaviors of card games from the classes. Therefore, when we use the current model on implementing BlackJack, the project could be better organized so that it is clear that which part is own to BlackJack and which is belong to every card game.

1. **Program running process**

* The program initials like this:

1. input how many players will play this game
2. input player's name
3. input player's money
4. choose whether the dealer is a real man or not
5. if the dealer is a real man, input the dealer's name

* Each round of the Trianta Ena goes like this:

1. for each player, check whether he has more than $10 (the least bet is $10)
2. player inputs the money he wants to bet (more than $10 and less than $100)
3. player keeps choosing from four actions (hit, stand, split, double up)
4. the system gives us the result of this round, and finishes current round
5. system asks the player whether he wants to play a new game
   * if yes, go step 1
   * if no, remove the player from current game and print his final balance
6. if there is more than 1 player in the game, go back to step 1

* At the end of the system: print the final money the player has

1. **UML**

Please refer to the next page.

