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:
   a) Add a variable named “role”, so we can tell player from banker and change role easily.
   b) Delete some cues that different from BlackJack’s
   c) Add a function that let the player choose if they want to make bet
   d) 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 Trianta Ena 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 Trianta Ena table.
4. Just a few changes for the logic of the game.

✧ ***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.
4. No change from the BlackJack version, since it can be commonly used in all card games.

✧ ***CardPlayer:***

1. It defines all participants in the game, and is the super class of BlackJackPlayer and BlackJackBanker.
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
4. Make this class much more general for hand card, remove some methods that aims to specific game.

✧ ***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 games.

✧ ***Shuffle:***

1. It's the machine which controls all cards and gives card randomly.
2. With the Shuffle class, the player and banker 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.
4. Make this class much more general for hand card, remove some methods that aims to specific game.

✧ ***TriantaEnaShuffle:***

1. It's has some methods giving card to bankers only relating with the Trianta Ena game.
2. With the TriantaEnaShuffle class, the banker can keep getting card randomly when all player stand or bust.
3. For future, the TriantaEnaShuffle can support more give card methods that only for Trianta Ena game.
4. This is a new class, for that the Shuffle class is much more general than it is in Black Jack.

✧ ***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.
4. Make this class much more general for hand card, remove some methods that aims to specific game.

✧ ***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.
4. Add some new methods relating with Trianta Ena.

✧ ***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.

2) 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 classes, 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 much 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 specified to BlackJack and which is belong to every card game.

3) Program running process

✧ The program initials like this:

1. input how many people will play this game (include banker)
2. input players' name
3. input players' money
4. input banker's money
5. choose a banker randomly

✧ Each round of the Trianta Ena goes like this:

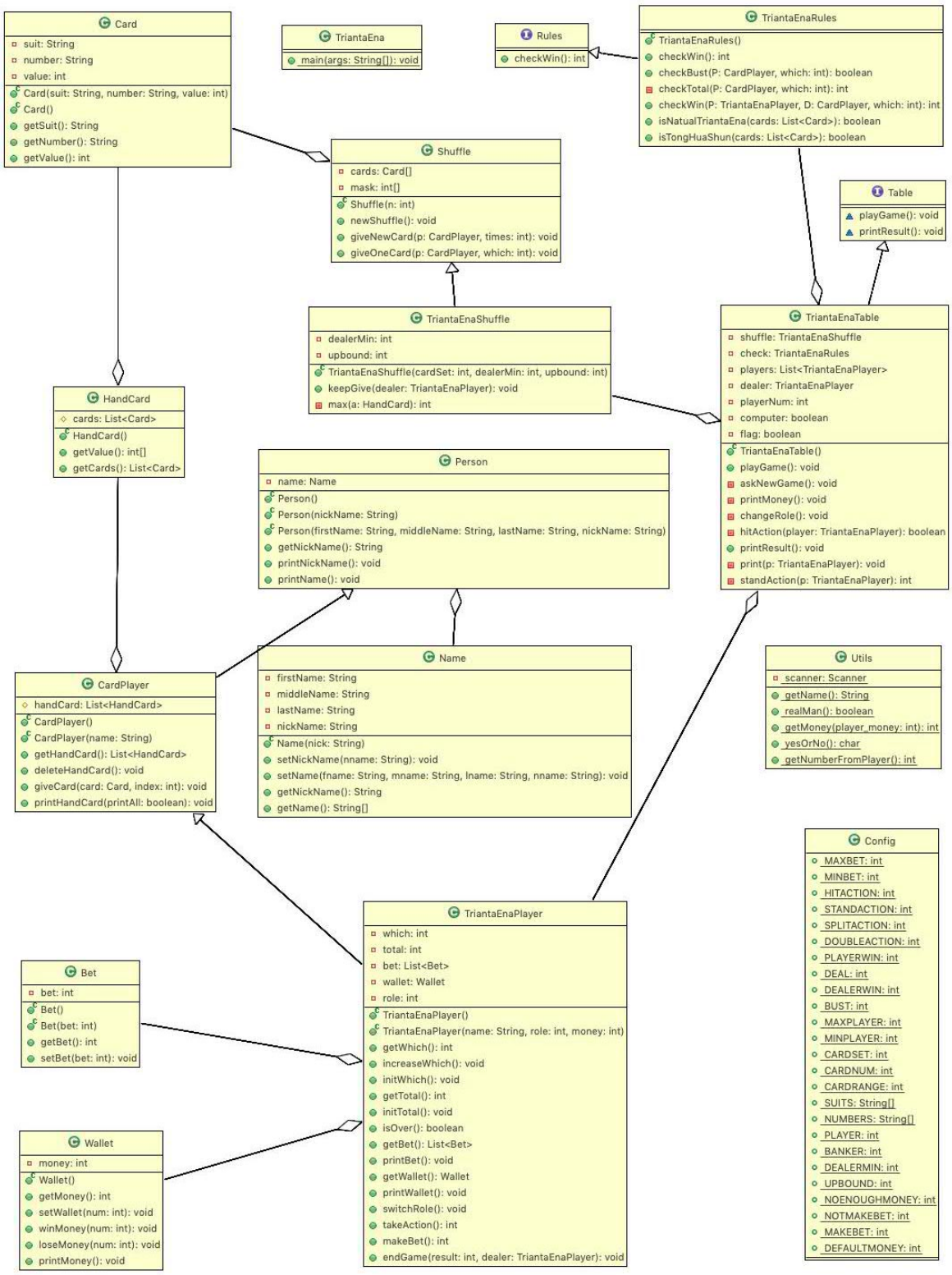
1. give each player one card
2. ask the player want to make bet or not
    - if no, go next player
    - if yes, move on
3. check whether he has more than $10 (the least bet is $10)
4. player inputs the money he wants to bet (more than $10 and less than $100)

After all players who want to make a bet make the bet:

5. player keeps choosing from two actions (hit, stand)
6. the system gives us the result of this round, and finishes current round
7. system asks the player whether he wants to play a new game
    - if yes, move on
    - if no, remove the player from current game and print his final balance
8. if there is less than 1 player in the game, end the game or move on
9. check if there are some players have more money than banker
    - if yes, ask those players who want to be banker, if nobody want to, then randomly choose one
    - if no, go to 1

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

4) **UML**


Card
suit: String
number: String
value: int
Card(suit: String, number: String, value: int)
Card()
getSuit(): String
getNumber(): String
getValue(): int
TriantaEna
main(args: String[]): void
Rules
checkWin(): int
TriantaEnaRules
TriantaEnaRules()
checkWin(): int
checkBust(P: CardPlayer, which: int): boolean
checkTotal(P: CardPlayer, which: int): int
checkWin(P: TriantaEnaPlayer, D: CardPlayer, which: int): int
isNatualTriantaEna(cards: List<Card>): boolean
isTongHuaShun(cards: List<Card>): boolean
Shuffle
cards: Card[]
mask: int[]
Shuffle(n: int)
newShuffle(): void
giveNewCard(p: CardPlayer, times: int): void
giveOneCard(p: CardPlayer, which: int): void
Table
playGame(): void
printResult(): void
TriantaEnaShuffle
dealerMin: int
upbound: int
TriantaEnaShuffle(cardSet: int, dealerMin: int, upbound: int)
keepGive(dealer: TriantaEnaPlayer): void
max(a: HandCard): int
TriantaEnaTable
shuffle: TriantaEnaShuffle
check: TriantaEnaRules
players: List<TriantaEnaPlayer>
dealer: TriantaEnaPlayer
playerNum: int
computer: boolean
flag: boolean
TriantaEnaTable()
playGame(): void
askNewGame(): void
printMoney(): void
changeRole(): void
hitAction(player: TriantaEnaPlayer): boolean
printResult(): void
print(p: TriantaEnaPlayer): void
standAction(p: TriantaEnaPlayer): int
HandCard
cards: List<Card>
HandCard()
getValue(): int[]
getCards(): List<Card>
Person
name: Name
Person()
Person(nickName: String)
Person(firstName: String, middleName: String, lastName: String, nickName: String)
getNickName(): String
printNickName(): void
printName(): void
Name
firstName: String
middleName: String
lastName: String
nickName: String
Name(nick: String)
setNickName(nname: String): void
setName(fname: String, mname: String, lname: String, nname: String): void
getNickName(): String
getName(): String[]
Utils
scanner: Scanner
getName(): String
realMan(): boolean
getMoney(player_money: int): int
yesOrNo(): char
getNumberFromPlayer(): int
CardPlayer
handCard: List<HandCard>
CardPlayer()
CardPlayer(name: String)
getHandCard(): List<HandCard>
deleteHandCard(): void
giveCard(card: Card, index: int): void
printHandCard(printAll: boolean): void
Config
MAXBET: int
MINBET: int
HITACTION: int
STANDACTION: int
SPLITACTION: int
DOUBLEACTION: int
PLAYERWIN: int
DEAL: int
DEALERWIN: int
BUST: int
MAXPLAYER: int
MINPLAYER: int
CARDSET: int
CARDNUM: int
CARDRANGE: int
SUITS: String[]
NUMBERS: String[]
PLAYER: int
BANKER: int
DEALERMIN: int
UPBOUND: int
NOENOUGHMONEY: int
NOTMAKEBET: int
MAKEBET: int
DEFAULTMONEY: int
TriantaEnaPlayer
which: int
total: int
bet: List<Bet>
wallet: Wallet
role: int
TriantaEnaPlayer()
TriantaEnaPlayer(name: String, role: int, money: int)
getWhich(): int
increaseWhich(): void
initWhich(): void
getTotal(): int
initTotal(): void
isOver(): boolean
getBet(): List<Bet>
printBet(): void
getWallet(): Wallet
printWallet(): void
switchRole(): void
takeAction(): int
makeBet(): int
endGame(result: int, dealer: TriantaEnaPlayer): void
Bet
bet: int
Bet()
Bet(bet: int)
getBet(): int
setBet(bet: int): void
Wallet
money: int
Wallet()
getMoney(): int
setWallet(num: int): void
winMoney(num: int): void
loseMoney(num: int): void
printMoney(): void