Python for Computer Science and Data Science 2 (CSE 3652)

MAJOR ASSIGNMENT-1: OBJECT-ORIENTED PROGRAMMING (OOP)

Card Game Simulation and Analysis

1 Problem Statement

Design and implement a basic card game framework that supports simple card games like Blackjack. The framework should include a deck of cards, players, and basic game rules. The goal of this project is to create an extendable and reusable structure that can accommodate the card game by implementing fundamental components such as a deck of cards, a hand, and a set of game rules.
For example, in Blackjack, the objective is to achieve a hand value as close to 21 as possible without exceeding it. The game involves a dealer and one or more players. The dealer follows fixed rules for drawing cards, while players can decide whether to draw additional cards.

2 Tasks

2.1 Task 1: Create Card and Deck Classes

• Create a Card class that represents a single playing card. Each card should have a **suit** (Hearts, Diamonds, Clubs, or Spades) and a **rank** (Ace, 2, 3, ..., 10, Jack, Queen, King).
• Use an enum type to define suits instead of using string values. This ensures consistency and prevents errors due to typos.
• Implement a Deck class that generates a full deck of 52 cards, consisting of four suits and thirteen ranks each.
• Add functionality to shuffle the deck randomly.
• Implement a method to draw a card from the deck, removing it from the available cards.
• **Example Data:**
– Example of a single card: Card(suit=Hearts, rank=Ace).
– Example of a shuffled deck: [Card(Spades, King), Card(Diamonds, 7), Card(Hearts, 2), ...].

2.2 Task 2: Implement a Simple Blackjack Game

• Blackjack is a game where players aim to get as close to 21 points as possible without exceeding it. Face cards (King, Queen, Jack) are worth 10 points, Aces can be worth 1 or 11 points, and other cards retain their numeric values.
• The dealer and the player are both dealt two cards at the start of the game.
• The player can choose to **hit** (draw an additional card) or **stand** (don't take any more cards.).
• If the player's total exceeds 21, they go **bust** and lose the game.
• The dealer must follow a set rule: if their total is 16 or below, they must hit; if 17 or above, they must stand.
• The player wins if their total is higher than the dealer's without exceeding 21.
• **Example Scenario:**
– **Player's Hand:** [Card(Hearts, 10), Card(Clubs, 7)]
Total: 17
– **Dealer's Hand:** [Card(Spades, 8), Card(Diamonds, 10)]
Total: 18
– **Result:** Dealer wins.

2.3 Task 3: Implement Multiplayer Support

• Extend the framework to support multiple players in a game.
• Allow each player to take turns making decisions (e.g., hit or stand in Blackjack).
• Implement a turn-based system to manage player actions and game progression.
• Display player hands and game status after each round.
• **Example Data:**
– Player 1 hand: [Card(Hearts, 7), Card(Spades, 5)].

– Player 2 hand: [Card(Diamonds, 9), Card(Clubs, 6)].
– Player 1 hits and receives Card(Hearts, 3).
– Updated hand: [Card(Hearts, 7), Card(Spades, 5), Card(Hearts, 3)] (To
tal = 15).

NB:

Basic Rules of the Game:

• The goal is to get as close to 21 as possible without going over.
• Face cards (King, Queen, Jack) = **10 points**.
• Aces (A) = **1 or 11 points**, depending on what benefits the player.
• If your total is higher than the dealer's without busting (going over 21), **you win**.
• If you go over 21, you automatically **lose (bust)**.
• The dealer must **hit if they have 16 or less** and **stand on 17 or more**.

Ans: -

```
import random
from enum import Enum


class Suit(Enum):
HEARTS = "Hearts"
DIAMONDS = "Diamonds"
CLUBS = "Clubs"
SPADES = "Spades"

class Card:
def __init__(self, suit, rank):
self.suit = suit
self.rank = rank

def __repr__(self):
return f"Card({self.suit.value}, {self.rank})"

def value(self):
if self.rank in ["Jack", "Queen", "King"]:
return 10
elif self.rank == "Ace":
return 11
else:
return int(self.rank)

class Deck:
def __init__(self):
ranks = ["Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen",
"King"]
self.cards = [Card(suit, rank) for suit in Suit for rank in ranks]
self.shuffle()
```

```
def shuffle(self):
random.shuffle(self.cards)

def draw(self):
return self.cards.pop() if self.cards else None

class Player:
def __init__(self, name):
self.name = name
self.hand = []

def draw_card(self, deck):
card = deck.draw()
if card:
self.hand.append(card)

def hand_value(self):
total = sum(card.value() for card in self.hand)
aces = sum(1 for card in self.hand if card.rank == "Ace")
while total > 21 and aces:
total -= 10
aces -= 1
return total

def is_busted(self):
return self.hand_value() > 21

class Dealer(Player):
def __init__(self):
super().__init__("Dealer")

def should_hit(self):
return self.hand_value() <= 16

class BlackjackGame:
def __init__(self, player_names):
self.deck = Deck()
self.players = [Player(name) for name in player_names]
self.dealer = Dealer()
self.start_game()

def start_game(self):
for _ in range(2):
for player in self.players:
player.draw_card(self.deck)
```

```
self.dealer.draw_card(self.deck)

self.play_game()

def play_game(self):
for player in self.players:
print(f"\n{player.name}'s turn:")
while True:
print(f"Hand: {player.hand}, Total: {player.hand_value()}")
if player.is_busted():
print(f"{player.name} busts!")
break
action = input(f"{player.name}, do you want to hit or stand? (h/s): ").lower()
if action == "h":
player.draw_card(self.deck)
else:
break
print("\nDealer's turn:")
while self.dealer.should_hit():
self.dealer.draw_card(self.deck)
print(f"Dealer's Hand: {self.dealer.hand}, Total: {self.dealer.hand_value()}")
self.determine_winners()

def determine_winners(self):
dealer_total = self.dealer.hand_value()
if self.dealer.is_busted():
print("\nDealer busts! All remaining players win!")
return
for player in self.players:
player_total = player.hand_value()
if player.is_busted():
print(f"{player.name} loses (busted).")
elif player_total > dealer_total:
print(f"{player.name} wins!")
elif player_total < dealer_total:
print(f"{player.name} loses.")
else:
print(f"{player.name} ties with the dealer (Push).")

player_names = input("Enter player names separated by commas: ").split(", ")
game = BlackjackGame(player_names)
```

Output: -

```
Enter player names separated by commas John, Alice

 John's turn:
Hand: [Card(Clubs, King), Card(Spades, 4)], Total: 14
 John, do you want to hit or stand? (h/s): h
Hand: [Card(Clubs, King), Card(Spades, 4), Card(Hearts, King)], Total: 24
 John busts!

Alice's turn:
Hand: [Card(Spades, 5), Card(Diamonds, 5)], Total: 10
Alice, do you want to hit or stand? (h/s): s

Dealer's turn:
Dealer's Hand: [Card(Clubs, Jack), Card(Hearts, 10)], Total: 20
 John loses (busted).
Alice loses.
```