Names: Christian Joseph Hernia
Eryl Joseph Aspera
Francis Nikolai Arceo
Antonio Gabriel Salmon

Project name: Baraha ni Kulot: Play Baccarat Now

Project description:

This project aims to simulate Baccarat's classic casino card game using java, particularly making use of the four pillars object oriented programming such as encapsulation, polymorphism, abstraction, and inheritance. It involves creating a deck of cards, dealing hands to the Player and Banker, calculating hand values, and determining the winner based on specific rules.

Key Functions of a Baccarat UI

Core Gameplay Functions:

- **Place Bets:**
  - Allow players to place bets on the Player, Banker, or Tie.
  - Provide input fields for bet amounts.
  - Visually indicate the current bet amounts on each option.
- **Deal Cards:**
  - Simulate the dealing of cards to the Player and Banker.
  - Visually display the cards dealt to each hand.
- **Draw Cards:**
  - Implement the draw rules for the Player and Banker based on their initial two-card hand values.
  - Visually animate the drawing of additional cards.
- **Determine Winner:**
  - Calculate the final hand values for the Player and Banker.
  - Determine the winning hand and display the result.
  - Visually highlight the winning hand.
- **Payout Winnings:**
  - Calculate and display the player's winnings or losses.
  - Update the player's balance accordingly.

Optional:

Data Persistence

- Save and load game statistics
- Save player stats, win/loss records between sessions, bankroll history

User Interface

Game History:

- Track past rounds and display a history of bets, outcomes, and winnings (csv files)
- Allow players to review their past performance.

User Profiles:

- Allow players to create accounts and save their preferences, betting history, and balance.
- Implement login and registration functionalities.

Requirements:

Main Menu

- New Game: Starts a new game with a fresh deck of cards.
- Load Profile: Loads an existing user profile
- Save profile: Saves a user profile
- Exit: Quits the game.

Game Screen

- **Player Information:**
  - Player's name
  - Current balance
  - Bet history
- **Betting Area:**
  - Player bet
  - Banker bet
  - Tie bet
- **Card Display:**
  - Player's hand (2-3 cards)
  - Banker's hand (2-3 cards)
- **Game Controls:**
  - Deal button (initiates a new round)
  - End round button
- **Game Outcome Display:**
  - Winner announcement (Player, Banker, Tie)
  - Win/loss amount
  - Updated player balance

Win/Loss History

- **Table of past rounds:**
    - Round number
    - Player bet
    - Banker bet
    - Tie bet
    - Winning hand
    - Net winnings/losses
- **Total winnings/losses**
- **Win/loss percentage**

OOP and possible classes in Baccarat

Core Classes:

Card:

Represents a single playing card.

Attributes: suit (e.g., Hearts, Diamonds, Clubs, Spades), rank (e.g., Ace, 2, 3, ..., King)

Methods: getSuit(), getRank(), toString()

Deck:

Represents a deck of cards.

Attributes: an array or list of Card objects

Methods: shuffle(), dealCard(), isEmpty()

Hand:

Represents a hand of cards held by a player or the banker.

Attributes: a list of Card objects

Methods: addCard(), getHandValue(), isNatural(), isPair(), toString()

Player:

Represents a player.

Attributes: name, balance

Methods: placeBet(), addWinnings(), deductLoss(), getBalance()

Game:

Manages the overall game flow.

Attributes: deck, player, banker, currentBet

Methods: dealInitialCards(), drawCards(), calculateHandValues(), determineWinner(), payOutWinnings(), startNewGame()

Additional Classes (for advanced features):

UserInterface:

Handles the user interface, including input and output.

Methods: displayMenu(), displayGameBoard(), promptForBet(), displayResult()

GameHistory:

Stores and retrieves game history.

Attributes: a list of past rounds

Methods: addRound(), getRoundHistory()

UserProfile:

Stores user-specific information.

Attributes: username, password, balance, win/loss history

Methods: saveProfile(), loadProfile()

Key Design Considerations:

Encapsulation: Keep the internal state of classes private and expose them through public methods.

Inheritance: Consider using inheritance for different types of players (e.g., human, computer) or different game variations.

Polymorphism: Use polymorphism to implement different strategies for drawing cards or calculating hand values.

Abstraction: Focus on the essential features of each class and hide unnecessary implementation details.