Code for the Online Blackjack Game

Write the object-oriented code to implement the design of the Blackjack game problem.

We'll cover the following

We've covered different aspects of the Blackjack game and observed the attributes attached to the problem using various UML diagrams. Let us now explore the more practical side of things where we will work on implementing the Blackjack game using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in the Blackjack game:

- Java
- C#
- Python
- C++
- JavaScript

Blackjack game classes

In this section, we will provide the skeleton code of the classes designed in the class diagram lesson.

> **Note:** For simplicity, we are not defining getter and setter functions. The reader can assume that all class attributes are private and accessed through their respective public getter methods and modified only through their public method functions.

Enumerations and custom data type

The following code provides the definition of the enumeration and custom data type used in the Blackjack game.

`Suit`: We need to create an enumeration to keep track of the suit of the card, whether it is diamond, spade, heart, or club.

`AccountStatus`: We need to create an enumeration to keep track of the status of the account, whether it is active, canceled, closed, blocked, or none.

The `Person` class is used as a custom data type. The implementation of the `Person` class can be found below:

> **Note:** JavaScript does not support enumerations, so we will be using the `Object.freeze()` method as an alternative that freezes an object and prevents further modifications.

```
// Enumeration
const Suit = Object.freeze({
  HEART,
  SPADE,
  CLUB,
  DIAMOND,
});

const AccountStatus = Object.freeze({
  ACTIVE,
  CLOSED,
  CANCELED,
  BLACKLISTED,
  BLOCKED
});

// Custom Person data type class
class Person {
    #name;
    #streetAddress;
    #city;
    #state;
    #zipCode
    #country;
    constructor(name, streetAddress, city, state,zipCode, country){
        this.#name = name;
        this.#streetAddress = streetAddress;
        this.#city = city;
        this.#state = state;
        this.#zipCode = zipCode;
```

Definition of enums and custom datatypes

Card

This class contains the playing cards or cards used in the Blackjack game.

```
public class Card {
  private Suit suit;
  private int faceValue;

  public Card(Suit suit, int faceValue);
}
```

The Card class

Deck and shoe

`Shoe` is a device to hold multiple `Deck` and a `Deck` has 52 cards of four suits. One suit contains nine number cards (2–10) and four face cards (King, Queen, Jack, and Ace).

```
class Deck {
  #cards;

  constructor(cards) {
    this.#cards = cards;
  }
  getCards();
}

class Shoe {
  #decks;
  #numberOfDecks;

  constructor(numberOfDecks, decks) {
    this.#decks = decks;
    this.#numberOfDecks = numberOfDecks;
  }
  createShoe();
  shuffle();
  dealCard();
}
```

The Deck and Shoe classes

Hand

The `Hand` class represents a Blackjack hand used in this game and contains multiple cards.

```
class Hand {
  #cards;

  constructor(cards) {
    this.#cards = cards;
  }

  getScores();
  addCard(Card card);
}
```

Players

The `Player` is an abstract class and the `BlackjackPlayer` and `Dealer` classes extend the `Player` class.

- `BlackjackPlayer`: They place the first wager and update the stake with winning and losing sums. They can choose between the hit and stand options.
- `Dealer`: They are primarily in charge of dealing cards and following the Blackjack rules.

```
class Player {
  #id;
  #password;
  #balance;
  #status;
  #person;
  #hand;

  constructor(id, password, balance, status, person, hand) {
    if (this.constructor == Vehicle) {
      throw new Error("Abstract classes can't be instantiated.");
    }
    else {
      this.#id = id;
      this.#password = password;
      this.#balance = balance;
      this.#status = status;
      this.#status = status;
      this.#person = person;
      this.#hand = hand;
    }
  }
  addHand(hand);
  removeHand(hand);
  resetPassword();
  addToHand(hand);
}

class BlackjackPlayer extends Player {
  #bet;
  #totalCash;
```

Player and its derived classes

Blackjack controller and game view

The `BlackjackController` class validates the actions (hit or stand) and responds accordingly. The `BlackjackGameView` class represents the game view.

```
class BlackjackController {
  validateAction();
}

class BlackjackGameView {
  playAction(action, hand);
}
```

Blackjack game

The `BlackjackGame` class represents how we can play this game or its basic sequence of play.

```
class BlackjackGame {
  #player;
  #dealer;
  #shoe;
  #MAX_NUM_OF_DECKS

  constructor(player, dealer, shoe) {
    this.#player = player;
    this.#dealer = dealer;
    this.#shoe = shoe;
    this.#MAX_NUM_OF_DECKS = 4;
  }

  playAction(action, hand);
  hit(hand);
  stand();
  start();
}
```

The BlackjackGame class

Wrapping up

We've explored the complete design of the Blackjack game in this chapter. We've looked at how a basic Blacljack game can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.

Back

Activity Diagram for th...

Next

Getting Ready: The M...

Mark as Completed