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
enum Suit {
  HEART,
  SPADE,
  CLUB,
  DIAMOND
}

enum AccountStatus {
  ACTIVE,
  CLOSED,
  CANCELED,
  BLACKLISTED,
  NONE
}

// Custom Person data type class
public class Person {
  private String name;
  private String streetAddress;
  private String city;
  private String state;
  private int zipCode;
  private String country;
}
```

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).

```
public class Deck {
  private List<Card> cards;

  public Deck();
  public List<Card> getCards();
}

public class Shoe {
  private List<Deck> decks;
  private int numberOfDecks;

  public Shoe(int numberOfDecks, List<Deck> decks) {
    // 1. createShoe();
    // 2. shuffle();
  }
  public void createShoe();
  public void shuffle();
  public Card dealCard();
}
```

The Deck and Shoe classes

Hand

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

```
public class Hand {
  private List<Card> cards;

  public Hand(Card card1, Card card2);
  public int getScores();
  public void addCard(Card card);
}
```

The Hand class

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.

```
public abstract class Player {
  private String id;
  private String password;
  private double balance;
  private AccountStatus status;
  private Person person;
  private Hand hand;

  public void addHand(Hand hand);
  public void removeHand(Hand hand);
  public abstract boolean resetPassword();
  public void addToHand(Hand hand);
}

public class BlackjackPlayer extends Player {
  private int bet;
  private int totalCash;

  public BlackjackPlayer(Hand hand);
  public void placeBet(int bet);
  public boolean resetPassword(){
    // definition
  }
}

public class Dealer extends Player {
  private Hand hand;

  public int getTotalScore();
  public boolean resetPassword(){
```

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.

```
public class BlackjackController {
  public boolean validateAction();
}

public class BlackjackGameView {
  public void playAction(String action, Hand hand);
}
```

The BlackjackController and BlackjackGameView classes

Blackjack game

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

```
public class BlackjackGame {
  private Player player;
  private Dealer dealer;
  private Shoe shoe;
  private final int MAX_NUM_OF_DECKS = 4;

  public BlackjackGame(BlackjackPlayer player, Dealer dealer);
  public void playAction(String action, Hand hand);
  public void hit(Hand hand);
  public void stand();
  public void 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 Blackjack game can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.

Back

Activity Diagram for the Online Blackjack Game

Complete

Next

Getting Ready: The Meeting Scheduler Problem