Code for the Movie Ticket Booking System

Write the object-oriented code to implement the design of the movie ticket booking problem.

We'll cover the following

We've gone over different aspects of the movie ticket booking system and observed the attributes attached to the problem using various UML diagrams. Let's explore the more practical side of things, where we will work on implementing the movie ticket booking system 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 movie ticket booking system:

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

Movie ticket booking system

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

The following code provides the definition of the various enumerations used in the movie ticket booking system:

> **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.

Java | C# | Python | C++ | JavaScript

```
// Enumerations
const PaymentStatus = Object.freeze({
  PENDING,
  CONFIRMED,
  DECLINED,
  REFUNDED
```

```
});

const BookingStatus = Object.freeze({
  PENDING,
  CONFIRMED,
  CANCELLED,
  DENIED,
  REFUNDED
});

const SeatStatus = Object.freeze({
  AVAILABLE,
  BOOKED,
  RESERVED
});
```

Enum definitions

Actors

This section contains the different people that will interact with our movie ticket systems, such as a `Customer`, `Admins`, and `TicketAgents`. All of these classes will inherit the properties of the `Person` class. The definition of these classes is given below:

Java | C# | Python | C++ | JavaScript

```
// Person is an abstract class
class Person {
  #name;
  #address;
  #email;
  #phone;

  constructor(name, address, phone, email) {
    if (this.constructor === Person) {
      throw new Error("Abstract classes can't be instantiated.");
    }
    else {
      this.#name = name;
      this.#address = address;
      this.#phone = phone;
      this.#email = email;
    }
  }
}

class Customer extends Person {
  constructor(name, address, phone, email) {
    this.#bookings = new Array(); // List of bookings
    super(name, address, phone, email)
  }

  // booking here refers to an instance of the Booking class
  createBooking(booking) {}
  updateBooking(booking) {}
  deleteBooking(booking) {}
}
```

Actors involved in the movie ticket booking system

Seat

The `Seat` will be an abstract class, which serves as a parent for three different types of seats: `Platinum`, `Gold`, and `Silver`. The definition of the `Seat` and its child classes is given below:

Java | C# | Python | C++ | JavaScript

```
// Seat is an abstract class
class Seat {
  #seatNo;
  #status;

  // Data members
  constructor(seatNo, status) {
    if (this.constructor === Seat) {
      throw new Error("Abstract classes can't be instantiated.");
    }
    else {
      this.#seatNo = seatNo;
      this.#status = status; // Refers to the SeatStatus enum
    }
  }
```

```
  // Member functions
  isAvailable() { }
  setSeat() { }
  setRate() { }
}

class Platinum extends Seat {
  #rate;

  constructor(seatNo, status, rate) {
    this.#rate = rate;
    super(seatNo, status);
  }
```

Seat and its derived classes

Movie, showtime, and movie ticket

Next, we will explore the `ShowTime`, `Movie`, and `MovieTicket` classes that provide the details of the movie to the customer. The definition of these classes is given below:

Java | C# | Python | C++ | JavaScript

```
class Movie {
  #title;
  #genre;
  #release_date;
  #language;
  #duration;

  // Data members
  constructor(title, genre, language, release_date, duration) {
    this.#title = title;
    this.#genre = genre;
    // release_date attribute represent date and time
    this.#release_date = release_date;
    this.#language = language;
    this.#duration = duration;
    this.#shows = []; // List of shows
  }
}

class ShowTime {
  #showId;
  #startTime;
  #date;
  #duration;

  // Data members
  constructor(showId, startTime, date, duration) {
    this.#showId = showId;
    // startTime and date attributes represent date and time
    this.#startTime = startTime;
```

The Movie, ShowTime, and MovieTicket classes

City, cinema, and hall

This section contains classes like `Hall`, `Cinema`, and `City` that make up the infrastructure of our movie ticket system. The definition of these classes is given below:

Java | C# | Python | C++ | JavaScript

```
class City {
  #name;
  #state;
  #zipCode;
  #cinemas;

  // Data members
  constructor(name, state, zipCode) {
    this.#name = name;
    this.#state = state;
    this.#zipCode = zipCode;
    this.#cinemas = new Array(); // List of cinemas
  }
}

class Cinema {
  #cinemaId;
  #city;

  // Data members
```

```
  constructor(cinemaId, city) {
    this.#cinemaId = cinemaId;
    this.#city = city;  // Refers to an instance of the City class
    this.#halls = new Array(); // List of halls
  }
}

class Hall {
  #hallID;

  // Data members
```

City, cinema and hall classes

Payment

The `Payment` class is another abstract class, with the `Cash` and `CreditCard` classes as its child. This takes the `PaymentStatus` enum to keep track of the payment status. The definition of this class is given below:

Java | C# | Python | C++ | JavaScript

```
// Payment is an abstract class
class Payment {
    #amount;
    #timestamp;
    #status;

    // Data members
    constructor(amount, timestamp, status) {
        if (this.constructor === Payment) {
            throw new Error("Abstract classes can't be instantiated.");
        }
        else {
            this.#amount = amount;
            this.#timestamp = timestamp;
            this.#status = status; // Refers to the PaymentStatus enum
        }
    }

    makePayment() { }
}
class Cash extends Payment{
    makePayment(){
        // functionality
    }
}
class CreditCard extends Payment{
    #nameOnCard;
    #cardNumber;
    #billingAddress;
    #code;
```

Payment class and its child classes

Notification

The `Notification` class is an abstract class that is responsible for sending notifications via email or phone/SMS after actions performed by either the admin and/or customer. Its definition is given below:

Java | C# | Python | C++ | JavaScript

```
// Notification is an abstract class
class Notification {
  #notificationId;
  #createdOn;
  #content;

  constructor(notificationId, createdOn, content) {
    if (this.constructor === Notification) {
      throw new Error("Abstract classes can't be instantiated.");
    }
    else {
      this.#notificationId = notificationId;
      this.#createdOn = createdOn;
      this.#content = content;
    }
  }

  // person here refers to an instance of the Person class
  sendNotification(person) {};
}

class EmailNotification extends Notification {
  // person here refers to an instance of the Person class
  sendNotification(person) {
    // functionality
  }
```

```
}

class PhoneNotification extends Notification {
  // person here refers to an instance of the Person class
```

Notification class and its child classes

Booking

The `Booking` class is the main class of our movie ticket booking system and will display the information relating to a particular customer's booking. The definition of this class is given below:

Java | C# | Python | C++ | JavaScript

```
class Booking {
  #bookingId;
  #amount;
  #totalSeats;
  #createdOn;
  #isBooked;
  #payment;
  #show;

  // Data members
  constructor(bookingId, amount, totalSeats, createdOn, status, payment, show) {
    this.#bookingId = bookingId;
    this.#amount = amount;
    this.#totalSeats = totalSeats;
    this.#createdOn = createdOn;
    this.#status = status; // BookingStatus enum

    // Instances of classes
    this.#payment = payment;
    this.#show = show;
    this.#tickets = new Array(); // List of movie tickets
    this.#seats = new Array(); // List of seats
  }
}
```

The Booking class

Search and catalog

The `Catalog` class contains the movie information and implements the `Search` interface class to enable the search functionality based on the given criteria (title, language, genre, and release date). The definition of these two classes is given below:

Java | C# | Python | C++ | JavaScript

```
class Search {
  searchMovieTitle(title) {} // Returns list of movie titles
  searchMovieLanguage(language) {} // Returns list of movie languages
  searchMovieGenre(genre) {} // Returns list of movie genres
  searchMovieReleaseDate(date) {} // Returns list of movie release dates
}

class Catalogue extends Search {
  #movieTitles;
  #movieLanguages;
  #movieGenres;
  #movieReleaseDates;

  constructor() {
    this.#movieTitles = new Map();
    this.#movieLanguages = new Map();
    this.#movieGenres = new Map();
    this.#movieReleaseDates = new Map();
  }

  // Returns list of movie titles
  searchMovieTitle(title) {
    // functionality
  }
  // Returns list of movie languages
  searchMovieLanguage(language){
    // functionality
  }
  // Returns list of movie genres
  searchMovieGenre(genre){
```

Wrapping up

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

Back

Activity Diagram for the Movie Ticket Bookin...

Next

Getting Ready: The Car Rental System

Mark as Completed