TP4 on Java Programming: Advanced Cinema Ticket Management System

1 Introduction

The objective of the fourth lab emphasizes:

- Abstract classes and interfaces for flexible design.
- Explicit use of method overloading and overriding.
- Exception handling for robust error management.[1]

The requirement of your report:

- Submit everything in a .zip file named: `JAVA_TP4_prenom_nom.zip`;
  - You can finish it in a group (≤ 2), but clearly state your contribution and which part you are responsible for. If you prefer to work alone, that's fine.
  - Each group submits one report. Please team up within the group that you belong to, such as a small group within group1. (for TP4)
  - This report is 15% of your final grade.
- **Important**: Write a report (necessary) and submit it with code before the end of the lab. Report should contains a pdf file with summary on what you have done and screenshot of codes. Zip of code should include multiple .java class files.

Deadline: The deadline of group1 is: 21:00:00, 02/12/2024; the deadline of group2 is: 21:00:00, 09/12/2024.

1.1 Class diagram

Please provide a class diagram for your design; you may refer to the example on page 62 of `https://www.qiongliu.info/assets/teaching/java/Java_lecture1.pdf` [2]

2 Class and Interface Design

2.1 Abstract Class: `Reservation`

- **Attributes:**
  - `reservationDate`: A `Date` type attribute representing the date of the reservation.
- **Abstract Methods:**
  - `calculatePrice()`: An abstract method that returns a `double` value representing the price of the reservation.
  - `reserve(Customer c)`: An abstract method for handling ticket reservations for a customer.
- **Concrete Methods:**
  - `getReservationDate()`: A concrete method that returns the reservation date as a `Date`.

[1] I do not have time to explain exception handling during last class, please refer to my slides page 44-51 `https://www.qiongliu.info/assets/teaching/java/Java_cm4.pdf`

[2] You can create the class diagram either at the beginning, during the design phase, to guide your project, or at the end as a summary of your work.

2.2 Interface: `Discountable`

- **Methods:**
  - `applyDiscount(double percentage)`: A method that applies a discount to a ticket or reservation. It returns no value (`void`).
  - `calculateFinalPrice(double basePrice)`: A method that calculates the final price of a ticket or reservation after applying the discount. It returns a `double`.

2.3 Class: `Ticket`

- **Inheritance:** This class extends `Reservation` and implements `Discountable`.
- **Attributes:**
  - `price: double` – Represents the price of the ticket.
  - `available: boolean` – Indicates whether the ticket is currently available.
- **Methods:**
  - `calculatePrice(): double` – Overrides the method in `Reservation` to return the price of the ticket.
  - `reserve(Customer c)` – Overrides the method in `Reservation` to handle ticket reservation for a specific customer.
  - `applyDiscount(double percentage)` – Overrides the method in `Discountable` to reduce the ticket price by a given percentage.
  - `applyDiscount(double percentage, double maxDiscount)` – An overloaded version of `applyDiscount`, which applies a discount but ensures it does not exceed the specified maximum discount.
  - `calculateFinalPrice(double basePrice): double` – Implements the method from `Discountable` to calculate the final price of a ticket or reservation after applying the discount. Returns a `double`.

2.4 Class: `SubscriptionTicket`

- **Inheritance:** This class extends `Ticket`.
- **Attributes:**
  - `series: int` – Represents the subscription series number for the ticket.
- **Methods:**
  - `reserve(Customer c)` – Overrides the method in `Ticket` to include additional logic specific to subscription-based reservations.
  - `calculatePrice(): double` – Overrides the method in `Ticket` to compute the ticket price based on the subscription series.

5. Class: `Customer`

- Attributes:
  - `name: String` – The name of the customer.
  - `phone: String` – The customer's phone number.
  - `membershipType: String` – The type of membership (e.g., `Regular`, `Premium`).

- Methods:
  - Overloaded `Customer(String name, String phone)` – Initializes a `Regular` customer.
  - Overloaded `Customer(String name, String phone, String membershipType)` – Initializes a customer with a specific membership type.

2.5 Class: `TicketManager`

- **Attributes:**
  - `reservations: Reservation[]` – An array to store up to 50 reservation objects.
  - `count: int` – Tracks the number of reservations currently in the system.
- **Methods:**
  - `addReservation(Reservation r)`:
    - Adds a reservation object to the system.
    - **Output:** If successful, the reservation is added, and the `count` is incremented. Throws an exception if the array is full.
  - `listAvailableReservations(): Reservation[]`:
    - Iterates through the `reservations` array and collects all reservations that are available.
    - **Output:** Returns an array of available reservations. If no reservations are available, returns an empty array.
  - `calculateTotalIncome(): double`:
    - Iterates through the `reservations` array, summing the prices of all reservations that have been sold.
    - **Output:** Returns the total income as a `double`. If no tickets have been sold, the total income is `0.0`.

2.6 Class: `MainTest`

- Data Initialization
  - Create a `TicketManager` with an array size of 10.
  - Add 6 `Ticket` objects and 4 `SubscriptionTicket` objects.
  - Randomly set 3 tickets as unavailable.
- Test Cases
  1. Add a 11st reservation and verify that `ArrayFullException` is thrown.
  2. Apply an invalid discount to a ticket and verify that `InvalidDiscountException` is thrown.
  3. Reserve an unavailable ticket and verify that `TicketNotAvailableException` is thrown.
  4. Calculate total income from all reservations.
  5. List all available reservations.

Several Tips:

- **Tip 1:** Method to Overloading and Overriding
  - Overloading Examples

    - `Customer(String name, String phone)` – Initializes a `Regular` customer.
    - `Customer(String name, String phone, String membershipType)` – Initializes a customer with a specific membership type.
    - `applyDiscount(double percentage)` – Applies a discount.
    - `applyDiscount(double percentage, double maxDiscount)` – Applies a discount but limits the maximum value.
  - Overriding Examples
    - `Ticket.calculatePrice()` – Returns the ticket price.
    - `Ticket.reserve(Customer c)` – Implements ticket reservation logic.
    - `SubscriptionTicket.reserve(Customer c)` – Adds additional checks for subscription tickets.
    - `SubscriptionTicket.calculatePrice()` – Calculates the subscription price based on series.

- **Tip 2: Exception Handling**
  - Custom Exceptions
    - `ArrayFullException` – Thrown when attempting to add more reservations than the array can hold.
    - `TicketNotAvailableException` – Thrown when attempting to reserve an unavailable ticket.
    - `InvalidDiscountException` – Thrown when a discount percentage is negative or exceeds 100.
  - Exception Usage Examples
    - `addReservation(Reservation r)`:
      - Throws `ArrayFullException` if the reservation array is full.
    - `applyDiscount(double percentage)`:
      - Throws `InvalidDiscountException` if `percentage` is invalid.