Summative Assessment: Airport Terminal Management System

Objective

Develop a **Java-based Airport Terminal Management System** to manage flights, passengers, and ticket reservations. The system will also include **different types of aircraft using inheritance**. Instead of serialization, reservations will be stored in a **CSV file** for better readability and manual editing.

This project will reinforce concepts including:

- **Object-Oriented Programming (OOP)**: Inheritance, Composition, Interfaces
- **File Handling**: Read and Write objects to a **CSV file**
- **Collections**: Using `ArrayList` and `HashMap`
- **Date and Currency Handling**: `LocalDate` and `BigDecimal`
- **Unit Testing**: Writing and executing JUnit test cases
- **GitHub:** The project will be stored in GitHub and features managed via issues

1. Flight and Passenger Management

You will design a system to store flight and passenger details, including reservations.

1.1 Passenger Class

- Must store the passenger's name and passport number.
- Should have appropriate constructors and getter methods.

1.2 Flight Class

- Must store flight number, departure date, ticket price, and an associated aircraft.
- Should have appropriate constructors and getter methods.

1.3 Reservation System

- Use a **HashMap** where:
  - **Key:** Flight number (String)
  - **Value:** List of passengers (`ArrayList<Passenger>`)
- Implement methods to:
  - Add a passenger to a flight.
  - Retrieve all passengers booked on a specific flight.

2. Aircraft Management (Using Inheritance)

The system should support **different types of aircraft**, each with unique attributes.

2.1 Base Class: Aircraft

- Attributes:
  - `String model`
  - `int capacity` (number of passengers)
  - `double fuelCapacity` (in liters)
- Constructor to initialize aircraft details.
- Getter methods for attributes.

2.2 Derived Classes (Extending Aircraft)

Create at least **two specific aircraft types** that inherit from `Aircraft`:

2.2.1 CommercialAircraft

- Represents passenger planes used for regular flights.
- Additional Attribute:
  - `String airlineName` (e.g., "American Airlines")

2.2.2 PrivateJet

- Represents small private jets used for luxury travel.
- Additional Attributes:
  - `boolean hasLuxuryService` (true if luxury service is available)
  - `int maxSpeed` (jet speed in km/h)

2.3 Flight Class (Updated)

- Each flight must be associated with an **Aircraft** object.
- The system should support flights with **both commercial aircraft and private jets**.

3. File Handling (CSV Format)

Instead of **serialization**, reservations will be stored in a **CSV file** (`reservations.csv`). Each entry in the file should follow this format:

```
Unset
flightNumber,departureDate,ticketPrice,passengerName,passportNumber,aircraftModel,aircraftType
AA101,2024-05-10,299.99,Alice Smith,P12345,Boeing 737,Commercial
BB202,2024-06-15,450.50,John Doe,P67890,Airbus A320,Commercial
PJ001,2024-07-20,5000.00,None,None,Gulfstream G650,PrivateJet
```

3.1 Writing Reservations to CSV

- Open the CSV file in append mode.
- Store flight details along with passenger information.
- Ensure each record is written as a new line.

3.2 Reading Reservations from CSV

- Read the file line by line.
- Extract flight details, passenger information, and aircraft type.
- Populate the reservations into a `HashMap` for quick access.
- If the aircraft type is **"Commercial"**, create a `CommercialAircraft` instance.
- If the aircraft type is **"PrivateJet"**, create a `PrivateJet` instance.

4. Unit Testing

Write **JUnit test cases** to validate:

1. Adding a reservation to the system.
2. Retrieving passengers for a specific flight.
3. Writing and reading reservations from the CSV file.
4. Creating different aircraft types and associating them with flights.

5. Bonus Task (Loyalty Program - Optional)

Implement a **Loyalty Program** using an **interface**.

5.1 Define the LoyaltyProgram Interface

- Should include a method for applying a discount on ticket prices.

5.2 Implement Loyalty Tiers

- **RegularPassenger**: Pays full ticket price.
- **VIPPassenger**: Gets a 20% discount on ticket prices.

5.3 Modify Reservation System

- Ensure that discounts are applied correctly when booking flights.

Submission Guidelines

1. **Code Structure**:
    - Apply **OOP principles** effectively.
    - Ensure code is **clean, well-documented, and follows Java naming conventions**.
2. **Testing & Execution**:
    - All **JUnit tests must pass**.
    - Provide a `main` method to demonstrate system functionality.
3. **Submission Format**:
    - Upload a **zipped folder** containing:
        - Java source files (`.java`)
        - CSV file (`reservations.csv`)
        - Test cases (`test` directory)
        - README file (`README.md`) with a brief explanation of the implementation.

Expected System Behavior

The **main method** should demonstrate:

1. Creating **Flight** and **Passenger** objects.
2. Adding passengers to a flight.
3. Associating flights with different **Aircraft types** (Commercial or Private Jet).
4. Saving reservations to a CSV file.
5. Loading reservations from the CSV file.
6. Running JUnit tests to validate functionality.

Resources

- **CSV File Handling in Java**: Java File IO Tutorial
- **JUnit Testing**: JUnit 5 Documentation
- **Java Inheritance**: Inheritance in Java
- **Java Collections Framework**: Collections Tutorial

GitHub Project Feature List – Airport Terminal Management System (Java)

This feature list outlines a **step-by-step development plan** for implementing the **Airport Terminal Management System** in Java. Each feature can be tracked as an **issue** or a **GitHub Project task**, ensuring a structured workflow.

Step 1: Project Setup

- **Initialize Java Project**: Create a new Java project with the required package structure.
- **Set Up GitHub Repository**: Push the initial project structure to GitHub.
- **Create `README.md`**: Include a project description and setup instructions.

Step 2: Core Data Models

- **Implement `Passenger` Class**
  - Attributes: `name`, `passportNumber`
  - Constructor, getters, and `toString()` method
- **Implement `Flight` Class**
  - Attributes: `flightNumber`, `departureDate` (`LocalDate`), `ticketPrice` (`BigDecimal`), `Aircraft`
  - Constructor, getters, and `toString()` method
- **Implement Base `Aircraft` Class**
  - Attributes: `model`, `capacity`, `fuelCapacity`
  - Constructor, getters, and `toString()` method
- **Implement `CommercialAircraft` Class (Extends `Aircraft`)**
  - Additional Attribute: `airlineName`
  - Constructor and override `toString()`
- **Implement `PrivateJet` Class (Extends `Aircraft`)**
  - Additional Attributes: `hasLuxuryService`, `maxSpeed`
  - Constructor and override `toString()`

Step 3: Reservation System

- **Implement `ReservationSystem` Class**
  - Stores reservations using `HashMap<String, ArrayList<Passenger>>`
  - Methods:
    - `addReservation(String flightNumber, Passenger passenger)`
    - `List<Passenger> getPassengersForFlight(String flightNumber)`

Step 4: CSV File Handling

- **Implement CSV Writing (`saveReservationsToCSV()`)**
  - Write reservations to `reservations.csv`
  - Format: `flightNumber,departureDate,ticketPrice,passengerName,passportNumber,aircraftModel,aircraftType`
- **Implement CSV Reading (`loadReservationsFromCSV()`)**
  - Read `reservations.csv` and populate `ReservationSystem`

Step 5: Unit Testing

- **Set Up JUnit for Testing**
- **Write Tests for `Passenger` and `Flight` Classes**
- **Write Tests for `ReservationSystem` (Adding & Retrieving Reservations)**
- **Write Tests for CSV Read/Write Methods**

Step 6: Final Integration & Testing

- **Implement `Main` Method to Demonstrate System**
- **Run & Validate Unit Tests**
- **Code Cleanup & Documentation Updates**

Step 7: Bonus - Loyalty Program (Optional)

- **Implement `LoyaltyProgram` Interface**
  - Method: `BigDecimal applyDiscount(BigDecimal ticketPrice)`
- **Implement `RegularPassenger` (No Discount)**
- **Implement `VIPPassenger` (20% Discount)**
- **Modify Reservation System to Apply Discounts**

Step 8: Deployment & Project Completion

- **Add Usage Instructions to `README.md`**
- **Push Final Code to GitHub**
- **Tag Release (v1.0) (Optional)**

How to Use This List

- Create **GitHub Issues** for each task.
- Use **GitHub Projects** to track progress. (Optionalis)
- Implement features **incrementally** to ensure stability.

Project Structure

Unset

```
AirportTerminalManagementSystem/
├── src/
│   ├── main/
│   │   └── com/
│   │       └── airport/
│   │           ├── data/       // Data Layer
│   │           │   └── CSVUtil.java
│   │           ├── domain/     // Domain Layer (Business Logic)
│   │           │   ├── model/
│   │           │   │   ├── Aircraft.java
│   │           │   │   ├── CommercialAircraft.java
│   │           │   │   ├── Passenger.java
│   │           │   │   ├── PrivateJet.java
│   │           │   │   └── Flight.java
│   │           │   ├── reservation/
│   │           │   │   └── ReservationSystem.java
│   │           │   └── loyalty/  // For Bonus
│   │           │   │   ├── LoyaltyProgram.java
│   │           │   │   ├── RegularPassenger.java
│   │           │   │   └── VIPPassenger.java
│   │           └── view/       // View/Presentation Layer
│   │               └── AirportTerminalApp.java // Main class (Console UI)
│   └── test/
│       └── com/
│           └── airport/
│               └── AirportTerminalTest.java
├── data/
│   └── reservations.csv
├── README.md
└── pom.xml  // If using Maven (recommended / optional)
```

Pseudocode

Okay, here's plain English pseudocode for each Java file in the layered architecture:

1. `src/main/java/com/airport/data/CSVUtil.java`

```
Unset
CSVUtil class:


  Method: loadReservationsFromCSV(filename)
    1. Open the CSV file.
    2. Create an empty HashMap to store reservations (key:
flight number, value: list of passengers).
    3. Read the file line by line.
    4. For each line:
      a. Split the line into parts (flight number, date,
price, passenger name, passport, aircraft model, aircraft
type).
      b. Create a Flight object using the flight number,
date, price, and aircraft details.
      c. Create a Passenger object using the passenger name
and passport.
      d. If the flight number already exists in the
HashMap, get the existing list of passengers and add the
new passenger to it.
```

```
      e. Otherwise, create a new list of passengers, add
the new passenger to it, and put the list in the HashMap
with the flight number as the key.
    5. Close the CSV file.
    6. Return the HashMap of reservations.

  Method: saveReservationsToCSV(filename, reservations)
    1. Open the CSV file in append mode.
    2. For each flight number in the reservations HashMap:
      a. Get the list of passengers for that flight number.
      b. For each passenger in the list:
        i. Get the flight details (date, price, aircraft
model, aircraft type).
        ii. Write a new line to the CSV file with the
flight number, date, price, passenger name, passport,
aircraft model, and aircraft type.
    3. Close the CSV file.
```

2. `src/main/java/com/airport/domain/model/Aircraft.java`

Unset
```
Aircraft class:


  Attributes: model, capacity, fuelCapacity


  Constructor: Aircraft(model, capacity, fuelCapacity)
    1. Set the model, capacity, and fuelCapacity
attributes.


  Getter methods for model, capacity, and fuelCapacity.
```

3. **`src/main/java/com/airport/domain/model/CommercialAircraft.java`**

Unset
```
CommercialAircraft class (extends Aircraft):

  Attributes: airlineName

  Constructor: CommercialAircraft(model, capacity,
fuelCapacity, airlineName)
    1. Call the Aircraft constructor with model, capacity,
and fuelCapacity.
    2. Set the airlineName attribute.

  Getter method for airlineName.
```

4. **`src/main/java/com/airport/domain/model/Passenger.java`**

```Unset
Passenger class:

  Attributes: name, passportNumber

  Constructor: Passenger(name, passportNumber)
    1. Set the name and passportNumber attributes.

  Getter methods for name and passportNumber.
```

5. **`src/main/java/com/airport/domain/model/PrivateJet.java`**

```Unset
PrivateJet class (extends Aircraft):

  Attributes: hasLuxuryService, maxSpeed

  Constructor: PrivateJet(model, capacity, fuelCapacity,
hasLuxuryService, maxSpeed)
    1. Call the Aircraft constructor with model, capacity,
and fuelCapacity.
    2. Set the hasLuxuryService and maxSpeed attributes.

  Getter methods for hasLuxuryService and maxSpeed.
```

6. `src/main/java/com/airport/domain/model/Flight.java`

Unset
```
Flight class:

  Attributes: flightNumber, departureDate, ticketPrice,
aircraft (Aircraft object)

  Constructor: Flight(flightNumber, departureDate,
ticketPrice, aircraft)
    1. Set the flightNumber, departureDate, ticketPrice,
and aircraft attributes.

  Getter methods for flightNumber, departureDate,
ticketPrice, and aircraft.
```

7. `src/main/java/com/airport/domain/reservation/ReservationSystem.java`

```Unset
ReservationSystem class:

  Attributes: reservations (HashMap: key = flight number, value = list of passengers)

  Method: addReservation(flightNumber, passenger)
    1. If the flightNumber exists in the reservations HashMap:
      a. Get the list of passengers for that flight.
      b. Add the new passenger to the list.
    2. Otherwise:
      a. Create a new list of passengers.
      b. Add the new passenger to the list.
      c. Put the list in the reservations HashMap with the flightNumber as the key.

  Method: getPassengersForFlight(flightNumber)
    1. Return the list of passengers associated with the given flightNumber from the reservations HashMap.  Return an empty list if the flight number isn't found.
```

8. **`src/main/java/com/airport/domain/loyalty/LoyaltyProgram.java`** **(Bonus)**

Unset
```
LoyaltyProgram interface:


  Method: applyDiscount(ticketPrice)  // Returns the
discounted price.
```

9. **`src/main/java/com/airport/domain/loyalty/RegularPassenger.java`** **(Bonus)**

Unset
```
RegularPassenger class (implements LoyaltyProgram):


  Method: applyDiscount(ticketPrice)
    1. Return the original ticketPrice (no discount).
```

10. `src/main/java/com/airport/domain/loyalty/VIPPassenger.java` (Bonus)

```
Unset
VIPPassenger class (implements LoyaltyProgram):


  Method: applyDiscount(ticketPrice)
    1. Calculate the 20% discount.
    2. Subtract the discount from the ticketPrice.
    3. Return the discounted price.
```

11. `src/main/java/com/airport/view/AirportTerminalApp.java`

```
Unset
AirportTerminalApp class:


  Main method:
    1. Create instances of Flight, Passenger, Aircraft
(Commercial or Private Jet).
    2. Create a ReservationSystem object.
    3. Add passengers to flights using the
ReservationSystem.
    4. Save reservations to the CSV file using CSVUtil.
    5. Load reservations from the CSV file using CSVUtil.
    6. Demonstrate retrieving passengers for a flight.
    7. (Bonus) Demonstrate loyalty program discounts.
```

12. `src/test/java/com/airport/AirportTerminalTest.java`

This file will contain JUnit tests. The pseudocode for each test method would look like this:

```
Unset
Test method: testAddReservation()
  1. Create Flight and Passenger objects.
  2. Create a ReservationSystem object.
  3. Call the addReservation method.
  4. Assert that the reservation was added correctly (e.g.,
check if the passenger is in the list for the flight).

Test method: testGetPassengersForFlight()
  1. Create Flight and Passenger objects.
  2. Create a ReservationSystem object.
  3. Add passengers to a flight.
  4. Call the getPassengersForFlight method.
  5. Assert that the correct list of passengers is
returned.

// Similar test methods for CSV read/write, aircraft
creation, etc.
```