**Design Patterns Used:**

**Observer Design Pattern:**

* The Observer pattern is used to notify passengers when any changes occur to their reserved flights, such as updates or cancellations.
* Without it, FlightService would have to explicitly handle notifying passengers whenever a flight is updated or deleted.
* This would lead to issues such as:
  + Code duplication
  + Reduced flexibility (e.g., changing notification methods like SMS or Email)
  + Harder maintenance (modifications to the notification system might require changes in multiple parts of the flight service)

**Factory Design Pattern:**

* The Factory Method pattern is used to create instances of concrete meal classes based on the passenger’s meal preference.
* This approach makes the system easily extendable, allowing new meal types to be added without modifying existing logic.
* Concrete meal implementations can be modified independently without affecting other parts of the system.

**Strategy Design Pattern:**

* The Strategy pattern is used to calculate ticket prices dynamically based on the chosen seat type.
* This keeps pricing logic flexible and extendable.
* Additional pricing strategies can be introduced to factor in elements such as:
  + Extra luggage
  + Priority boarding
  + Seasonal discounts

**State Design Pattern:**

* The State pattern is used to manage seat availability dynamically based on the type of seat booked.
* When a passenger reserves a seat, the system transitions to the appropriate state, decrementing the number of available seats in the corresponding category (e.g., economy or business class).

**Dependency Injection Pattern:**

* The Dependency Injection (DI) pattern is used in service classes to manage dependencies efficiently.
* Instead of service classes creating instances of required components (e.g., FlightRepository, PassengerRepository), they are injected through the constructor.
* Benefits:
  + Improves loose coupling
  + Enhances maintainability and testability
  + Increases system flexibility

**Service Layer Pattern:**

* In the application, the Controller layer handles incoming HTTP requests and delegates the business logic to a Service layer, which interacts with the Repository layer for data persistence.
* This separation promotes:
  + Code reusability
  + Easier extension of functionalities without affecting other layers

**Additional Concepts Used:**

**Generics:**

* Multiple lists and repository interfaces utilize generics.
* Example: FlightRepository extends JpaRepository<Flight, Long> where:
  + Flight represents the entity type
  + Long specifies the type of its primary key
* This ensures type safety by restricting operations to Flight objects only.

**Collections:**

* Collections are used throughout the application to store and manage data efficiently.
* They provide built-in methods for:
  + Adding, removing, and accessing data
  + Filtering specific notifications related to a passenger

**Lambda Expressions:**

* Lambda expressions enhance code readability and conciseness.
* Used to streamline operations such as:
  + Iterating over collections
  + Defining behavior for specific actions