

Question 5

Team Number: 21

* + Aaditya Gupta (2022A1PS1446G)
  + Arjit Kumar Singh (2022A4PS0930G)
  + Shivan Gupta (2022B5AA1017G)
  + Vedant Chatterjee (2022B5A30943G)

Question 5:-

You are tasked with managing the operations of an airport. The airport consists of various types of terminals, including International Flight Terminal and Domestic Flight Terminal.

Within these terminals, there are different types of planes available, such as passenger planes and cargo planes. These planes can further be classified into standard and premium versions based on the type of goods they transport. Passenger flights are equipped with crew members, including pilots, co-pilots, and in-flight attendants. Additionally, you would also need to manage the passengers for the plane. Passengers should have the ability to perform online booking, cancellation, and postponement of both international and domestic flights. They should also be able to book their meals according to their preferences. These basic minimum features are essential, but to establish a comprehensive system, additional functionalities may need to be incorporated. The entire system has to be designed using OOP concepts:

Minimum 6 classes are required to accommodate all the requirements specified in the design problem. Additionally, it should include the following:

(I) Overloaded methods (minimum 2)

(II) Overloaded constructors (minimum 2)

(III) Vararg overloading (minimum 2)

(IV) Nested classes (static or non-static, atleast 1, this is a part of I above)

(V) Abstract class (minimum 1) (VI) Interface (minimum 1, it can be nested interface or single level or multiple inheritance) (VII) Hierarchical Inheritance (atleast 1)

(VII) Multiple Inheritance (atleast 1, this should be in addition to VI above)

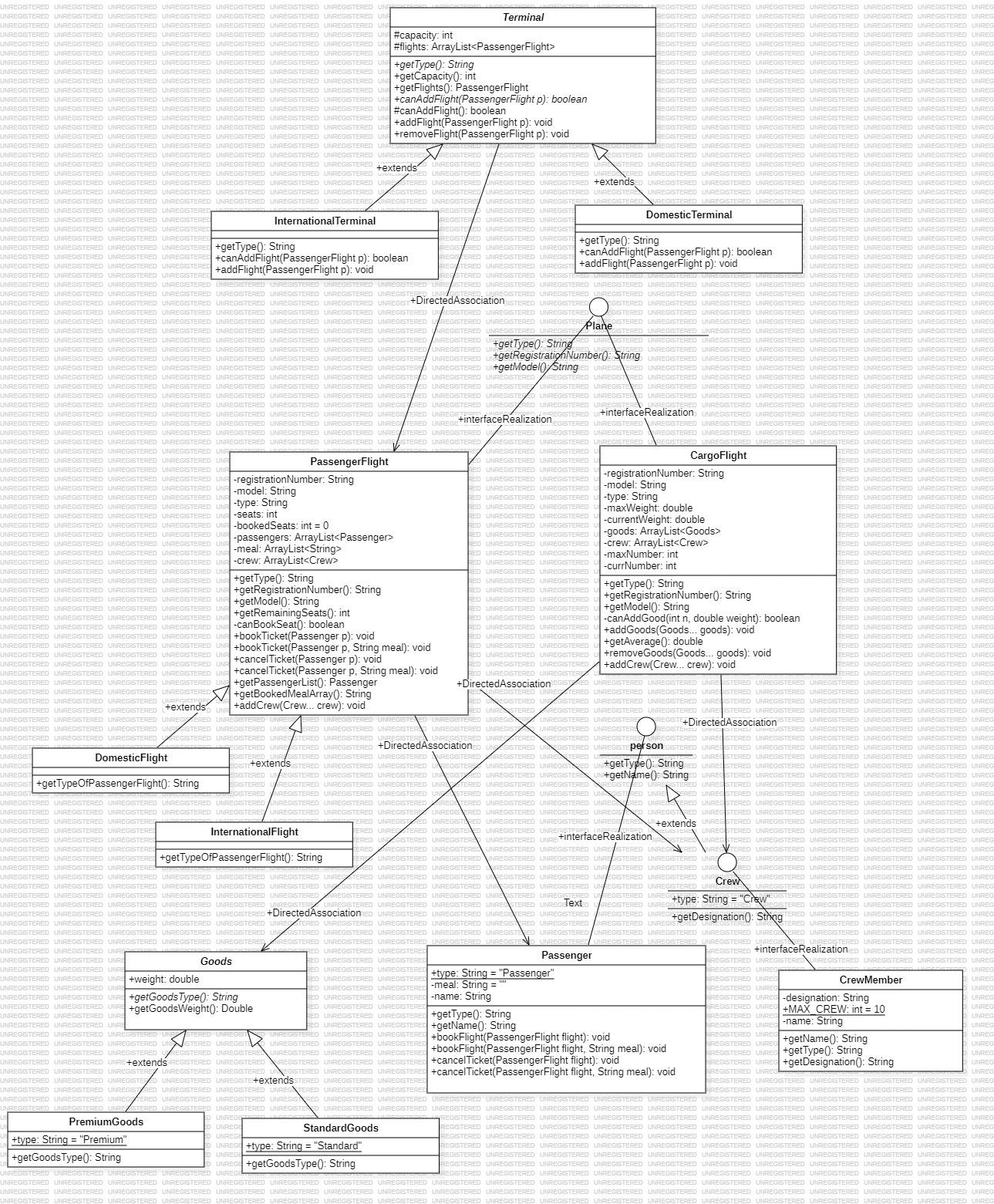
(VIII) Wrappers

(IX) Package

(X) Exception handling (atleast two cases) (XI) I/O: File Handling, scanner class etc. (atleast one from each of these)

(XII) Multithreading (by either Implementing the Runnable interface or extending the thread class)

UML Diagram



Airport

The `Airport` class simulates the functionality of an airport, including terminals and flights. It facilitates the management of flights, passengers, and terminals within the airport environment.

The class contains the following instance variables:

- `d1`: Represents a terminal for domestic flights.

- `i1`: Represents a terminal for international flights.

The class constructor (`Airport`) initializes the domestic and international terminals (`d1` and `i1`, respectively) with a fixed capacity of 10 flights each.

The `populateTerminal` method populates the domestic and international terminals with flights and assigns passengers to these flights. It creates instances of various types of flights (both domestic and international) and passengers, then books passengers onto the flights. Finally, it adds the flights to their respective terminals.

The `displayTerminalFlights` method displays the flights currently assigned to a terminal. It iterates through the flights in the terminal and prints their registration numbers.

The `displayPassengers` method displays the passengers currently booked on a specific flight. It iterates through the passenger list of the flight and prints their names.

The `main` method serves as the entry point for the airport simulation. It creates an instance of the `Airport` class and calls the `populateTerminal` method to set up the initial state of the airport.

Overall, the `Airport` class encapsulates the functionality required to manage flights, passengers, and terminals within the airport, allowing for simulation and testing of airport operations.

CargoFlight

The `CargoFlight` class represents a cargo flight, which is a type of aircraft used primarily for transporting goods. This class implements the `plane` interface, indicating that it adheres to certain standards and behaviors expected of aircraft.

The class has several instance variables:

- `registrationNumber`: Stores the unique registration number of the cargo flight.

- `model`: Represents the model of the cargo flight.

- `type`: Specifies the type of flight (always set to "Passenger" in this implementation).

- `maxWeight`: Indicates the maximum weight capacity of the cargo flight.

- `currentWeight`: Tracks the current total weight of goods loaded onto the cargo flight.

- `goods`: An ArrayList storing the goods currently loaded onto the flight.

- `crew`: An ArrayList holding the crew members assigned to the cargo flight.

- `maxNumber`: Defines the maximum number of goods allowed on the cargo flight.

- `currNumber`: Tracks the current number of goods loaded onto the cargo flight.

The class constructor (`CargoFlight`) initializes the cargo flight object with the provided registration number, model, maximum number of goods allowed, and maximum weight capacity.

The `addGoods` method allows for the addition of goods to the cargo flight. It checks whether the flight can accommodate the specified goods based on their number and total weight. If the goods can be added, they are appended to the goods list, and the current weight and number of goods are updated accordingly.

The `getAverage` method calculates the average weight of the goods currently loaded onto the cargo flight. It returns this value, handling any potential division by zero errors.

The `removeGoods` method enables the removal of specific goods from the cargo flight. It iterates through the goods list to find the specified goods and removes them if found. The current weight and number of goods are adjusted accordingly.

The `addCrew` method allows crew members to be added to the cargo flight. It appends the provided crew members to the crew list.

These methods collectively facilitate the management of goods and crew on the cargo flight, ensuring that it operates within its specified capacity and adheres to safety standards.

Crew

The `Crew` interface represents personnel who work aboard the aircraft, such as pilots, flight attendants, and engineers. This interface extends the `person` interface, implying that crew members share certain characteristics and behaviors with individuals in general.

The interface declares the following attributes and method:

- `type`: A constant string indicating the type of personnel, set to "Crew". This attribute is implicitly public, static, and final.

- `getDesignation()`: An abstract method to be implemented by classes that implement the `Crew` interface. It returns a string representing the specific designation or role of the crew member, such as "Pilot" or "Flight Attendant".

Overall, the `Crew` interface provides a blueprint for classes representing various roles within the aircraft's crew. It ensures consistency in behavior and attributes among different types of crew members.

# Crew Member

The `CrewMember` class represents an individual crew member working aboard the aircraft. Each `CrewMember` object has a specific designation or role within the crew, such as pilot, co-pilot, or flight attendant.

The class has the following instance variables:

- `designation`: Stores the designation or role of the crew member.

- `name`: Represents the name of the crew member.

- `MAX\_CREW`: A constant integer defining the maximum number of crew members allowed (set to 10 in this implementation).

The class constructor (`CrewMember`) initializes a `CrewMember` object with the provided name and designation.

The class provides the following methods:

- `getName()`: Returns the name of the crew member.

- `getType()`: Implements the `getType()` method from the `Crew` interface, returning the type of personnel (which is "Crew" in this case).

- `getDesignation()`: Implements the `getDesignation()` method from the `Crew` interface, returning the specific designation or role of the crew member.

Overall, the `CrewMember` class encapsulates the characteristics and behaviors of individual crew members, facilitating their management and integration into the aircraft's crew structure.

DomesticFlight

The `DomesticFlight` class represents a type of passenger flight that operates within a country's domestic airspace. This class is a subclass of the `PassengerFlight` class, inheriting its attributes and behaviors related to passenger flights.

The class constructor (`DomesticFlight`) initializes a `DomesticFlight` object with the provided registration number, model, and the number of seats, using the constructor of the superclass (`PassengerFlight`) to handle the initialization of these attributes.

The class provides the following method:

- `getTypeOfPassengerFlight()`: Returns a string indicating the type of passenger flight, which is "Domestic" in this case. This method is specific to `DomesticFlight` and does not exist in the superclass.

By extending the `PassengerFlight` class, the `DomesticFlight` class inherits functionality related to passenger flights while also providing specific information about its type. This allows for better organization and management of flights within the application, distinguishing between domestic and international flights.

DomesticTerminal

The `DomesticTerminal` class represents a terminal specifically designated for handling domestic flights within an airport. It is a subclass of the `Terminal` class, inheriting its attributes and functionalities related to terminal management.

The class provides the following constructors:

- `DomesticTerminal(int capacity)`: Initializes a `DomesticTerminal` object with the specified capacity (maximum number of flights).

- `DomesticTerminal(int capacity, PassengerFlight[] flights)`: Initializes a `DomesticTerminal` object with the specified capacity and an array of initial flights.

The `getType` method returns a string indicating the type of terminal, which is "Domestic" in this case.

The `canAddFlight` method determines whether a particular flight can be added to the terminal. It checks if the terminal has available capacity and if the flight is an instance of `DomesticFlight`. If both conditions are met, it returns true; otherwise, it returns false.

The `addFlight` method adds a given `PassengerFlight` object to the terminal if it meets the criteria specified by the `canAddFlight` method. If the flight can be added, it is appended to the list of flights managed by the terminal.

Overall, the `DomesticTerminal` class provides specialized functionality for managing domestic flights within the airport terminal infrastructure. It ensures that only domestic flights are accommodated in the terminal, maintaining organization and efficiency in flight operations.

Goods

The `Goods` class serves as an abstract representation of goods or cargo items that can be transported on a flight. It defines common attributes and behaviors for different types of goods.

The class is declared as abstract, indicating that it cannot be instantiated directly but serves as a template for subclasses to implement specific types of goods.

The class has the following instance variable:

- `weight`: Represents the weight of the goods.

The class constructor (`Goods`) initializes a `Goods` object with the specified weight.

The class provides the following methods:

- `getGoodsType()`: An abstract method that subclasses must implement to specify the type or category of goods.

- `getGoodsWeight()`: Returns the weight of the goods.

Overall, the `Goods` class lays the foundation for modeling various types of cargo items that can be transported on a flight. Subclasses can extend this class to define specific types of goods, each with its own characteristics and behaviors.

InternationalFlight

The `InternationalFlight` class represents a type of passenger flight that operates between countries, typically crossing international borders. This class is a subclass of the `PassengerFlight` class, inheriting its attributes and behaviors related to passenger flights.

The class constructor (`InternationalFlight`) initializes an `InternationalFlight` object with the provided registration number, model, and the number of seats, using the constructor of the superclass (`PassengerFlight`) to handle the initialization of these attributes.

The class provides the following method:

- `getTypeOfPassengerFlight()`: Returns a string indicating the type of passenger flight, which is "International" in this case. This method is specific to `InternationalFlight` and does not exist in the superclass.

By extending the `PassengerFlight` class, the `InternationalFlight` class inherits functionality related to passenger flights while also providing specific information about its type. This allows for better organization and management of flights within the application, distinguishing between domestic and international flights.

InternationalTerminal

The `InternationalTerminal` class represents a terminal specifically designated for handling international flights within an airport. It is a subclass of the `Terminal` class, inheriting its attributes and functionalities related to terminal management.

The class provides the following constructors:

- `InternationalTerminal(int capacity)`: Initializes an `InternationalTerminal` object with the specified capacity (maximum number of flights).

- `InternationalTerminal(int capacity, PassengerFlight[] flights)`: Initializes an `InternationalTerminal` object with the specified capacity and an array of initial flights.

The `getType` method returns a string indicating the type of terminal, which is "International" in this case.

The `canAddFlight` method determines whether a particular flight can be added to the terminal. It checks if the terminal has available capacity and if the flight is an instance of `InternationalFlight`. If both conditions are met, it returns true; otherwise, it returns false.

The `addFlight` method adds a given `PassengerFlight` object to the terminal if it meets the criteria specified by the `canAddFlight` method. If the flight can be added, it is appended to the list of flights managed by the terminal.

Overall, the `InternationalTerminal` class provides specialized functionality for managing international flights within the airport terminal infrastructure. It ensures that only international flights are accommodated in the terminal, maintaining organization and efficiency in flight operations.

Passenger

The `Passenger` class represents an individual passenger who intends to travel on a flight. Passengers have names and may optionally request a meal preference for their journey.

The class provides the following constructors:

- `Passenger(String name)`: Initializes a `Passenger` object with the specified name.

- `Passenger(String name, String meal)`: Initializes a `Passenger` object with the specified name and meal preference.

The class has the following instance variables:

- `meal`: Represents the meal preference requested by the passenger (if any).

- `name`: Stores the name of the passenger.

The class provides the following methods:

- `getType()`: Returns a string indicating the type of entity, which is "Passenger" in this case.

- `getName()`: Returns the name of the passenger.

- `bookFlight(PassengerFlight flight)`: Books a ticket for the passenger on the specified `PassengerFlight`. If the passenger has specified a meal preference, it is included in the booking.

- `bookFlight(PassengerFlight flight, String meal)`: Books a ticket for the passenger on the specified `PassengerFlight` with the provided meal preference.

- `cancelTicket(PassengerFlight flight)`: Cancels the ticket booked for the passenger on the specified `PassengerFlight`. If the passenger had a meal preference, it is included in the cancellation.

- `cancelTicket(PassengerFlight flight, String meal)`: Cancels the ticket booked for the passenger on the specified `PassengerFlight` with the provided meal preference.

Overall, the `Passenger` class encapsulates the characteristics and behaviors of individuals traveling on flights. It allows passengers to book and cancel tickets, optionally specifying meal preferences for their journey.

PassengerFlight

The `PassengerFlight` class represents a type of flight designed to carry passengers. It implements the `plane` interface, indicating that it adheres to certain standards and behaviors expected of aircraft.

The class has several instance variables:

- `registrationNumber`: Stores the unique registration number of the passenger flight.

- `model`: Represents the model of the passenger flight.

- `type`: Specifies the type of flight (always set to "Passenger" in this implementation).

- `seats`: Indicates the total number of seats available on the passenger flight.

- `bookedSeats`: Tracks the number of seats booked on the passenger flight.

- `passengers`: An ArrayList storing the passengers booked on the flight.

- `meal`: An ArrayList storing the meal preferences of passengers booked on the flight.

- `crew`: An ArrayList holding the crew members assigned to the passenger flight.

The class constructor (`PassengerFlight`) initializes the passenger flight object with the provided registration number, model, and the total number of seats. It also initializes the passenger and meal lists.

The class provides methods to:

- Retrieve information about the flight, such as its type, registration number, and model.

- Determine the remaining available seats on the flight.

- Book tickets for passengers, optionally specifying their meal preferences.

- Cancel booked tickets for passengers, optionally removing their meal preferences.

- Retrieve the list of passengers booked on the flight and their associated meal preferences.

- Add crew members to the passenger flight.

Overall, the `PassengerFlight` class facilitates the management of passengers, seats, and crew members on a flight, ensuring that bookings are handled appropriately and passenger preferences are accommodated.

Person

The `person` interface represents an abstraction for individuals within the system. It defines common methods that any class representing a person must implement.

The interface declares the following methods:

- `getType()`: An abstract method that returns a string indicating the type of person. This method allows for distinguishing between different types of individuals within the system.

- `getName()`: An abstract method that returns the name of the person. This method provides access to the name attribute of a person object.

Overall, the `person` interface provides a standardized way to interact with and retrieve information about individuals within the system, regardless of their specific roles or characteristics. It allows for uniform treatment of different types of people and promotes consistency in the implementation of person-related functionality.

Plane

The `plane` interface defines a contract for objects representing aircraft within the system. It specifies methods that any class representing a plane must implement.

The interface declares the following methods:

- `getType()`: An abstract method that returns a string indicating the type of aircraft. This method allows for distinguishing between different types of planes within the system.

- `getRegistrationNumber()`: An abstract method that returns the registration number of the aircraft. This method provides access to the unique identifier assigned to the plane.

- `getModel()`: An abstract method that returns the model of the aircraft. This method allows for retrieving information about the specific model of the plane.

Overall, the `plane` interface provides a standardized way to interact with and retrieve information about aircraft objects within the system. It ensures consistency in the implementation of aircraft-related functionality and promotes interoperability between different types of planes.

PremiumGoods

The `PremiumGoods` class represents a specific type of goods or cargo items that are considered premium in nature. It is a subclass of the `Goods` class, inheriting its attributes and behaviors.

The class provides the following constructor:

- `PremiumGoods(double weight)`: Initializes a `PremiumGoods` object with the specified weight, using the constructor of the superclass (`Goods`) to handle the initialization.

The class also declares a constant:

- `type`: Represents the type of goods, which is set to "Premium". This constant is declared as public and static, indicating that it is accessible outside the class and shared among all instances of the class.

The class overrides the `getGoodsType` method from the `Goods` class to provide a specific implementation for premium goods. It returns the value of the `type` constant, indicating that the goods are of premium type.

Overall, the `PremiumGoods` class extends the functionality of the `Goods` class to represent premium cargo items. It allows for the differentiation of goods based on their premium status and provides specific behavior for handling premium goods within the system.

StandardGoods

The `StandardGoods` class represents a specific type of goods or cargo items that are considered standard or regular in nature. It is a subclass of the `Goods` class, inheriting its attributes and behaviors.

The class provides the following constructor:

- `StandardGoods(double weight)`: Initializes a `StandardGoods` object with the specified weight, using the constructor of the superclass (`Goods`) to handle the initialization.

The class also declares a constant:

- `type`: Represents the type of goods, which is set to "Standard". This constant is declared as public and static, indicating that it is accessible outside the class and shared among all instances of the class.

The class overrides the `getGoodsType` method from the `Goods` class to provide a specific implementation for standard goods. It returns the value of the `type` constant, indicating that the goods are of standard type.

Overall, the `StandardGoods` class extends the functionality of the `Goods` class to represent standard cargo items. It allows for the differentiation of goods based on their standard status and provides specific behavior for handling standard goods within the system.

Terminal

The `Terminal` class serves as an abstract representation of a terminal within an airport, where flights arrive and depart. It provides functionality for managing flights assigned to the terminal.

The class has the following instance variables:

- `capacity`: Represents the maximum capacity of the terminal, indicating the maximum number of flights it can accommodate.

- `flights`: An ArrayList storing the `PassengerFlight` objects currently assigned to the terminal.

The class provides the following constructors:

- `Terminal(int capacity)`: Initializes a `Terminal` object with the specified capacity. It also initializes the `flights` ArrayList.

- `Terminal(int capacity, PassengerFlight[] flights)`: Initializes a `Terminal` object with the specified capacity and an array of initial flights.

The class declares the following abstract method:

- `getType()`: An abstract method that subclasses must implement to specify the type of terminal (e.g., Domestic or International).

The class provides methods to:

- Retrieve the capacity of the terminal.

- Retrieve the list of flights currently assigned to the terminal.

- Determine whether a specific flight can be added to the terminal.

- Add a flight to the terminal, if it can accommodate it.

- Remove a flight from the terminal.

The `Terminal` class serves as a base for specific types of terminals (e.g., DomesticTerminal or InternationalTerminal) and provides common functionality for managing flights within the airport environment.

|  |  |  |
| --- | --- | --- |
| Function | Classes used in | Number of time used |
| Overloaded methods | Passenger  PassengerFlight | 8 |
| Overloaded constructors | Terminal | 2 |
| Vararg overloading | PassengerFlight  CargoFlight | 2 |
| Abstract class | Goods  Terminal | 2 |
| Interface | Crew  Person  Plane | 3 |
| Hierarchical Inheritance | PassengerFlight  Terminal | 2 |
| Multiple Inheritance | CrewMember  Passenger  PassengerFlight | 3 |
| Exception handling | CargoFlight  PassengerFlight  Airport | 3 |