UNIVERSITÉ Concordia UNIVERSITY

COEN 6312

Model Driven Software Engineering

Project Deliverable 3

Class diagram and Constraints

Course Instructor
Dr. Wahab Hamou-Lhadj

Team: Techno_Stars
Project Title: Airline Reservation System

Submitted by

Sravan Kumar Thumati	**40070088**
Prashanth reddy Somireddy	**40047831**
Nayana Raj Cheluvaraju	**40071318**
Sri Akhil Varma Alluri	**40082333**
Santosh Reddy Chelluri	**40041074**

Table of Contents

Class Diagram:

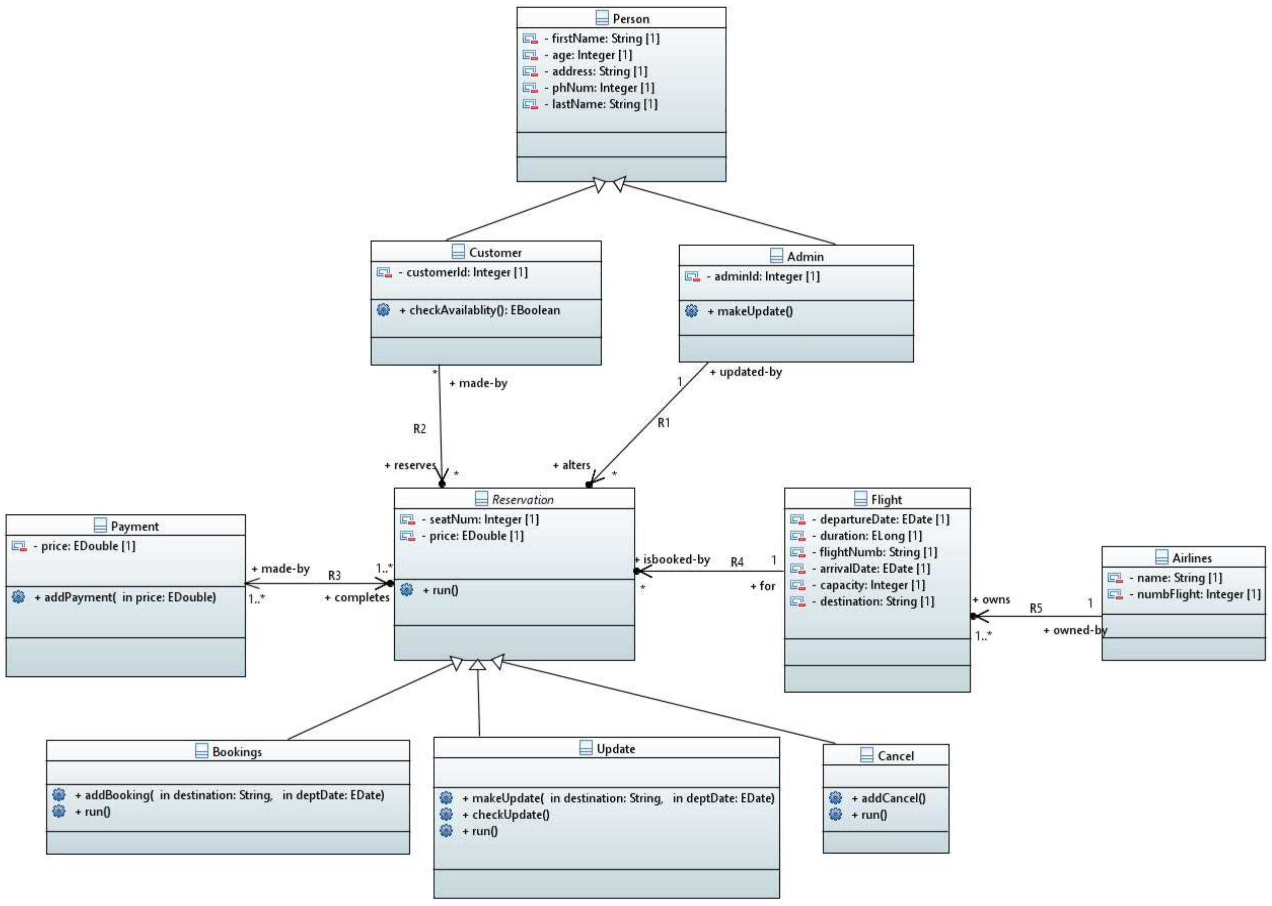

Fig:Class Diagram for Airline Reservation

We have used papyrus for generating class diagram. We are using **strategy design pattern** among Reservation class and child classes of Reservation class. We have implemented a method called run() and we are assigning different behaviour to this method in the derived classes. The customer can use any of these dynamic strategic behaviours. With the use of this pattern we are reducing the coding for writing different code for all different children instead we are writing the code for only run method and adding different behaviours to it for the derived classes.

Major Classes:

Person:

Describes the properties of a person and this class is mainly for updating the person information. This class acts as a base class for the classes Customer and Admin.

- **Data attributes:**
  firstName: String
  lastName: String
  address: String
  age: String
  phNum: Integer

Customer:

This class inherits the basic properties from the base class person and has some unique behaviour. This class is mainly for the customer to book for the flight.

- **Data attributes:**
  customerId: Integer
- **Methods**:
  Boolean checkAvailability(): This method is used by the customer to find available flights based on his preferences. The precondition for this method is that a valid customer and the post condition is the list of available flights based on his preferences .
- **Association:** This class has an association with Reservation class meaning that the multiple customers can reserve for multiple reservations.

Admin:

This class also inherits the basic properties from the Person class and has some unique behaviour . This class is mainly for the Admin who can modify the existing reservations based on the profile of the customers or status of the flights.

- **Data attributes:**
  adminId: Integer
- **Methods:**
  Void makeUpdate(): This method is used by the admin to make the changes to the existing reservations of the customers. The pre condition for this method is a validating customer record exist and the post condition for this method is the updated reservations of the customers.

- **Association:**This class has an association with reservation class meaning that the admin can make changes to the existing multiple reservations or monitors them.

Payment:
This class is mainly for calculating the charges for the customer itinerary.

- **Data attributes:**
  price: Double
- **Methods:**
  Void addPayment(double price): This method is the main method of this class which is used to aggregate the total price for the customer. The precondition for this method is a valid reservation for valid price and the postcondition is the aggregation of the price that results the overall price for the itinerary.
- **Association:**This class has a bi directional association with Reserving class meaning that each of the 3 types of reservations have a relationship with payment and the payment can be calculated multiple for multiple instances of reservations.

Flight:
This class is mainly for calculating the overall money for the particular itinerary of customer.

- **Data attributes:**
  flightNumb:String
  departureDate: Date
  arrivalDate: Date
  duration:Long
  capacity:Integer
- **Association:** This class has an association with Airlines class and Reservations class. Meaning that there can be multiple reservations for a single flight(not from same customer) and the Airlines has multiple flights.

Airlines:
This class is mainly for managing all flights under respective airlines and its associated information.

- **Data attributes:**
  numOfFlights:Integer
  Name: String
- **Association:** Airlines class is associated to flight class by R5, which is used to show the details of different flights. Meaning that an airline contains multiple flights.

Reservation:

Reservation class information about seat number and price of the ticket.

- **Data attributes:**
  seatNum:Integer
  price: Double

- **Methods:**
  void run(): This methods helps in implementing strategy design pattern.
- **Association:** Customer class and admin class are associated to Reservation class by R1 and R2 respectively. If there are available seats customer will be able to reserve a seat for him/her. Admin can make any modifications to the existing booking or even add new booking.

Booking:

Booking class helps to customer or admin to book the ticket/tickets.

- **Methods:**
  addBooking(in destination: String, in deptDate: Date): This method is the main method of this class which is used to Confirm ticket to the passenger. The Precondition of the method is that there exists the availability of seats and there should not be any reservation for that passenger on the same flight and same departure date. Post condition is to confirm the tickets.
  run(): This method extends the run() method from Reservation class which is the parent class for Booking class. During run time if user decides to reserve the ticket, this behavior will be invoked. This method routes back to addBooking(in destination: String, in deptDate: Date) functionality.
- **Association:** Booking class inherits the functionalities like seatNum, price etc., from Reservation class

Update:

Update class helps to customer to modify the booking details of the ticket/tickets and also to check if there are any updates on the booking details.

- **Methods:**
  makeUpdate(in destination: String, in deptDate: Date): This method helps customer to modify the booking details.post condition is that there exists the booking to make modifications. Post condition is to replicate the changes.
  checkUpdate(): This method helps users to check if there are any changes made to their booking details (E.g.: checking changes in flight timings, changes by admins etc.,). Pre

condition is that there exists a modification to the reservation. Post condition is to return the modified reservation details.
run(): This method extends the run() method from Reservation class which is the parent class for Update class. During run time if user decides to make modifications to the reserved ticket, this behavior will be invoked. This method routes back to makeUpdate(in destination: String, in deptDate: Date) functionality.

- **Association:** Update class inherits the functionalities like seatNum, price etc., from Reservation class. Admin cannot operate on this class.

Cancel:
Cancel class helps to customer or admin to cancel the ticket/tickets booking.

- **Methods:**
  makeCancel(): This method helps users to cancel their booking. The Precondition of the method is that there exists a reservation. Post condition is to cancel the tickets.
  run(): This method extends the run() method from Reservation class which is the parent class for Cancel class. During run time if user decides to cancel the ticket, this behavior will be invoked. This method routes back to addCancel() functionality.
- **Association:** Cancel class inherits the functionalities like seatNum, price etc., from Reservation class.

OCL Expressions:

Condition	Each Customer should have unique customerId
Expression	Context Customer: Inv: allInstances()->forAll(c1,c2 :Customer| c1<>c2 implies c1.customerId<>c2.customerId)

Condition	Each person should have a valid firstName and lastname
Expression	Context Person: Inv: self.firstName->length()>0 and self.lastName->length()>0

Condition	The flight capacity should be at most 250 passengers

Expression	Context Flight: Inv: self.capacity>0 AND self.capacity<=250

Condition	Each Admin should have unique Id
Expression	Context Admin: Inv: allInstances()->forAll(a1,a2 :Admin| a1<>a2 implies a1.adminId<>a2.adminId)

Condition	Payment should be calculated for the customer
Expression	Context Payment::addPayment(double p): void pre: self.R3 ->notEmpty() and p<>0 Post: self.price=price@pre+p

Condition	Admin should be able to modify the reservations
Expression	Context Admin::makeUpdate(): void pre: self.R1 ->notEmpty() Post: self.R1 -> size() =if(self.R1.oclIsKindOf(Update)) then self.R1->size@pre+1 endif

Condition	Admin must be of 45 years old or more
Expression	Context Person: Inv: self.age >= if(self.oclIsKindOf(Admin)) then 45 endif

Condition	Each Customer must be 18 years to be able to reserve a flight
Expression	Context Person: Inv: self.age >= if(self.oclIsKindOf(Customer)) then 18 endif

Condition	Payment should be calculated for the customer

Expression	Context Payment::addPayment(double p): void pre: self.R3 ->notEmpty() and p<>0 Post: self.price=price@pre+p

Condition	Airlines should have unique names
Expression	Context:Airlines Inv: self.Airlines->select(n1,n2 :Airlines| n1<>n2 implies n1.name <> n2.name)

Condition	Airlines should have the total flights of at least 10 and atmost 50.
Expression	Context:Airlines Inv: self.R5->size()>=10 AND self.R5->size()<=50

Condition	The customer must be able to Book a reservation
Expression	Context Reservation::addBooking(String destination, Date departureDate): void pre: self.R4.getCapacity()>0 AND self.R4.getCapacity()<249 Post: self.R4.getCapacity()= self.R4.getCapacity()@pre+1

Condition	The customer must be able to cancel the reservation
Expression	Context Reservation::addCancel(): void pre: self.R2.size() > if(self.oclIsKindOf(Cancel)) then 0 Post: self.R4.getCapacity()= self.R4.getCapacity()@pre-1

Condition	Every passenger of the same flight should be allocated different seat number.
Expression	Context Flight inv: self.R4->forAll(R1, R2 : Reservation | R1 <> R2 implies R1.getSeatNum() <> R2.getSeatNum())

Condition	One passenger cannot be able to book more than one seat on the same flight.
Expression	Context Customer inv: self.R2.R4->forAll(F1, F2 : Flight | (F1 <> F2 AND F1.getflightNumb <> F2.getflightNumb) implies (F1.getDestination() <> F2.getDestination() AND F1. getDeptartureDate() <> F2.getDeptartureDate()))