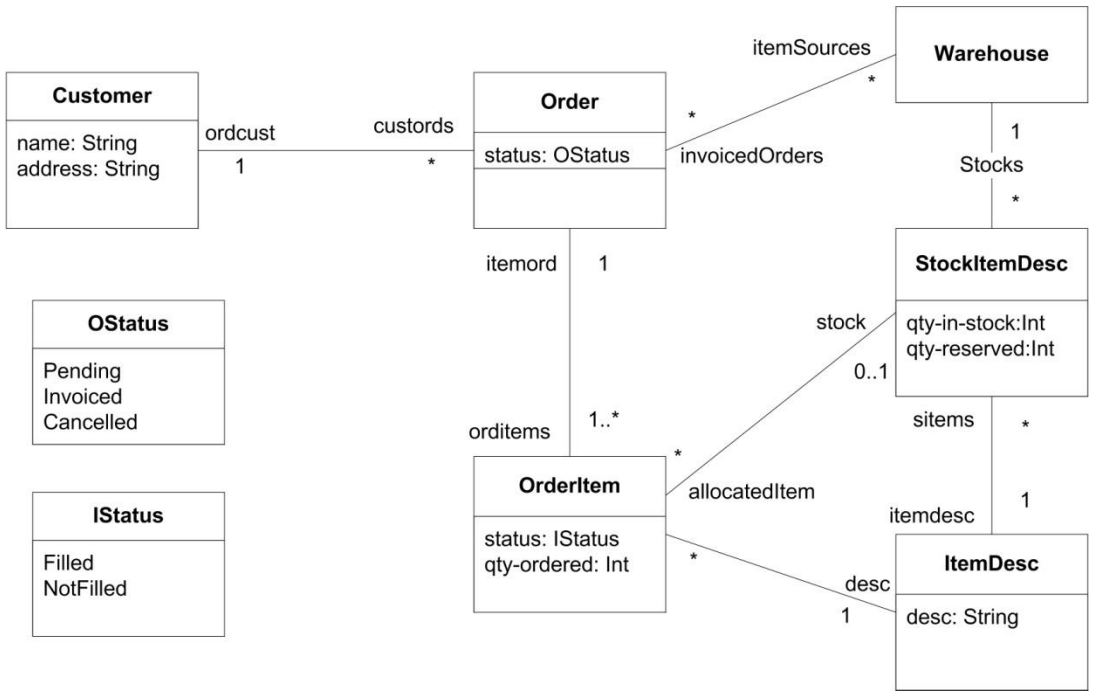
SWEN 746 –Model-Driven Development
Rochester Institute of Technology

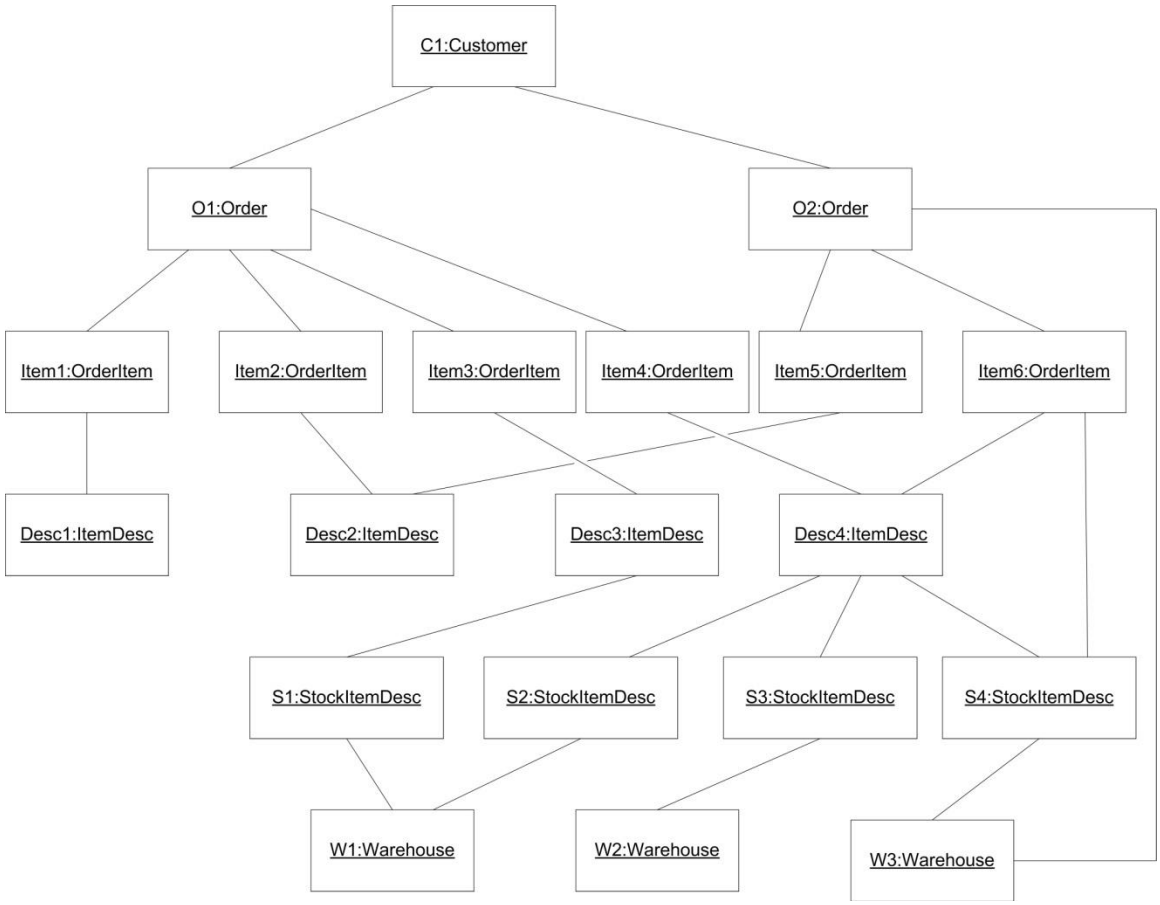
OCL - Modeling HW7 (*50 points*)

Q1:

The following is a Class Model for a simple order processing system (SOPS). In the UML class diagram, the class Order has an attribute, *status* with an enumerated type. Status can be one of the following values: *Pending, Invoiced, Cancelled*. A pending order is one that has not been filled. An invoiced order is one that has been filled (i.e., stocked items have been allocated to all the order items of the order) and a cancelled order is one that has been cancelled by the customer. Each order item of an order is also associated with a status that can have one of the following values: *Filled, NotFilled*. The *Warehouse* class represents warehouses that contain products (items). For each type of product (item) stored in a warehouse (a *Warehouse* object) there is a *StockItemDesc* that contains information about the total number of products of the type stored in the warehouse and the total number of reserved products of the type. Order items that are filled are linked to the *StockItemDesc* object from which products have been allocated. An invoiced order is linked to all the warehouses (objects of *Warehouse*) from which order items are filled.



Part A – OCL Queries: Evaluate the following OCL expressions using the SOPS object diagram given below. The expressions all return collections of objects. Use object names shown in the object diagram to refer to objects, for example, the expression C1.Order.orditems evaluates to {Item1, Item2, Item3, Item4, Item5, Item6}, that is, C1.O1.orditems = {Item1, Item2, Item3, Item4} **(10 points)**



- (2 points) O2.OrderItem.ItemDesc.StockItemDesc = {S2, S3, S4}
- (2 points) W1.StockItemDesc.itemdesc.OrderItem = {Item3, Item4,Item6}
- (3 points) O1.OrderItem → select(i | i.ItemDesc.StockItemDesc → isEmpty()) = {Item1,Item2}
- (3 points) O1.OrderItem → select(i | i.ItemDesc.StockItemDesc → size() > 1) = {Item4}

Part B: Complete the following OCL constraints (10 points):

- (3 points) If an order is invoiced then all its order items are filled.

  context Order
  inv: if self.status = OStatus::Invoiced

  then self.**invoiced** → forAll(item | item.status = **filled**)

- (3 points) If an order item is filled then it is linked to a *StockItemDesc* object.

  context OrderItem
  inv: if self.status = IStatus::filled

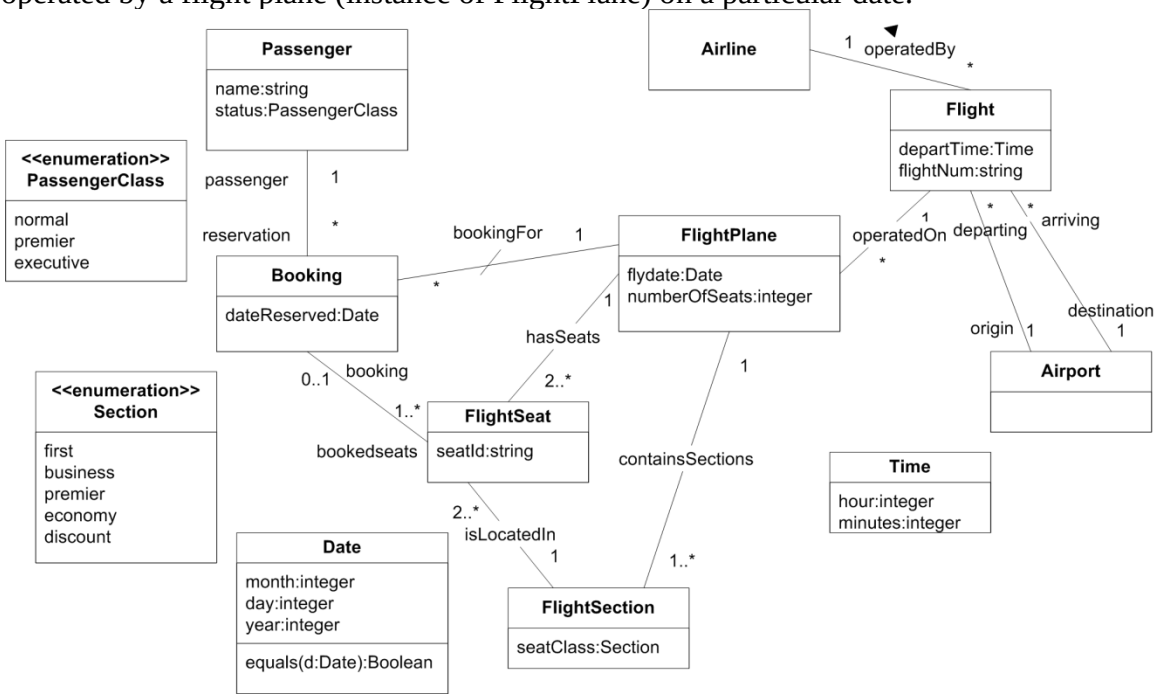
  then self.stock → self.***StockItemDesc***

- (4 points) If an order item is linked to a *StockItemDesc* object then the associated order is linked to the warehouse linked to the *StockItemDesc* object.

  context OrderItem
  inv: if self.stock → **self.*StockItemDesc***

  then self.itemord.**warehouse** →includesAll(self.itemord.**orders**)

Q2:
The following questions pertain to the class model for an airline reservation system shown below. The diagram describes a system that maintains information about flights (instance of Flight) and bookings (instances of Booking). Passengers are booked on flights and are assigned seats (instances of FlightSeat) at the time of booking. A flight is operated by a flight plane (instance of FlightPlane) on a particular date.



The following are OCL statements associated with model elements in the domain model. State in English the constraints they express:

- **(5 points)**
  Context Booking
  inv: self.bookedseats.hasSeats → forAll(f |f = self.bookingFor)

  This invariant shows that the hasSeats Object by FlightSeat Class is equals to bookedFlight

- **(5 points)**
  Context FlightPlane
  inv: self.numberOfSeats = self.hasSeats → size()
  This invariant shows that FlightPlane Class , numberOfSeats value is number of FlightSeat Object

- **(5 points)**
  Context FlightSection
  inv: (seatClass = Section::first or seatClass = Section::business or
  seatClass = Section::premier)
  implies isLocatedIn → collect (f.booking.passenger.status) →
  forAll(s | s = PassengerClass:: executive or s = PassengerClass:: premier)

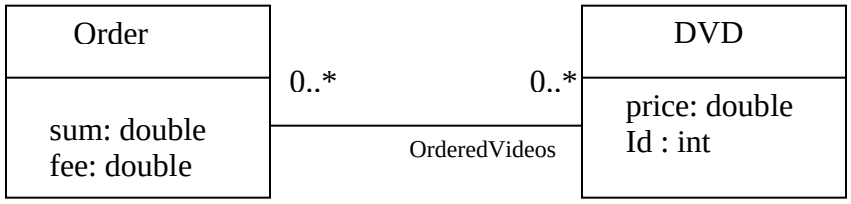
This invariant shows that Flight Section Class's seatClass Attribute of values "first", "business", "premier" must be present only for Passenger of status "executive" and "premier"

Q3: What are operation contracts? What are the advantages and disadvantages of using OCL in specification of operation contracts? Justify your answer by giving an example. **(5 points)**

- Operation contracts use a pre- and post-condition form to describe detailed changes to objects in a domain model, as the result of a system operation.
- Advantages are:
- contributes to avoiding misunderstandings and hard-to-track bugs; because assumptions and promises are explicit: if all contracts are explicit and dovetail nicely, the bug is in the code that doesn't fulfil its contract
- supports clear documentation of a module– clients should not feel the need to read the code! reading the contract should be enough

- supports defensive programming; e.g., when you implement an operation, verify that the postcondition holds before returning; o/w fail gracefully and report a bug
- allows avoidance of double testing. e.g., when you implement an operation, you need not test that your preconditions are satisfied: that's the job of the client to ensure.

Q4 Invariants: (**10 Points**)
Write the OCL equivalent of the following invariants for the given model



a. Give an OCL invariant that specifies that the *sum* attribute cannot be negative.

Context Order
inv: self.sum > 0

b. Give an OCL invariant that specifies that the *sum* attribute will be zero if no DVDs are ordered.

context DVD
inv: if self.OrderedVideos -> size() = 0

then Order.Sum = 0

c. Give an OCL invariant that specifies that the *sum* attribute really describes the price of the DVDs ordered.

context Order
inv: self.Sum = OrderedVideos.price -> sum()

d. Give an OCL invariant that specifies that different instances of DVD have different *Id*

context DVD
inv Id : DVD.allInstances ->isUnique(Id)