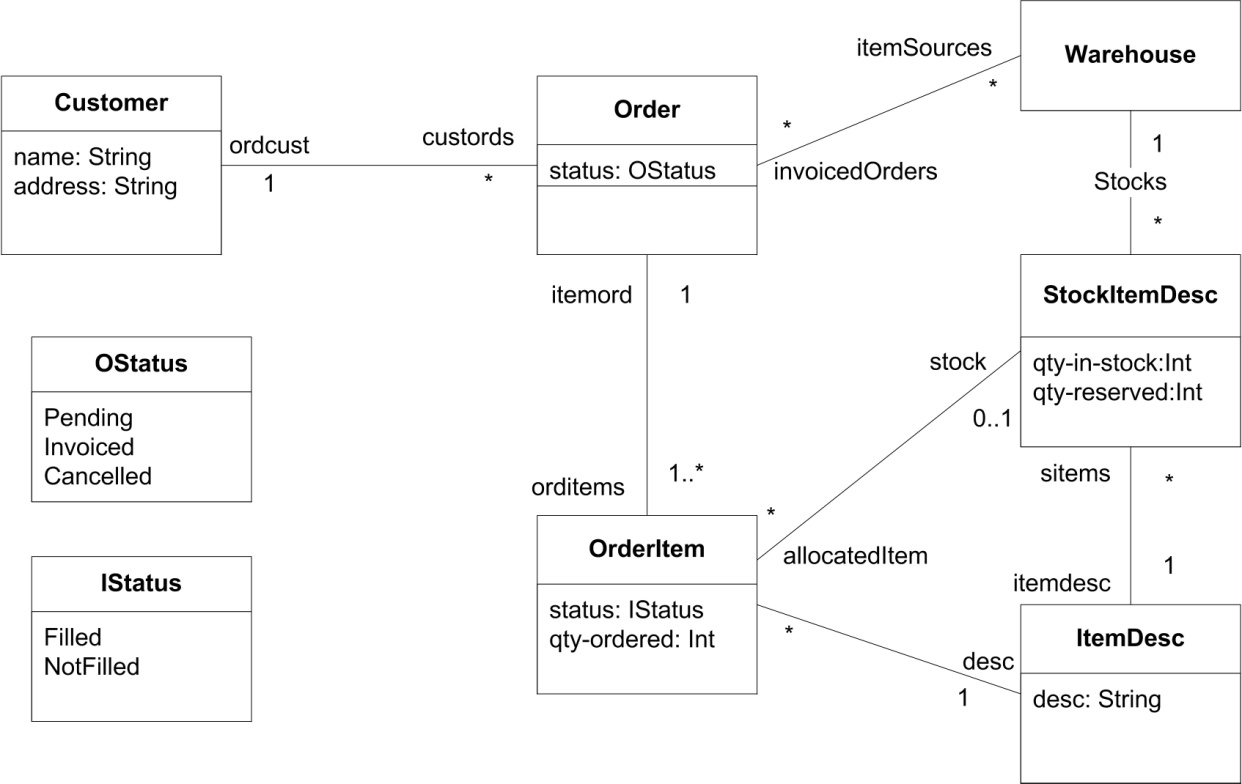
OCL - Modeling HW6 (*50 points*)

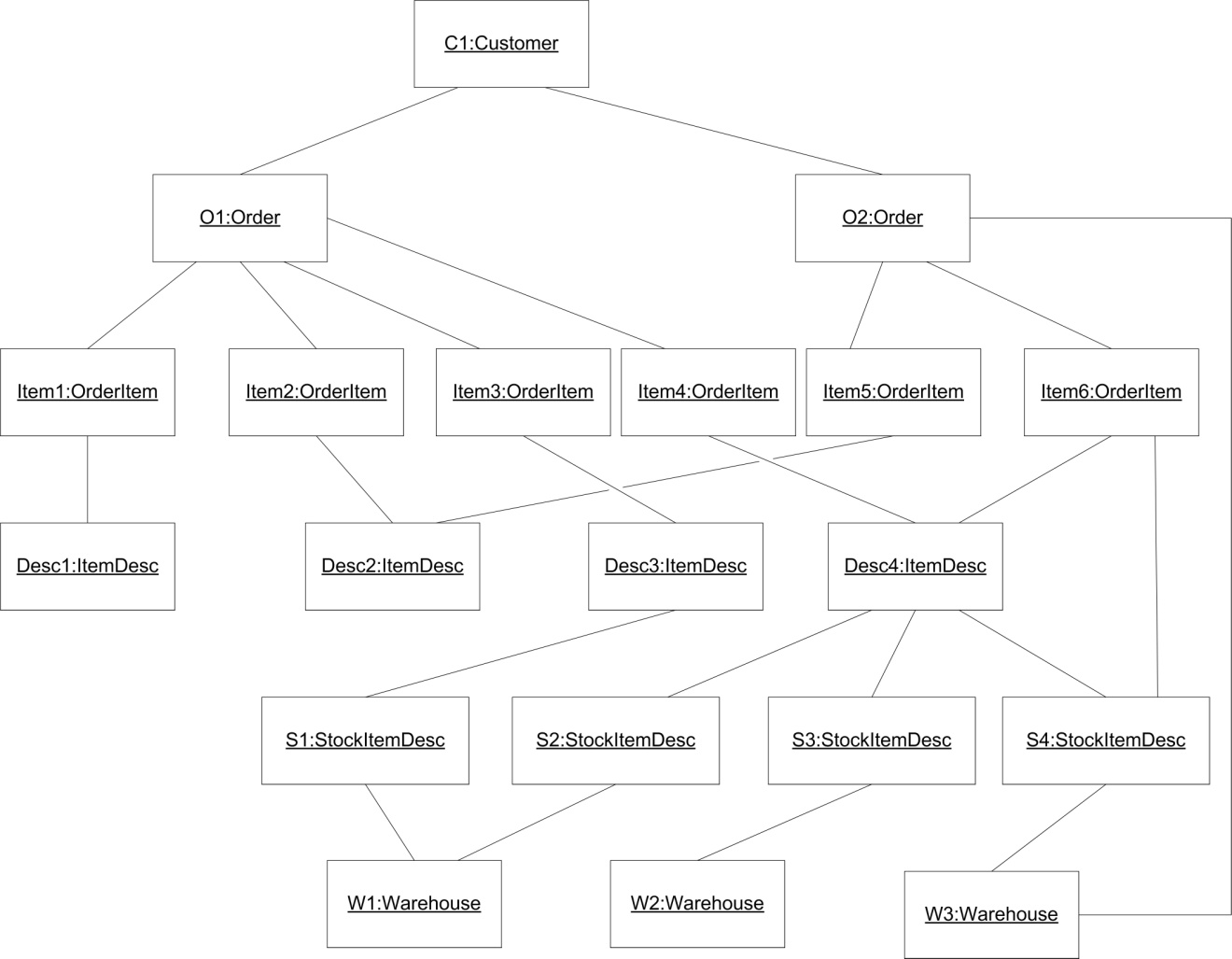
**Q1:**

The following is a Class Model for a simple order processing system (SOPS). In the 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:** 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,

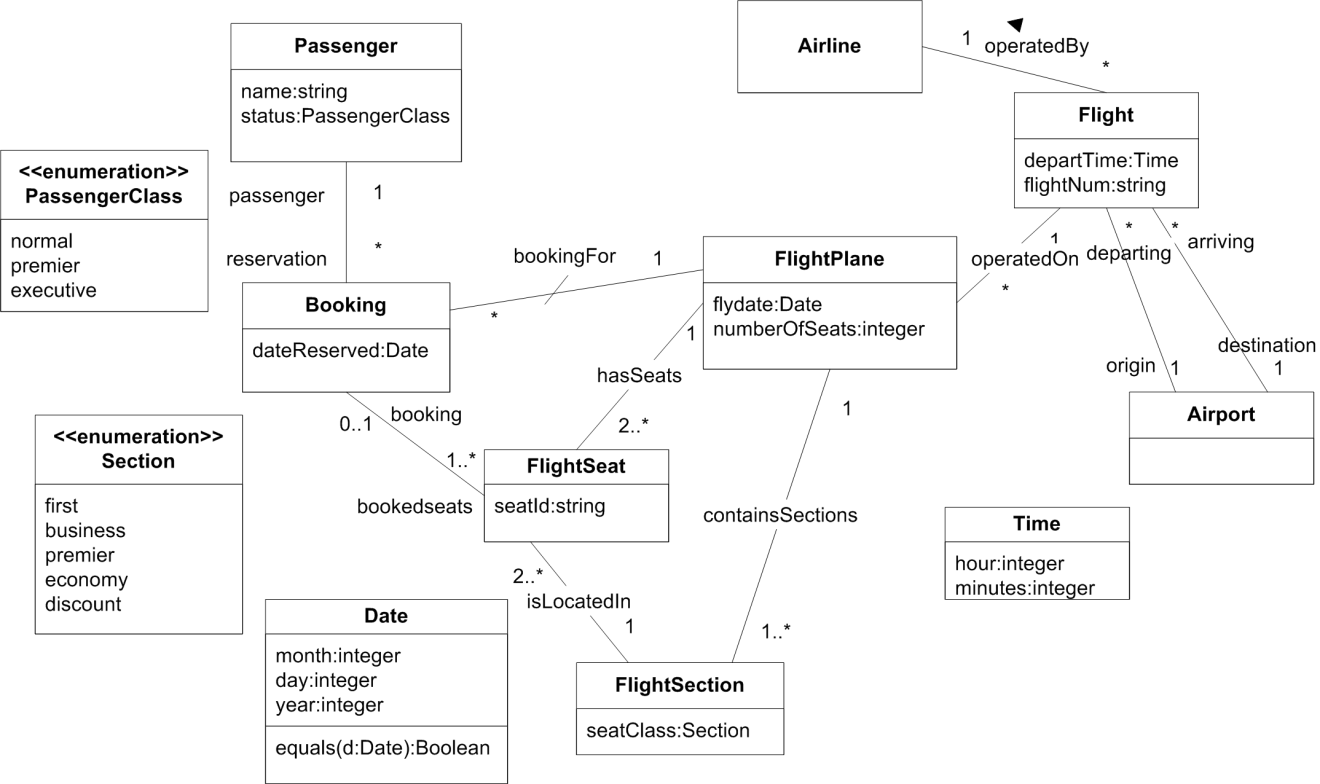
C1.O1.orditems = {Item1, Item2, Item3, Item4} **(10 points)**  
  


* (2 points) O2.OrderItem.ItemDesc.StockItemDesc =
* (2 points) W1.StockItemDesc.itemdesc.OrderItem =
* (3 points) O1.OrderItem 🡪 select(i | i.ItemDesc.StockItemDesc 🡪 isEmpty()) =
* (3 points) O1.OrderItem 🡪 select(i | i.ItemDesc.StockItemDesc 🡪 size() > 1) =

**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.\_\_\_\_\_\_\_\_\_\_\_\_\_ 🡪 forAll(item | item.status = \_\_\_\_\_\_\_\_\_\_\_\_\_\_ )
* (3 points) If an order item is filled then it is linked to a *StockItemDesc* object.  
    
  **context** OrderItem  
  **inv**: if self.status = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
    
   then self.stock 🡪 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
* (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 🡪 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
    
   then self.itemord. \_\_\_\_\_\_\_\_\_\_\_\_\_ 🡪includesAll(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )

**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)
  + **(5 points)   
    Context** FlightPlane  
    **inv**: self.numberOfSeats = self.hasSeats 🡪 size()
  + **(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)

**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)**

**Q4:** (**10 Points**)

Write the OCL equivalent of the following invariants for the given model

DVD

Order

sum: double

fee: double

price: double

Id : int

OrderedVideos

0..\*

0..\*

1. Give an OCL invariant that specifies that the *sum* attribute cannot be negative.
2. Give an OCL invariant that specifies that the *sum* attribute will be zero if no DVDs are ordered.
3. Give an OCL invariant that specifies that the *sum* attribute really describes the price of the DVDs ordered.
4. Give an OCL invariant that specifies that different instances of DVD have different *Id*