

# ESOF 422 Homework 2

---

## **Instructions:**

Work with your partner on this homework.

Make sure your printout is stapled together and all names appear in the front page.

Hand in copies (printouts) of UML class, UML object and UML sequence diagrams.

Hand in copies (printouts) of .use, and .x files.

Your homework is worth 30 points.

**Due:** 2/9 (Friday) during class. –No exceptions so plan accordingly.

## **Question 1 (5 pts)**

In this exercise we will use the previously developed version of the Movie Rental model and the imperative SOIL enhancements to further refine its structure. You will find a copy of the .use file with this assignment.

### **Instructions**

1. Start with the .use file provided
2. You will now refactor the design to move responsibilities to the classes that have the information and to add a new method to the Customer class that calculates the total charges.
  - a. Move calculation of the amount charged for each rental from the *Customer.getAmount()* method to the Rental class in a new method called *getCharge()*.
  - b. Add a new method called *getTotalCharge()* to the Customer class to sum up all the charges.
1. Printout a copy of your Class diagram showing all attributes, associations, roles, multiplicities and operations.
2. Printout an Object diagram showing a valid state of the system. Make sure you 'check' the validity of the structure.
3. Printout a Sequence diagram that shows the execution of the *Statement()* call on a customer.

## **Question 2 (15 pts)**

Select any design pattern (except Singleton) and create a USE model description of a generic version of your design pattern. Make sure you include all operations (with SOIL), attributes and multiplicities as intended by the pattern's design. You will then add OCL constraints as needed to decorate your pattern. Now generate a valid instantiation of said design pattern making sure that no constraint is violated. Your constraints should come in the form of invariants, and/or pre and post conditions.

Trace a representative call of the pattern and also print out the sequence diagram.

Printout a description of the pattern you selected.

Printout a copy of your Class and valid Object diagram and hand them in with your .use, and .x files.

```
-- This is a USE model that has embedded SOIL operations in it
--
```

```
model MovieRental
```

```
enum PriceCode {regular, family, newRelease}
```

```
--classes
```

```
class Customer
```

```
attributes
```

```
    name:String
```

```
    numRen:Integer
```

```
operations
```

```
    addRental()
```

```
        begin
```

```
        end
```

```
    getName()
```

```
    getAmount(aRen:Rental):Real
```

```
        begin
```

```
            declare wrkCh:Real, m:Movie, pc:PriceCode,dy:Integer;
```

```
            m:=aRen.getMovie();
```

```
            dy:=aRen.getDaysRented();
```

```
            pc:=m.getPriceCode();
```

```
            wrkCh:=0;
```

```
            if pc=PriceCode::regular then
```

```
                wrkCh:=2.0;
```

```
                if dy > 2 then
```

```
                    wrkCh:=wrkCh + (dy - 2) * 1.5;
```

```
                end;
```

```
            end;
```

```
            if pc=PriceCode::family then
```

```
                wrkCh:=1.5;
```

```
                if dy > 3 then
```

```
                    wrkCh:=wrkCh + (dy - 3) * 1.5;
```

```
                end;
```

```
            end;
```

```
            if pc=PriceCode::newRelease then
```

```
                wrkCh:=dy * 3.0;
```

```
            end;
```

```

        result:=wrkCh;
    end

Statement()
begin
    declare aCharge:Charge, sm:Movie, ch:Real, t:String;

    self.numRen:=self.rentals->size();
    for ren in self.rentals do
        ch:=self.getAmount(ren);
        sm:=ren.getMovie();
        t:=sm.getTitle();
        aCharge:= new Charge;
        aCharge.chVal:=ch;
        aCharge.chT:=t;
        insert(self,aCharge) into customerCharges
    end
end

end

class Rental
attributes
    daysRented:Integer

operations
    getDaysRented():Integer
    begin
        result := self.daysRented;
    end

    getMovie(): Movie
    begin
        result := self.movie;
    end
end

class Movie
attributes
    title:String
    priceCode:PriceCode

operations
    getPriceCode():PriceCode
    begin
        result := self.priceCode;
    end

    setPriceCode(code:PriceCode)
    begin
        self.priceCode := code;
    end
end

```

```

    getTitle():String
    begin
        result := self.title;
    end
end

class Charge
attributes
    chVal:Real
    chT: String

operations
end

--associations

association custRentals between
    Customer [1] role renter
    Rental [0..*] role rentals
end

association movRental between
    Rental [0..*] role movRentals
    Movie [1] role movie
end

association customerCharges between
    Customer [1] role cust
    Charge [0..*] role charges
end

--constraints

constraints
--Example constraints
--You may remove these constraints in your design.  They are here
--just as examples.

context Customer
    inv maxRental:numRen <= 10
    inv agreement:rentals->size = numRen
    inv rentals:rentals->notEmpty
    inv daysRented:rentals->select(daysRented > 3)->notEmpty

```