FILS, English Stream, Year 3
Course: Software Development Methods
Prof. Luca Dan Serbanati

"E-Film Hiring" Project

FOURTH MACRO-ACTIVITY: SOFTWARE BEHAVIORAL MODEL

Deliveries: Interaction Diagrams

Procedure

1. Design the interaction (sequence or collaboration) diagrams for each use-case and system operation identified in the previous homework. Use also system operation contracts when they are available. Do not hesitate to introduce new classes when needed:

- Introduce a controller for each use case: a use case will be managed by at least one controller and each controller manages at least one use case.
- Introduce graphical interface classes such that each time when an actor interacts with the system he/she does through a GUI.

2. Give reason for your design decisions by indicating the used patterns.

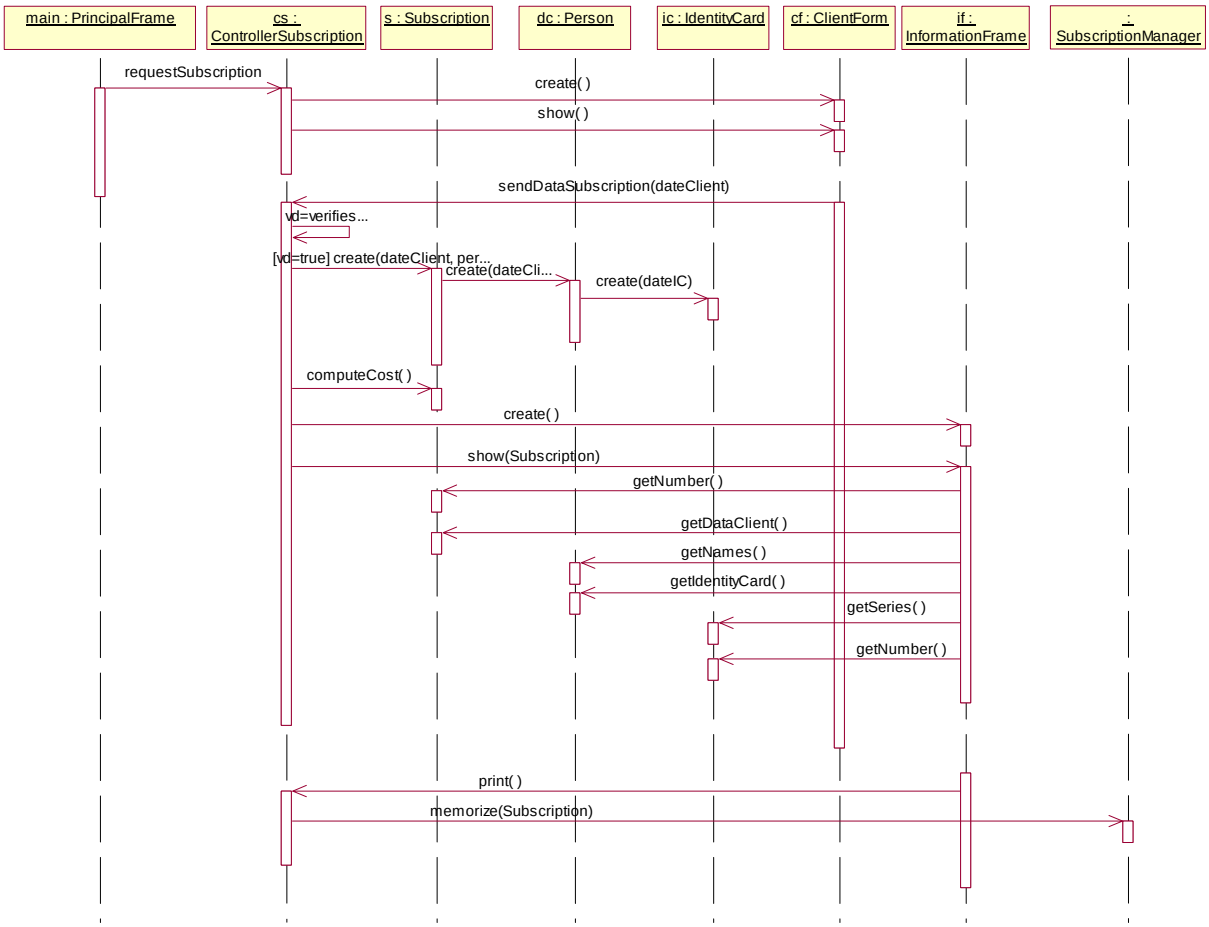
3. Complete the static model with the classes created in this step.

We will apply the above-described procedure for each use case.

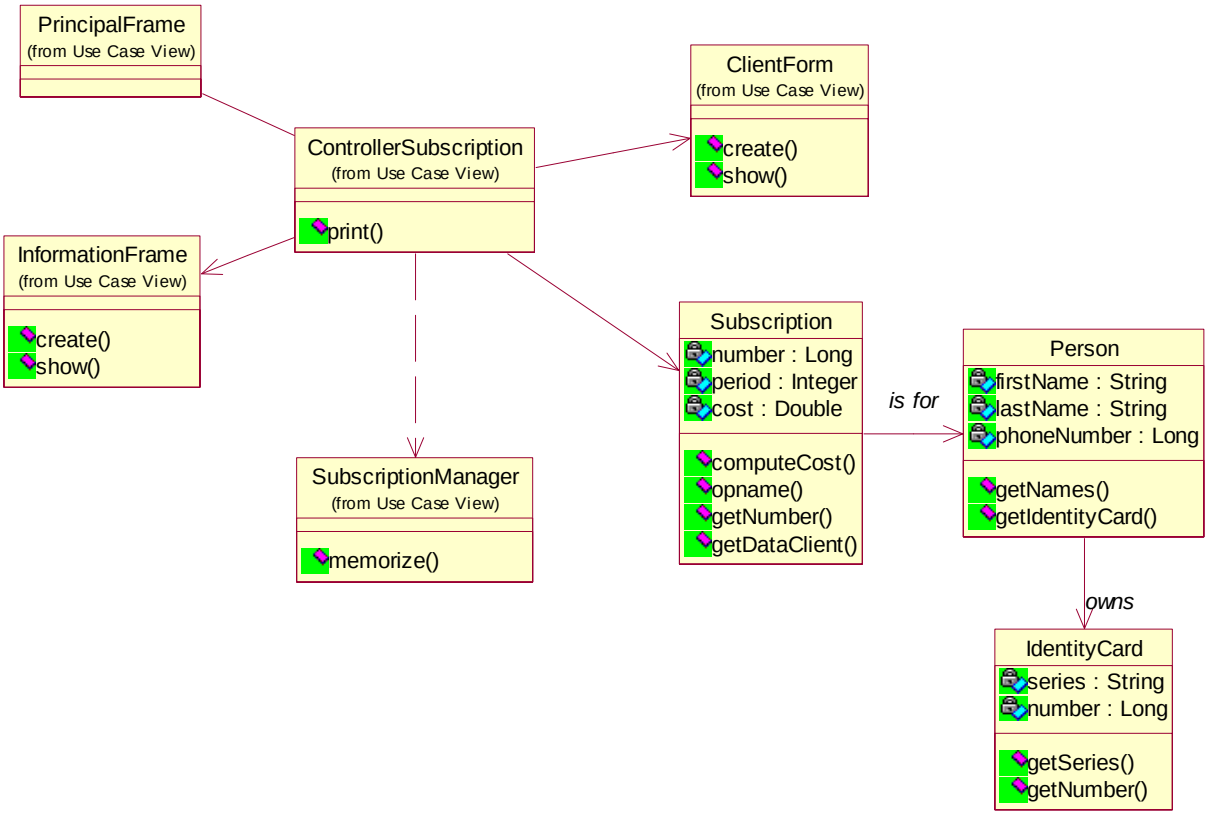
Subscription creation

1. In order to create the sequence diagram for this use case we introduced the following new classes with the needed responsibilities:
    a. ControllerSubscription class manages this use case and deals with all events generated by user in user graphical interfaces.
    b. PrincipalFrame and ClientForm are GUIs that have the responsibility to receive, ask user to insert the necessary data needed to supply the desired functionality, or to display all the information.
    c. SubscriptionManager is a class that is responsible to save/ load data in/ from database the data of the subscriptions.

With these new classes and all classes from the conceptual model that collaborate in order to realize the use case we obtain the following sequence diagram:

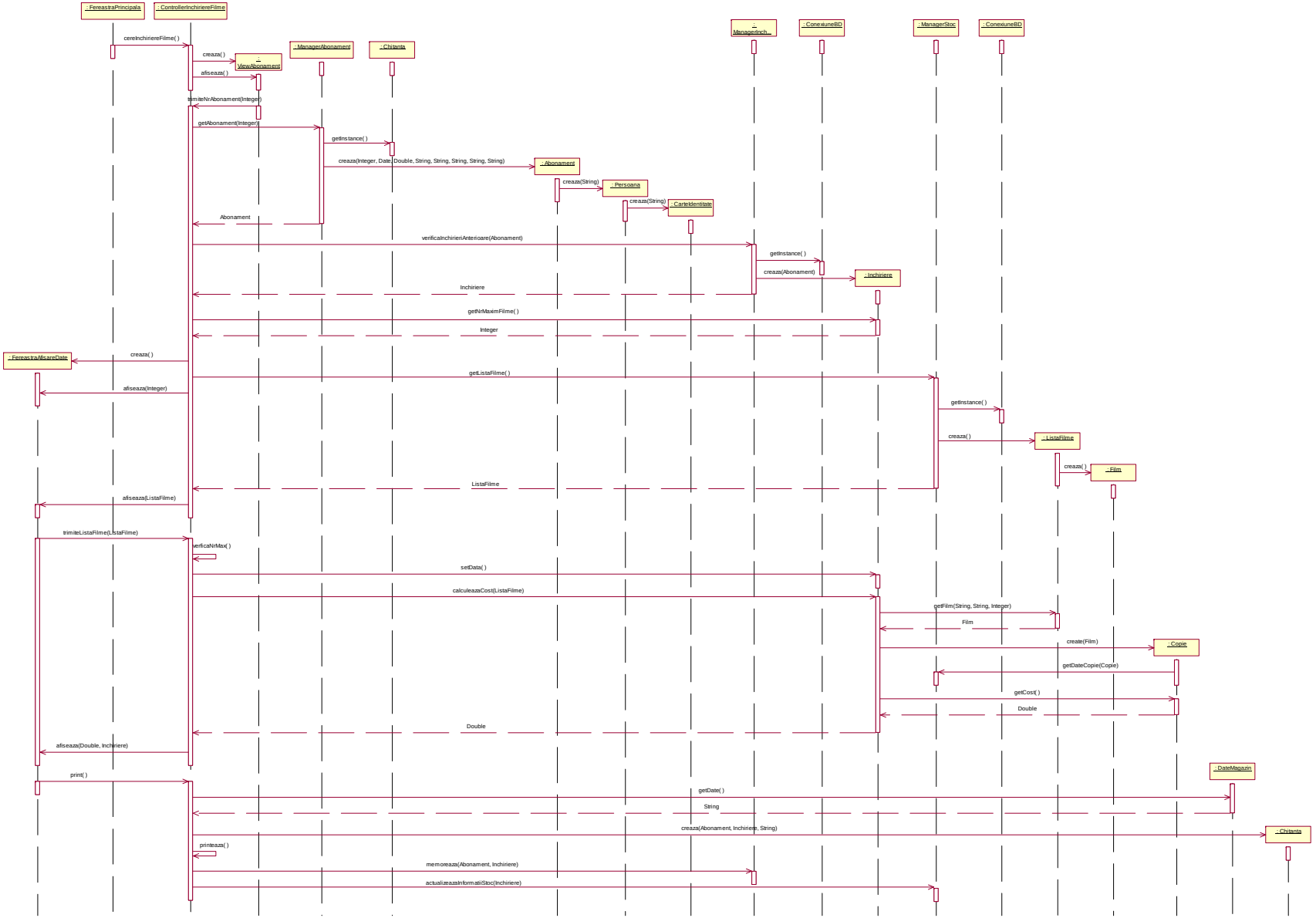


2. In constructing of the above diagram we used the following patterns: Information Expert, Low coupling, High Cohesion and Creator:
    a. The ControllerSubscription class has the responsibility to create an object by type Subscription because it has all required information (applying the Creator design pattern) on which it receives from the ClientForm frame.
    b. The responsibility to compute the cost of subscription is assigned to the Subscription class because this has the information (memorized in the attribute period) required to carry out the responsibility. In this case we use the Expert pattern which promotes low coupling (by putting methods in the classes that have the information required by the methods).
    c. The ClientForm class also has the responsibility to show information about subscription, namely: the number of subscription, the client data (first and last names, phone number, data of identity card), and the period of availability (8 or 12 months).
    d. The responsibility to print the subscription is assigned to the ControllerSubscription class because this controller manages all the events generated by user during this use case resolution and has all necessary information, because it knows the information from subscription.
3. From the previous step 2.a, it results that between the classes ClientForm and ControllerSubscription, the later and the Subscription class we have association relations.



Renting movies

1. In order to create the collaboration diagram for this use case we introduced the following new classes with the needed responsibilities:
   a. ControllerInchiriereFilme class manages this use case and deals with all events generated by user in user graphical interfaces.
   b. ManagerInchirieri class that deals with hiring management. This class was introduced because of the reason:
      i. Now we can deal with the following requirement: “A client may hire (in totally, up to 5 movies) more than one time if he/she didn’t exceed the return date”.


: FereastraPrincipala
: ControllerInchiriereFilme
: ManagerInch...
: ConexiuneBD
: ManagerStoc
: ConexiuneBD
cereInchiriereFilme( )
creaza( )
: ViewAbonament
: ManagerAbonament
: Chitanta
afiseaza( )
trimiteNrAbonament(Integer)
getAbonament(Integer)
getInstance( )
creaza(Integer, Date, Double, String, String, String, String, String)
: Abonament
creaza(String)
: Persoana
creaza(String)
: CarteIdentitate
Abonament
verificaInchirieriAnterioare(Abonament)
getInstance( )
creaza(Abonament)
: Inchiriere
Inchiriere
getNrMaximFilme( )
Integer
: FereastraAfisareDate
creaza( )
getListaFilme( )
afiseaza(Integer)
getInstance( )
creaza( )
: ListaFilme
creaza( )
: Film
ListaFilme
afiseaza(ListaFilme)
trimiteListaFilme(ListaFilme)
verificaNrMax( )
setData( )
calculeazaCost(ListaFilme)
getFilm(String, String, Integer)
Film
create(Film)
: Copie
getDateCopie(Copie)
getCost( )
Double
Double
afiseaza(Double, Inchiriere)
: DateMagazin
print( )
getDate( )
String
creaza(Abonament, Inchiriere, String)
: Chitanta
printeaza( )
memoreaza(Abonament, Inchiriere)
actualizeazaInformatiiStoc(Inchiriere)

Design class diagram