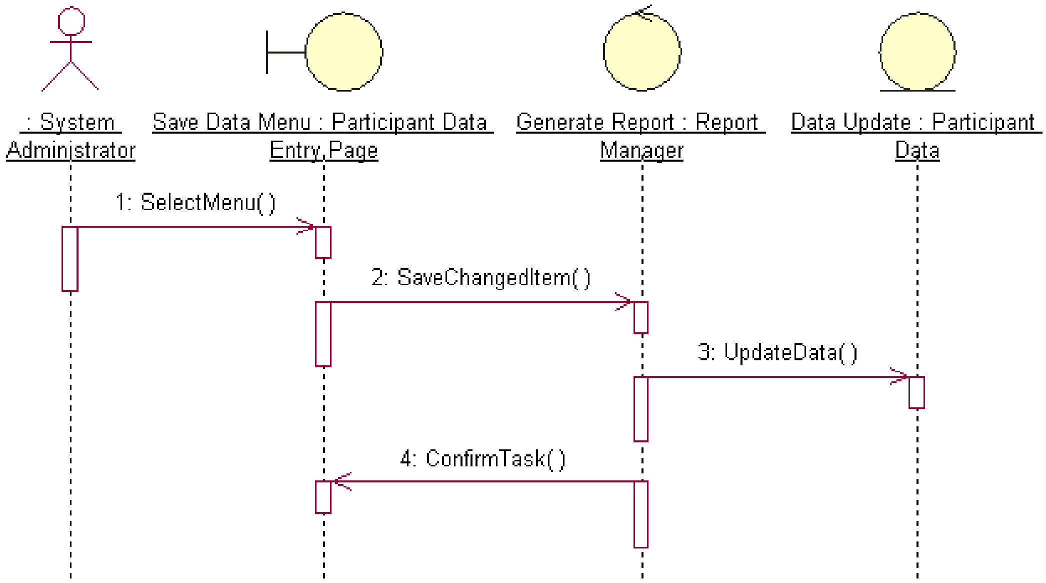
WIA2002 Software Modelling
Semester 1, 2016/17
Tutorial 8

1. **You are asked to develop a library information system (LIS) for a public library. The LIS will be used to handle book loan and return procedures. Draw a communication diagram for a book return scenario in which the book is overdue. You need to include entity, boundary and control classes in this diagram.**

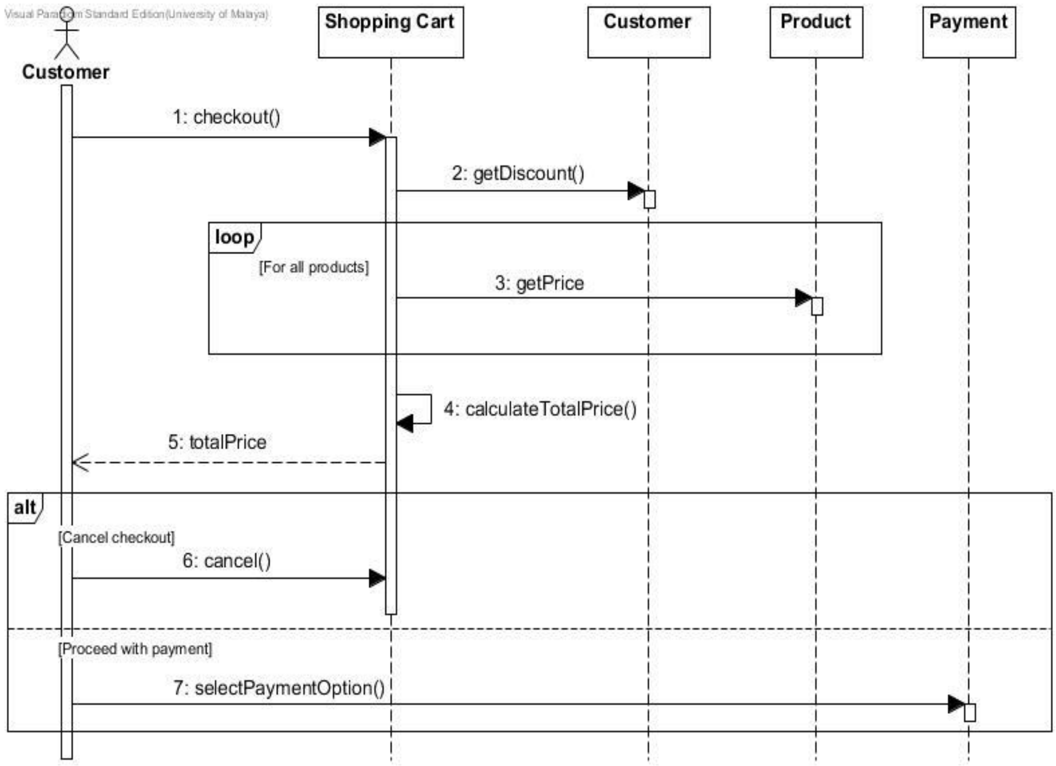
   Return Book Scenario:

   Books are returned through a collection box at the library entrance. When the Library assistants check in the returned books with the LIS administrator terminals, he/she has to scan the barcode of the book. The system displays the book details and the due date of the book loan. If a book is returned late, an overdue fine is calculated and a new fine record is created. Overdue fine is $1 per day. Once the book return is completed, the system will display a message “Check in Finish”.

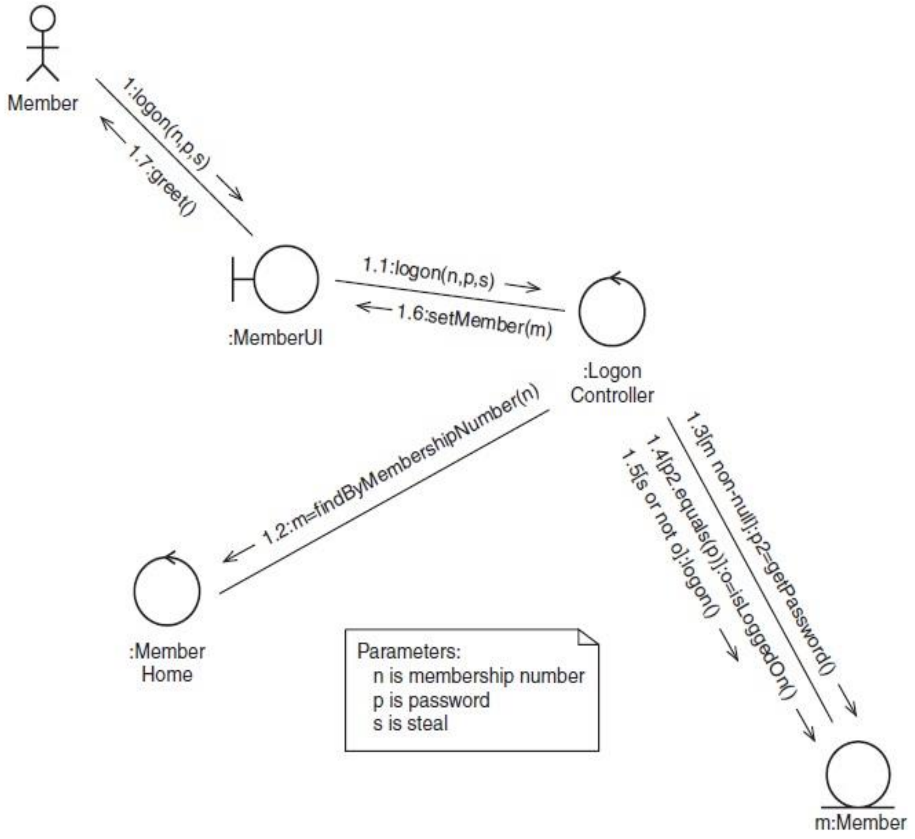
2. **Draw a communication diagram based on the following UML sequence diagram.**


: System Administrator
Save Data Menu : Participant Data Entry Page
Generate Report : Report Manager
Data Update : Participant Data
1: SelectMenu( )
2: SaveChangedItem( )
3: UpdateData( )
4: ConfirmTask( )

3. **Draw a communication diagram based on the following UML sequence diagram.**

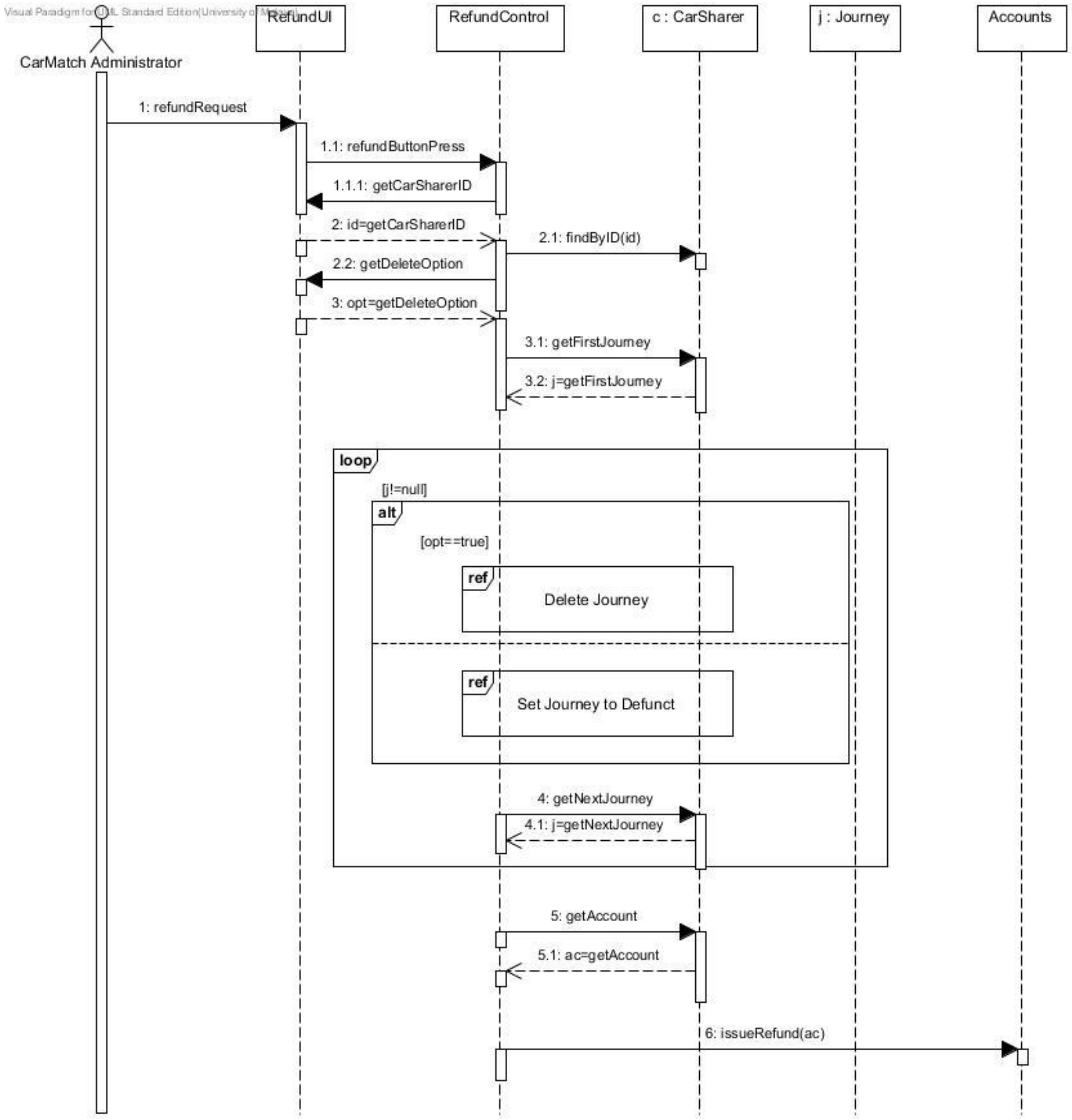


4. **Draw a sequence diagram based on the following UML communication diagram.**



5. Interaction Overview Diagram

Convert the following Sequence Diagram for Refund Membership Fee to an Interaction Overview Diagram:



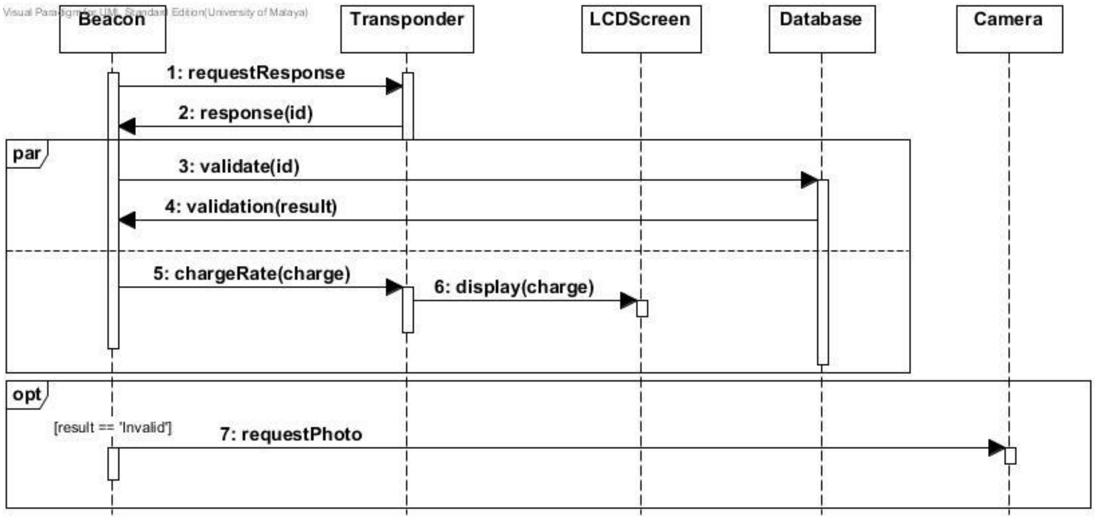
Tips: Steps to produce an Interaction Overview Diagram

1. Decide on the interaction model:
    a. This sequence diagram has a loop and alternative branches in it. We shall focus on the lifeline control class RefundControl.
2. Break up the interaction
    a. In the sequence diagram, we already have two referenced interaction occurrences: Delete Journey and Set Journey to Defunct.
    b. The messages before the loop can be considered as one interaction occurrence or used as an inline interaction. (messages No. 1 to 3.2)
    c. Within the loop, there is the message getNextJourney to the CarSharer.
    d. Finally, there are the messages concerned with communicating with the Accounts system

3. Replace alternatives, options, loops and parallel structures with the appropriate notation from activity diagrams
    a. First draw a frame for the diagram, then an initial node.
    b. Include the first set of messages (messages No. 1 to 3.2) as an inline interaction within its own frame to a simple sequence diagram called "Get UI Data"
    c. The next step is to include the branch into the two alternative paths (Delete Journey and Set Journey to Defunct). Add decision and merge nodes with appropriate guard conditions.
    d. Then add in the messages to get the next journey. Convert this part of the diagram to an interaction fragments (using interaction operation - ref):
        i. "Get Next Journey" (Messages No. 4 and 4.1)
    e. There is then a loop decision point: if there are no more journeys [nextJourney == null] then carry on, otherwise loop back and deal with the next Journey [nextJourney != null].
    f. Finally, there is an interaction occurrence to deal with notifying account system. Convert this part of the diagram to an interaction fragments (using interaction operation - ref):
        i. "Notify Accounts" (Messages No. 5 to 6).
    g. End the diagram with an activity final node.

6. Interaction Overview Diagram

Convert the following Sequence Diagram for Vehicle Passes Beacon to an Interaction Overview Diagram:

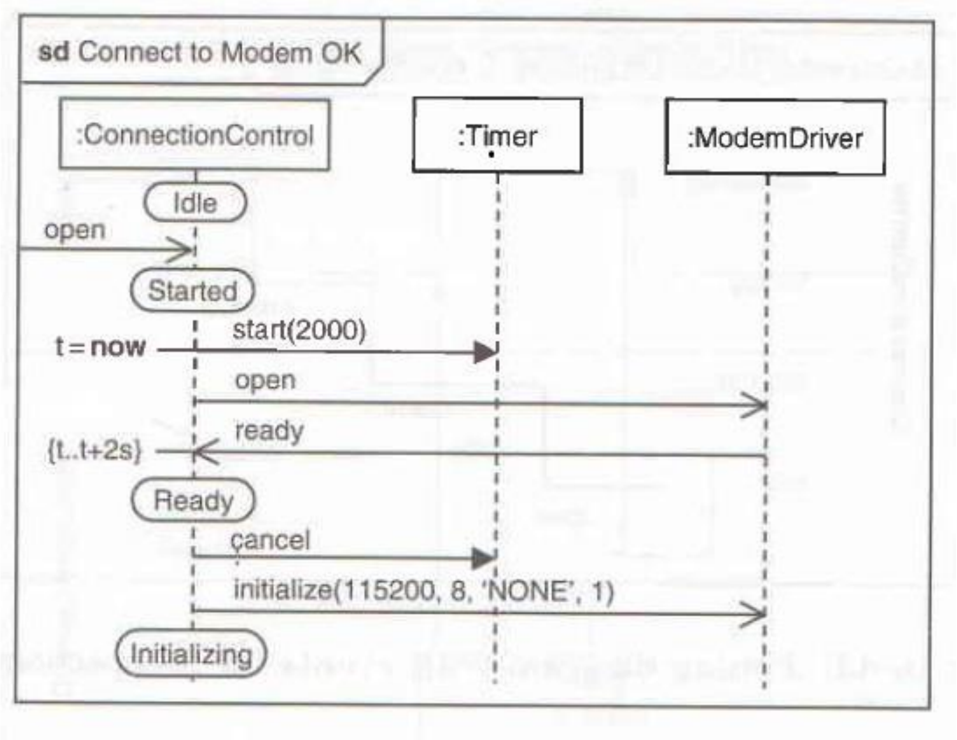


NOTES:

- Based on the tips showed in Question 4, you may apply the concept of interaction fragment, ref interaction operator to indicate part of the interactions in your diagram. Give an appropriate name for the interaction fragment.

7. **Timing Diagram**

Draw a timing diagram based on the following sequence diagram with states and timing constraints.


sd Connect to Modem OK
:ConnectionControl
:Timer
:ModemDriver
Idle
open
Started
t = now
start(2000)
open
ready
{t..t+2s}
Ready
cancel
initialize(115200, 8, 'NONE', 1)
Initializing