Homework 1

Instructions:

Hand in a copy (printout) of the UML class, UML object and UML sequence diagrams.

Hand in a copy (printout) of the .use, and .x files

Your homework is worth 10 points.

Due: 1/24 (Friday) at the beginning of class.

UML-based Specification Environment (USE) Tool

Standalone UML Diagramming and Analysis Tool

Introduction

The USE tool allows a person to model a system using UML, create instances of the system as object models and analyze structural and additional constraints on the system model and object models. Beyond this, a simple language (called Simple OCL-based Imperative Language, or SOIL), can be used in the system model to specify methods, which then allows an object model to 'run' a scenario through the system, checking constraints and structure along the way. These scenarios allow the tool to create sequence diagrams of the object interactions.

The USE tool runs on Java, and you can download it from here:
https://sourceforge.net/projects/useocl/

You need to download the latest version of the tool.

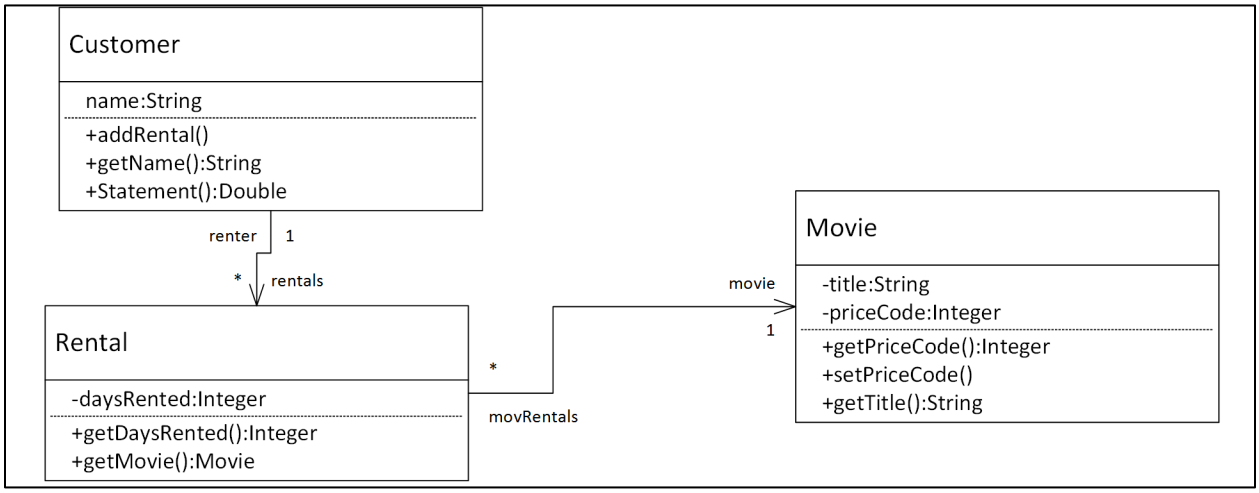
The USE tool is described here:
http://useocl.sourceforge.net/w/index.php/Main_Page

The SOIL language (Simple OCL-based Imperative Language) is described here:
http://useocl.sourceforge.net/w/index.php/SOIL

This manuscript documents how to apply the USE tool to the first design of a simple movie rental system. In the first design of this system, the main functionality is embedded in a method that prints the rental charges for a customer in a statement. This assignment is structured as follows:

1. The design class diagram for the system is shown.
2. A USE model of this design is given, and instructions for loading this model into the tool are given along with steps to explore the model and print it.
3. The model is augmented with SOIL statements to allow rudimentary method calls and exploration of the assignment of responsibilities across classes in order to realize system functionality.
4. An example object model is created by USE commands given in a script file, and instructions for executing the script and exploring the resulting object model are given.
5. Instructions for displaying the sequence diagram relating to the object model commands are given.

1. Original System Design



Movie Rental System Design 1 Class Diagram (©Robert France)

The Statement() method in the Customer class calculates and prints the customer's rental charges.

2. Movie Rental System USE Model

General Description of a USE Model

A USE model consists of a textual description of a class diagram. A design model can have both attributes and methods in it, while a requirements model does not have any methods in it. The textual description is in an ascii text file, that by convention ends with '.use'. For our example, the USE model will be in a file called 'movRentv1.use'.

The first executable line of a USE model consists of the keyword **`model`** and then the model name. Comments can be included in the file using lines that begin with 2 dashes ('`--`'), or be enclosed in '**/***' and '***/**' (e.g. '/* … */'). The model should not have any dashes in any names, such as movRent-v1, as this confuses the parser. Classes are specified using the keyword **class** followed by the class name, and separate lines with the keywords **attributes**, **operations**, and finally, **end**. Any attributes or operations are specified in lines after these keywords. The end keyword signifies the end of the class specification. Associations are specified using the keyword **association**, followed by the association name, the keyword **between**, one line that declares each end of the association, and the keyword **end**. The declaration of each end of the association contains the name of the class at that end, the required multiplicity in square brackets (e.g. **'[0..*]**') the keyword **role**, and the role name. (Thus, an association has to have three names: one for the association, and one for each end of the association.)

An example of a USE model file that has two classes, A and B, and one association between them is as follows:

```
model Example

class A
attributes
operations
end

class B
attributes
operations
end

association anAssoc between
 A[0..*] role aSide
 B[1..1] role bSide
end
```

Movie Rental USE Model file

Create a file called movRentv1.use with the following contents:

```
-- This is a USE model of the first design of a movie rental system
model MovieRental

class Customer
attributes
  name:String
operations
  addRental()
  getName()
  Statement()
end

class Rental
attributes
  daysRented:Integer
operations
  getDaysRented()
  getMovie()
end

class Movie
attributes
  title:String
  priceCode:Integer
operations
  getPriceCode()
  setPriceCode()
  getTitle()
end
```

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

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

Loading the USE model into the USE tool:

USE runs in a JVM, so you need to have Java installed on the computer you want to use. Then download the USE tool from the URL given at the beginning of this assignment, and unpack it into a directory. Navigate to the bin directory where you will see a file called *use.bat*. Run this file in a Windows machine. In a Mac, you should be able to just run the *use* executable. A command interface will open, then a Java GUI for the USE tool will open. The command window will have a 'use>' prompt. Ignore this window and interact with the Java tool GUI as follows.

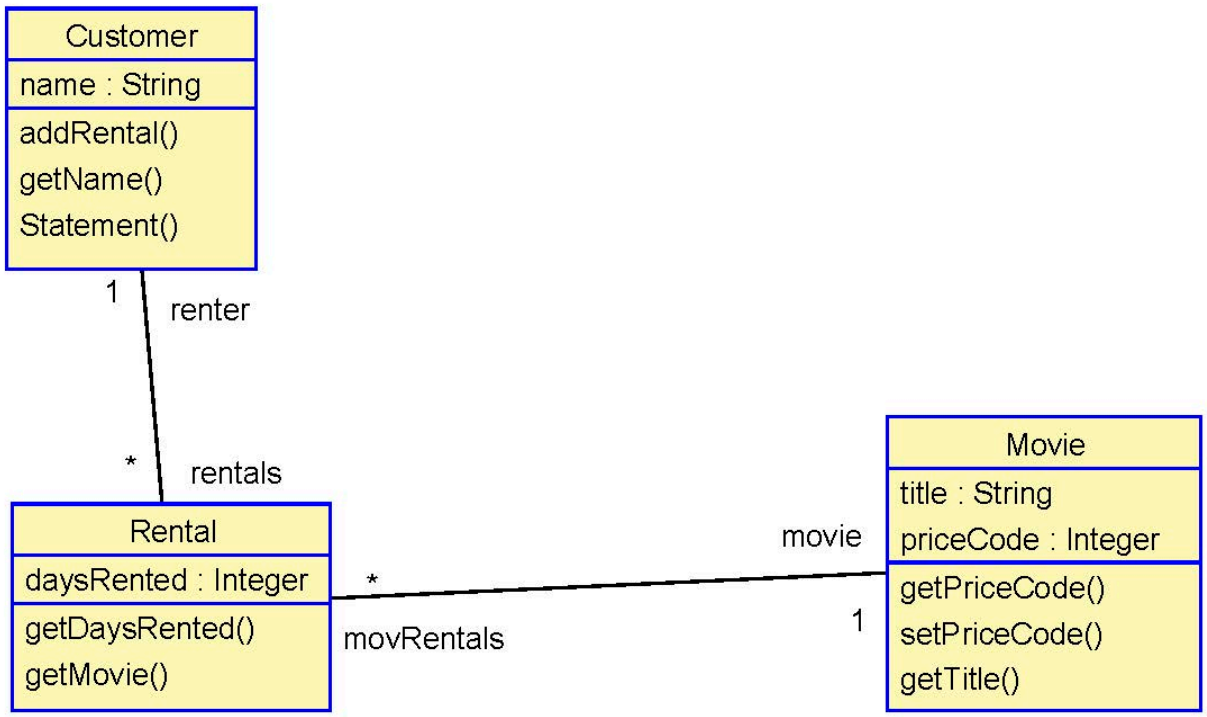
To load the USE model, click 'File, then 'Open specification'. A dialog window will open, with the current folder being the bin folder. Click the icon with a folder and an up arrow to navigate up the hierarchy, then find the folder where you stored the model file, movRentv1.use, click on this file name and select the Open button, or double click on the folder icon in front of the file name. The GUI window has several parts. On the upper left a hierarchy of the model will be displayed, showing the classes and associations. If you click the arrows in front of either of these folders you will see the classes (respectively, the associations) that are defined. If you click on the name of a class the lower left window shows the specification of the class (attributes and operations). Clicking on an association displays its definition in this window. Any errors or general information associated with loading the USE model will be displayed in the window labeled 'Log' at the bottom of the GUI window.

Viewing the Design Class Diagram:

To view this design class diagram, click on the icon just to the right of the icon labeled 'OCL'; the icon looks like a class, with 3 sections inside a rectangle. A window will be opened in the upper right of the GUI. You can enlarge it by dragging a corner, or by clicking the icon on the blue name bar that has an arrow moving to the upper right of a square. This enlarges it to the size of the encompassing window. The classes are yellow and can be moved by dragging them around the canvas. By default only the name of each class and the name of each association are shown on this diagram. Right click in the diagram to bring up a list of items that can be displayed. Unclick the option for 'Show association names', and instead choose 'Show role names', 'Show multiplicities', and 'Show attributes'. You can move labels using the left button on a mouse; the label will highlight in yellow and be enclosed by a dashed box and line to its origin as you move it. There is also an option for auto-layout. While this can help initially, it must be turned off to stop it, and often you will need to provide additional manual changes for a complex diagram. If you select a class, there are small black rectangles that are put around it that you can use to make the rectangle larger. This is often needed so that all the text shows up on a printed diagram. Once you have a layout that works, it is best to save it. Do this with the 'Save layout...' choice. This brings up another dialog box. It is best to save it in the same folder as the .use model, with the name of the .use model. The system will automatically save it as a '.clt' file.

Printing the Design Class Diagram:

You can print the class diagram from 'File' -> 'Print Diagram'. This brings up a printer command window where you can choose the printer, including a PDF printer. The class diagram is shown below.



Class diagram for movie rental system, printed from USE tool to PDF printer, then cropped, saved as a jpg, and inserted.

3. Augmenting the USE Model

Adding SOIL statements to further define the class methods can extend the initial USE model. This is useful to explore the responsibility structure across the classes, and can also be used to show how particular functionality is spread across the system.

Methods that utilize SOIL have **begin** and **end** statements. A method can call another method as long as the called method has begin and end statements. Collections are referenced through association role names, and SOIL has constructs to iterate through a collection. From the design diagram above, the Customer class has an association to the Rental class such that one Customer object is associated with 0 or more Rental objects. It is clear from this design that the Statement() method will have to access each object in this collection to determine how many days a movie was rented. Further, each Rental object will have to access its associated Movie object to determine what movie was rented and its price code.

Change your USE model file to include the SOIL statements. See the URL in the Introduction for more details on the SOIL language.

```
-- This is a USE model of the first design of a movie rental system
model MovieRental

enum PriceCode {regular, family, newRelease}

class Customer
attributes
  name:String
operations
  addRental()

  getName()

  Statement()
    begin
        declare m:Movie;
        for ren in self.rentals do
            ren.getDaysRented();
            m := ren.getMovie();
            m.getPriceCode();
            m.getTitle();
        end
    end
end

class Rental
attributes
  daysRented:Integer
operations
  getDaysRented()
    begin
    end

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

class Movie
attributes
  title:String
  priceCode:Integer
operations
  getPriceCode()
    begin
    end

  setPriceCode()

  getTitle()
    begin
    end
end
```

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

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

Load this USE model file into the USE tool.

4. Creating an Object Model and Methods to 'Execute' the Statement() Method

An object model can either be created by entering USE commands into the Windows command window, by running a script in this window that contains USE commands, or through the GUI as described in the USE documentation. We will create a file for these commands. By convention of the USE development team, command files end in '.cmd'. However, Windows takes exception to such file names, so our command files will end in '.x'. Create a file called `mr_cmds-v1`.x that contains the following:

```
-- This file creates 1 customer, 2 movies, and 2 rentals. It sets up
--   associations between them, and finally calls the Statement() method
--   for the customer.

!create c1:Customer
!create m1:Movie
!create r1:Rental
!set m1.priceCode := PriceCode::regular
!set m1.title := 'Movie1'
!set r1.daysRented := 5
!insert (c1,r1) into custRentals
!insert (r1,m1) into movRental

!create m2:Movie
!create r2:Rental
!set m2.priceCode := PriceCode::newRelease
!set m2.title := 'Movie2'
!set r2.daysRented := 3
!insert (c1,r2) into custRentals
!insert (r2,m2) into movRental

!c1.Statement()
```

Running the Object Creation Script

Read the .x file you just created from the USE command line. You can use the command **read** <filename.x> or **open** <filename.x>. When you enter this command line, the USE tool searches for this file and executes the commands in it. Each line is echoed to the USE command window, and any output associated with commands is printed. If you need to correct errors in the file, it is important to know that some of the commands may have executed correctly, leaving some objects created in the system. To reset the system, you need to go to the GUI interface window, and click on 'State' -> 'Reset'. Click 'Yes' in the dialog box, and all objects that may have been created will be destroyed. You can then re-run the script using the read command.

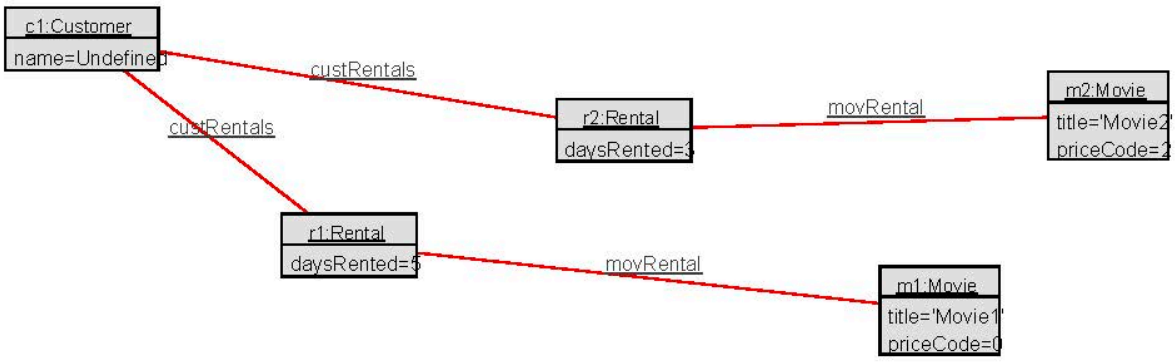
Viewing the Object Model

To view the resulting object model, click the button next to the class button you chose to view the design class diagram. This one has 3 white rectangles that are connected with lines. Similar to the class view, a small window will appear which you can make bigger, and the objects are shown as blue rectangles with their identifying names (e.g. m3) followed by their type (e.g. Movie) and their attribute values (e.g. daysRented=5). You can move the objects around by dragging them on the canvas, and if you right-click with a mouse you will see options for what is displayed. You can hide different objects, show role names instead of association names, etc. You can also save a layout (this time the system automatically creates a file ending in '.olt') and load it at a later time.

You should note that the object model shows the objects that were created from the script file (using '**!create**' commands), with attributes that were explicitly set (using '**!set**' commands), and associations that were explicitly created (using '**!insert**') commands. Finally, the '**!c1.Statement()**' command in the script causes the Statement() method in the c1 Customer object to be executed.

Printing the Object Model:

You can print the object model diagram from 'File' -> 'Print Diagram'. This brings up a printer command window where you can choose the printer, including a PDF printer. The object model is shown below.



Object model diagram for movie rental system; objects and associations created from USE commands; diagram printed from USE tool to PDF printer and then cropped and inserted.

4. Viewing the Sequence Diagram associated with the Object Model

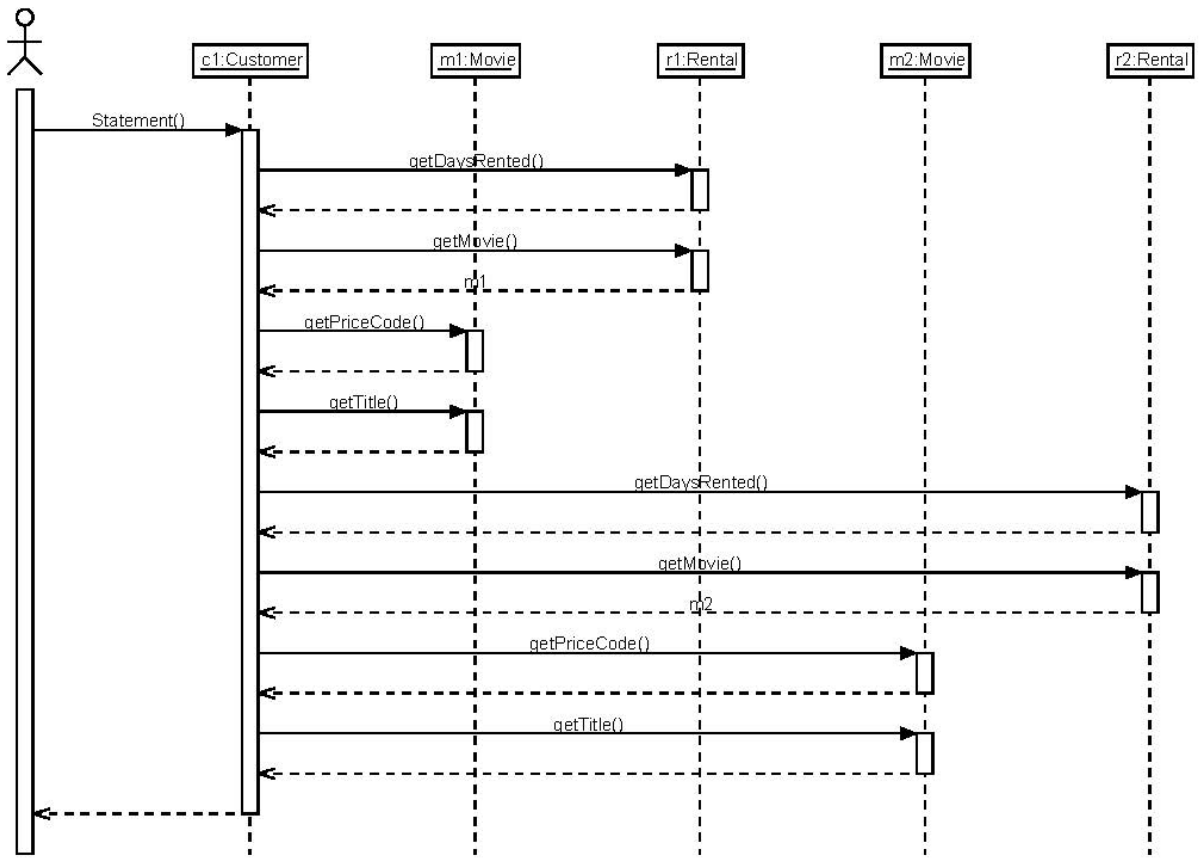
Since there was an object method executed from the script, a sequence diagram can be created that traces the SOIL-enhanced methods associated with that method.

To view the sequence diagram, click the fourth button from the right in the GUI window. The icon looks like a 2-object sequence diagram, with 2 small white rectangles at the top, each with a tall white rectangle underneath it. The long rectangles have arrows near their top and bottom, the upper one pointing to the right and the lower one pointing to the left.

A window with the sequence diagram is created, and you must enlarge it to see the full diagram. You can scroll right and left and up and down if the diagram is bigger than you can make the window.

Printing the Sequence Diagram:

You can print the sequence diagram from 'File' -> 'Print Diagram'. This brings up a printer command window where you can choose the printer, including a PDF printer. The sequence diagram is shown below.



Sequence diagram for object model created from USE commands; diagram printed from USE tool to PDF printer and then cropped and inserted.

A note about printing USE diagrams:

USE scales the diagrams to fit on a single page, so if they are very complex the resulting diagram is unreadable. For complex class and object diagrams you can often hide elements of the diagrams to include only necessary parts in a printed diagram. However, for sequence diagrams, you must simplify USE commands scripts to make the diagrams reasonably sized. Since command scripts are often used to explore functional responsibilities across a system, multiple scripts should be created that are each simple and explore something different regarding responsibilities. If scripts are being created to test how the system works, again, multiple simple scripts that each test a different situation should be created.

Your Homework:

Pick any Design Pattern (we studied many in ESOF 322). Draw the class diagram, generate an object diagram that is compliant with your class diagram. Now sprinkle some SOIL to exemplify the operations of your design pattern and generate the sequence diagram.