SEG2105 - Introduction to Software Engineering

Lab 3a Instructions – Class Diagrams

October 2013

Purposes:

1. To enable you to better understand how to model in UML

2. To enable you to understand how UML can be translated into Java code, and how Umple allows you to write code that includes UML constructs.

You may work individually or in groups of two. The TA will take attendance and verify that you are following the directions. It is essential to have the TA verify your work since it will count for your participation grade.

There is no lab report or assignment associated with this lab.

You will need to keep this file open since there are things you will need to copy and paste from here to a web browser or to other files.

1. Using UmpleOnline

Go to the following website
http://try.umple.org

You will see a window with three areas.

On the right is an area where you can draw UML diagrams.

In the middle are some icons or labels you can click on to issue commands.

On the left is an area that will contain code in the Umple language. The Umple code looks a lot like Java. It can contain declarations of UML entities like classes, attributes and associations. It can also contain ordinary Java methods.

Later on we will show you a fourth pane that appears below with generated code.

2. Do the following to draw a UML class diagram and make Umple code appear

a) Click on the 'class' icon, then click on the right pane in the webpage.

You should see a UML class symbol with the name NewClass.

b) Double-click on the class name and edit it so it changes from NewClass to Student

Notice that on the left hand side of the screen you see 'class Student{}'. Umple converts whatever you type as a diagram into a textual form.

c) Click in the attributes area of the Student class where the text says '--Add More --', then replace this by 'studentID: Integer'.

You just added a UML attribute.

Notice on the right side it now says 'Integer studentId;'.

It has converted UML syntax into Umple textual syntax, which is also Java syntax.

d) Add another attribute called 'name'. Don't explicitly give it a type.

Notice that the type 'String' is automatically added if you don't specify a type in the UML diagram.

e) Click on 'Options' in the middle panel, and select 'layout editor'.

You will see the 'position' information appear at the bottom of the left panel. This is Umple's way of storing the layout of the diagram separately from the model.

f) Drag the Student class around in the right hand panel.

You will see the position information being adjusted.

g) On the left hand text panel edit the word 'name' so it says 'fullName'.

Notice that the UML diagram on the right hand side changes.

3. Do the following to generate code

a) In the middle panel, make sure the 'Tools' panel is displayed, and below the 'GENERATE' label it says 'Java Code'. Then click on the button that says 'Generate Code'.

You will see a label pop up for a few seconds saying that code generation is complete.

b) Now scroll down.

You will see the java code for the student class that you had created earlier.

Notice that the attributes have been converted into private fields, and getter, setter and several other methods have been created.

4. Work with an Umple program is complete with a main method

a) Select the user manual from the link at the top of UmpleOnline, or go to http://manual.umple.org ,

b) Select the 'Hello World Examples' page, and scroll down until you see 'Another Example' with Umple code for Person, Student and Mentor.

c) Study this code.
Note how methods are embedded, and the fact that you can define a class Person twice to add additional information (called a mixin).

d) Click on the 'Load the above code into UmpleOnline' link at the very bottom.

e) Generate Java code from the example using UmpleOnline

Don't worry if a warning appears. Umple is just telling you that the main method will be passed through untouched when the program is compiled.

f) Click on the 'Download the following as a zip file' link to obtain a copy of the code.

*g) Compile the code using javac *.java*

h) Run the compiled code using java Person and observe the results

i) Modify the Student-Mentor association in Umple, so that a mentor may be the mentor of at most two students.

i) Generate the code and figure out what has changed.

k) Modify the main program to attempt to add two more students to Nick The Mentor, but print a warning if this fails.

Hint: You will have to determine the API to call to add students. You can generate Javadoc to see this.

l) Show the TA your revised Umple model and also the results above where you modified the main program.

5 Explore other Examples in UmpleOnline and the Manual

a) In Umpleonline, under the Load menu / class diagrams / select 'Airline Example'

b) Under the 'Options' menu, try out the various options.

c) Under the Tools / Generate Code menu, generate Real Time C++and review it quicly. Download it as a zip file and unzip to see the directories and individual files more quickly.

c) Generate php code and review it.

d) Look at the page on reflexive associations in the user manual, and generate Java for one of the examples.

e) Look at the page on association classes in the manual and load one of the examples.

6. Compiling Umple on the command line

a) Go to the manual page on sorted associations.

b) Click on 'View Plain' just above the example.

c) Copy the text from the plain view into a text editor of your choice and save the resulting file as Sorted.ump

d) Obtain the command line compiler for Umple by clicking going to http://dl.umple.org and clicking on the link that says 'The latest build of the command-line compiler at the University of Ottawa'

e) Compile your code as follows

java –jar umple.jar Sorted.ump -c -
(The -c - says to compile the Java once it has been generated)

f) Run the program as follows and show your TA.

java Academy

6. Editing a more substantive UML Model

a) Start up UmpleOnline using the following URL, which uses automatic diagram layout
http://try.umple.org?diagramtype=GvClass

b) Copy and paste the following code into the UmpleOnline version that you just started. (you can also copy this from http://www.site.uottawa.ca/~tcl/seg2105/coursenotes/Grocery.ump)

```
// Model of the grocery store problem as found in the textbook on E90 b (p185)
// and in the course notes, chapter 5, p39
// This version is one of the versions we created on the board on Sept 27th

class Store {
  id;
}

class Item {
  // If true, then the price per kg; otherwise price per item
  Boolean byWeight;
  Boolean taxable;

  // Price in cents; can be overridden by special prices
  Integer defaultPriceInCents;

  * -- 0..1 Store stockedIn;
}

// Price offered for members only
class SpecialPrice {
  Integer priceInCents;
}

association {
  1 Item -- * SpecialPrice;
  1 Item -- * GroupPrice;
}

// Type of membership card such as AirMiles
class MembershipCardType {
  name;
}

// Membership card belonging to a particular person
class MembershipCard {
  Integer number;
  * -- 1 MembershipCardType;
}

association {
  * SpecialPrice -- 1 MembershipCardType;
}

// Purchase of a certain number of a particular item
class Purchase {
  numberOfItems;
  * -- 1 Item;
  * -- 0..1 MembershipCard;
}

// Price for bulk purchases
class GroupPrice {
  Integer priceInCents;
  Integer numberToBuyToGetPrice;
}
```

c) Add a main java method that does the following (you might choose to do this in a text editor first):

i. Create an instance of Store
ii. Create three instances of item
iii. Creates a MembershipCardType with the name 'specialMiles'
iv. Creates a MembershipCard number 12233 for the above
v. Creates a special price for one of the items
vi. Creates a group price for one of the items
vii. Creates a purchase for one of the items
viii. Prints out the items, special price, group price, membership cards, and purchases to standard output

d) Compile, execute and run your code, and debug as needed

e) Modify your model from part c so that the alternative design we discussed in class is implemented: i.e. merge the SpecialPrice and GroupPrice class so that we have the following single class in place of the two classes.

```
class SpecialPrice {
  Integer priceInCents;
  Integer numberToBuyToGetPrice;
}
```

You will also need to adjust the main program to match the new design. Remember we said that the numberToBuyToGetPrice could be 1 if there is a membership special price.

f) Compile your code, get it working and show the TA the output.

You may leave when you have finished this exercise, or when three hours are up. If you do not finish part d, try to get it done on your own time. But the results will not be handed in. Students who get it finished in class time and show the TA will get slightly better participation grades.

Next week in the lab we will do exercises about state diagrams