

COMP2801

2004

Software
Engineering
Studio

Prac week 6

Semester 2, 2004

School of Information Technology
and Electrical Engineering
The University of Queensland

Goals:

1. Using Rational Rose (as an example UML tool)
2. Reverse engineering a simple program
3. Creating a new class
4. Creating an interface
5. Creating a class that implements the interface
6. Creating a class that references an instance of the interface
7. Adding implementation notes to the diagram
8. Generating code

Note:

All exercises are to be performed using your user account.

Object Oriented Design (OOD)

Using OOD, we can describe an application in terms of interfaces rather than an implementation, and those interfaces can become software entities that neither depend upon the software that uses them, nor the software that implements them. This property of OOD allows us to create systems that support more flexible and robust designs. In OOD, software engineers still employ the concepts of information hiding, coupling and cohesion, but apply them in different ways than in structured design.

1. Rational Rose Enterprise Edition

These exercises use the Rational Rose suite of tools to manipulate UML diagrams.

What is Rational Rose?

Rational Rose is a suite of tools that allows software developers to model the software they are working on in a visual environment, using industry standard notations. In particular it provides strong support for the use of UML at all phases of the software lifecycle. We will only be examining the use of UML for Object Design. We will be examining the use of UML to create Design Models. We will use UML to document the classes and interfaces we are designing, as well as how the methods and attributes of the classes can be used in the design.

2. Exercise – Reverse engineering a simple program

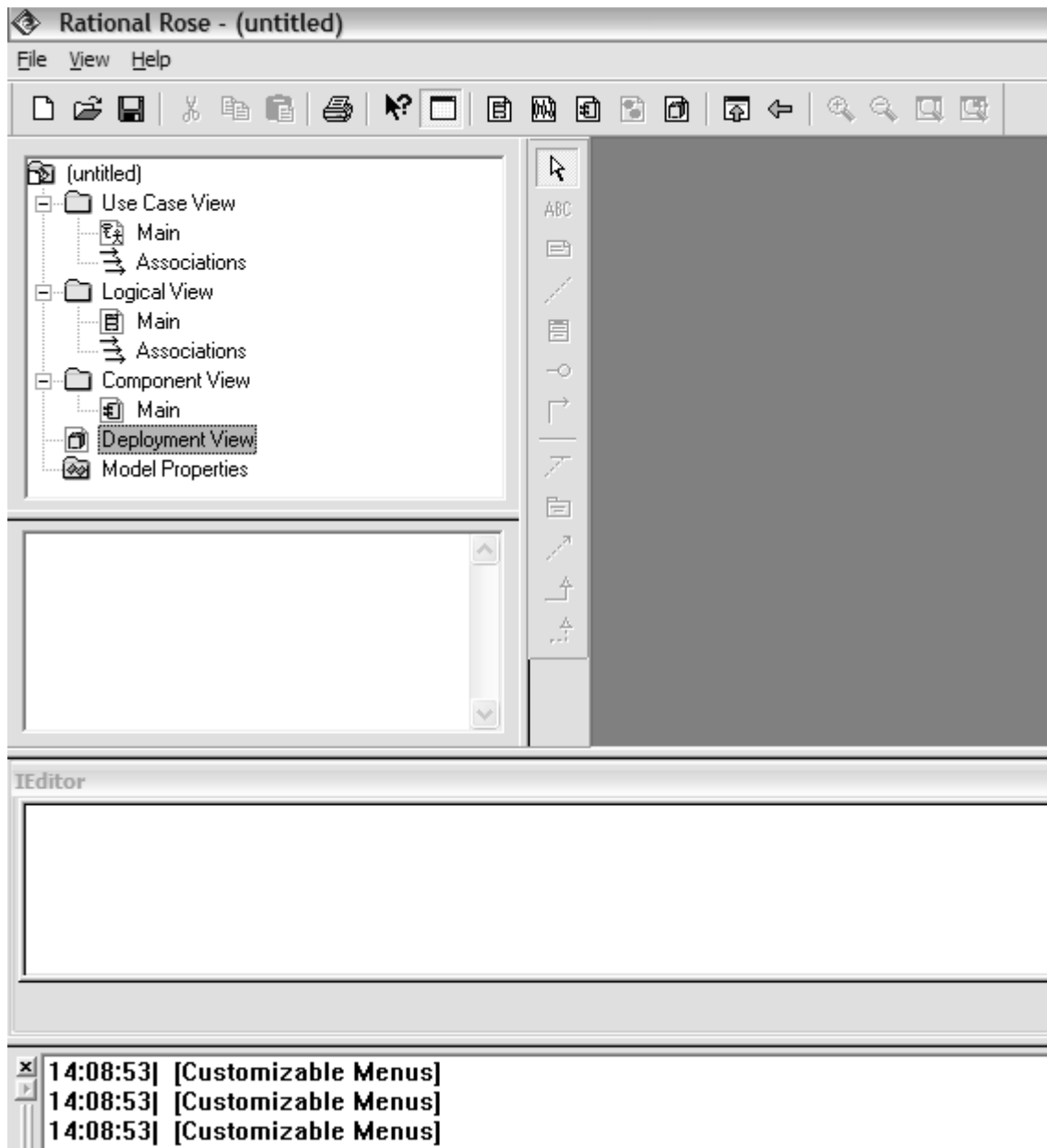
Copy the uml6.jar file from the comp2801 account and place it in a newly created directory. This is a modified form of the team.jar from earlier prac exercises.

```
mkdir wk6
```

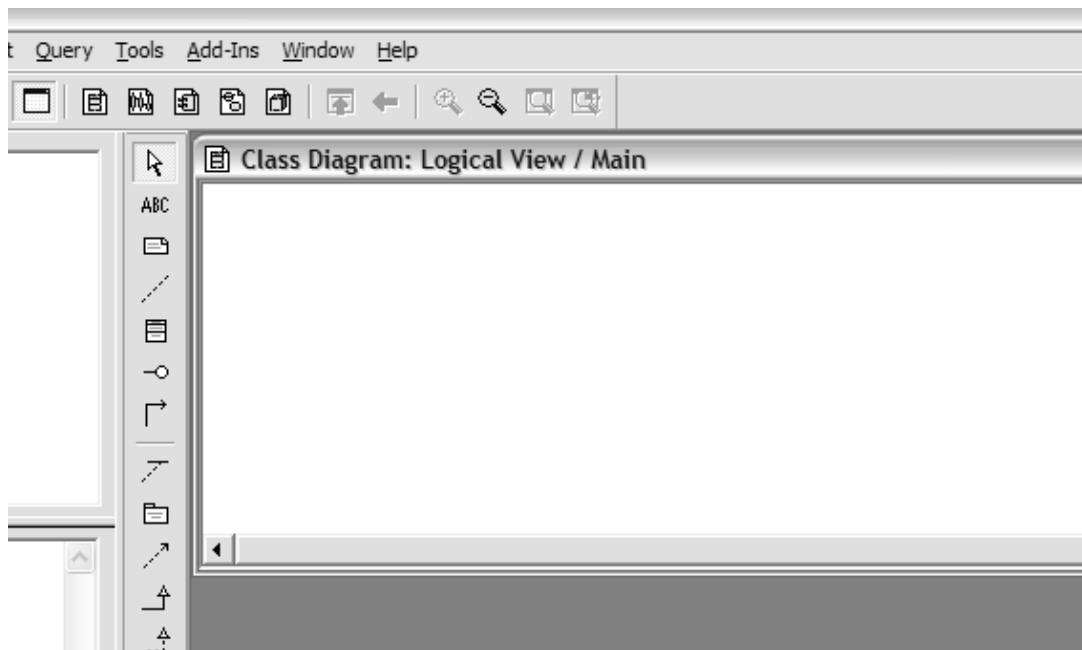
```
cp ~comp2801/prac6/uml6.jar wk6
```

In windows (not Unix), start Rational Rose Enterprise Edition from the start programs menu.

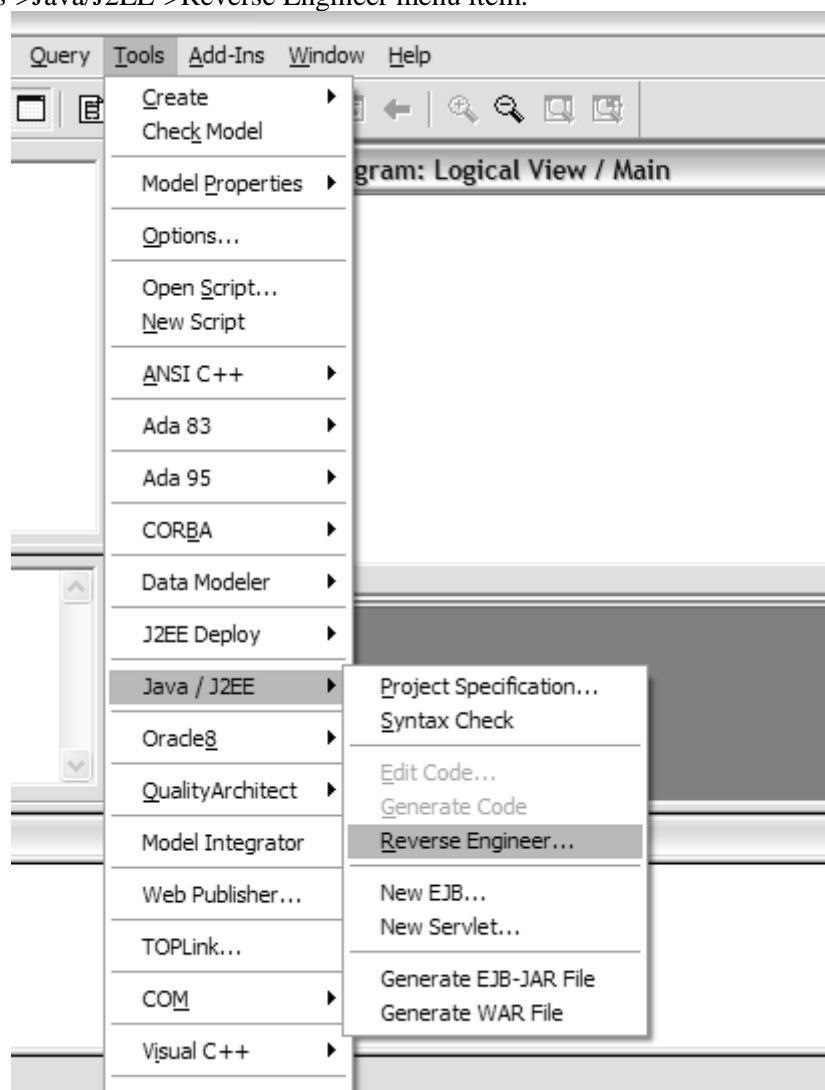
Create a blank new project.



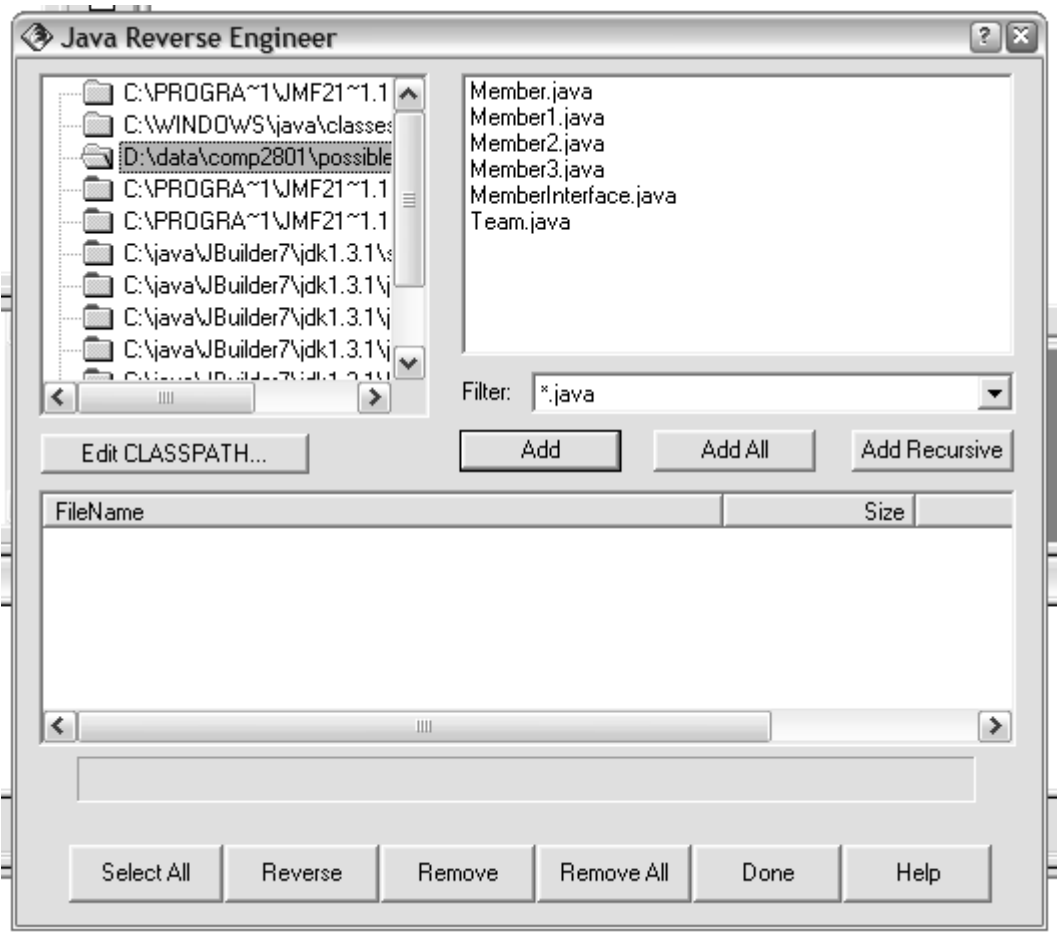
Next, double click on the Logical View / Main diagram icon. This should display:



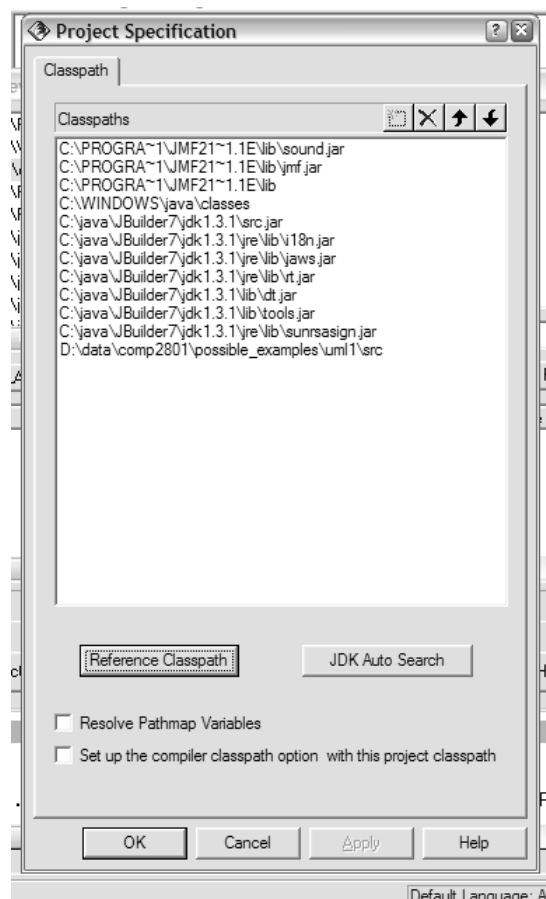
This is required, due to the somewhat strange menu behaviour of Rational Rose. To get the full list of menus at the top of the page requires a Diagram window to be activated. Next under the Tools menu, select the Tools->Java/J2EE->Reverse Engineer menu item.



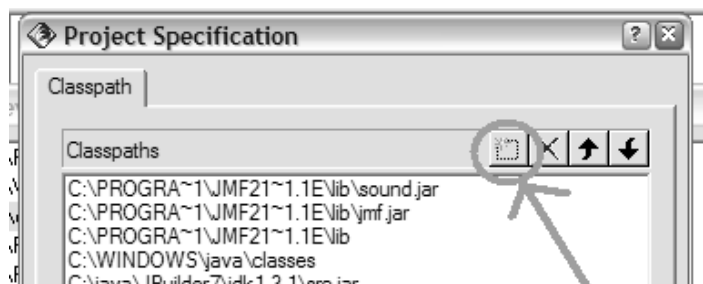
This will display the following dialog:



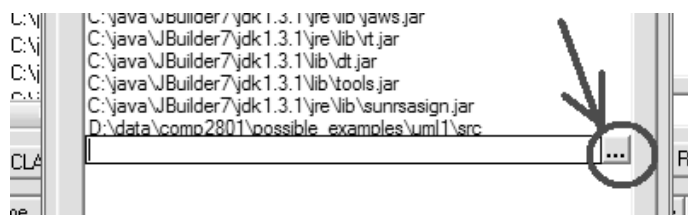
To import our java classes for the uml1.jar file we have just downloaded, we have to add the uml1.jar file to Rational's CLASSPATH. Select "Edit CLASSPATH..."



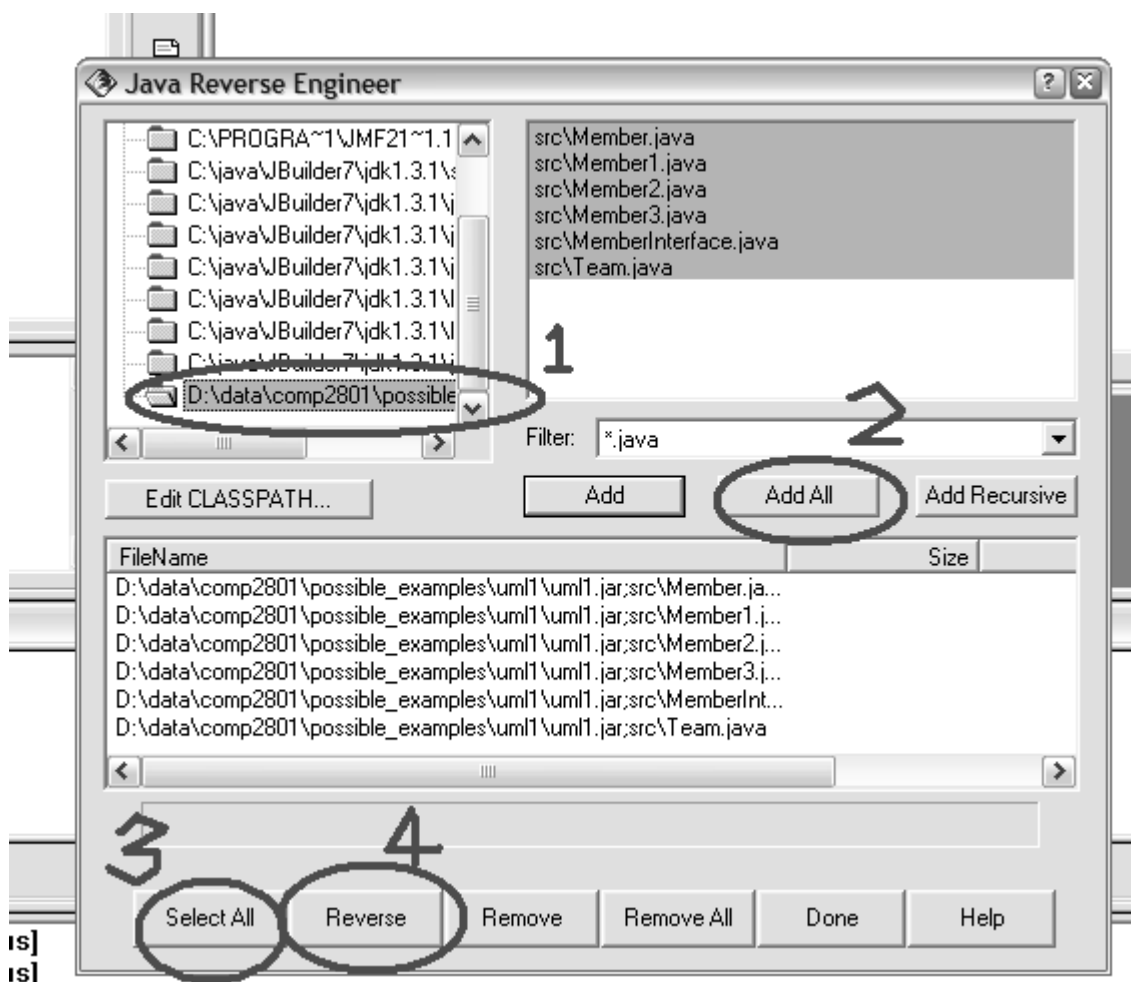
Add the class path by first selecting the new item box:



This will create a box where the user may type the class path or use the “...” button to add the class path graphically.



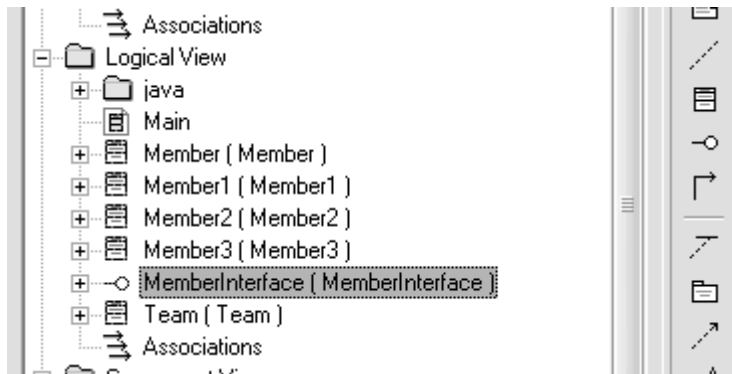
Add the uml6.jar file to the classpath and return to the dialog in the Java Reverse Engineer window.



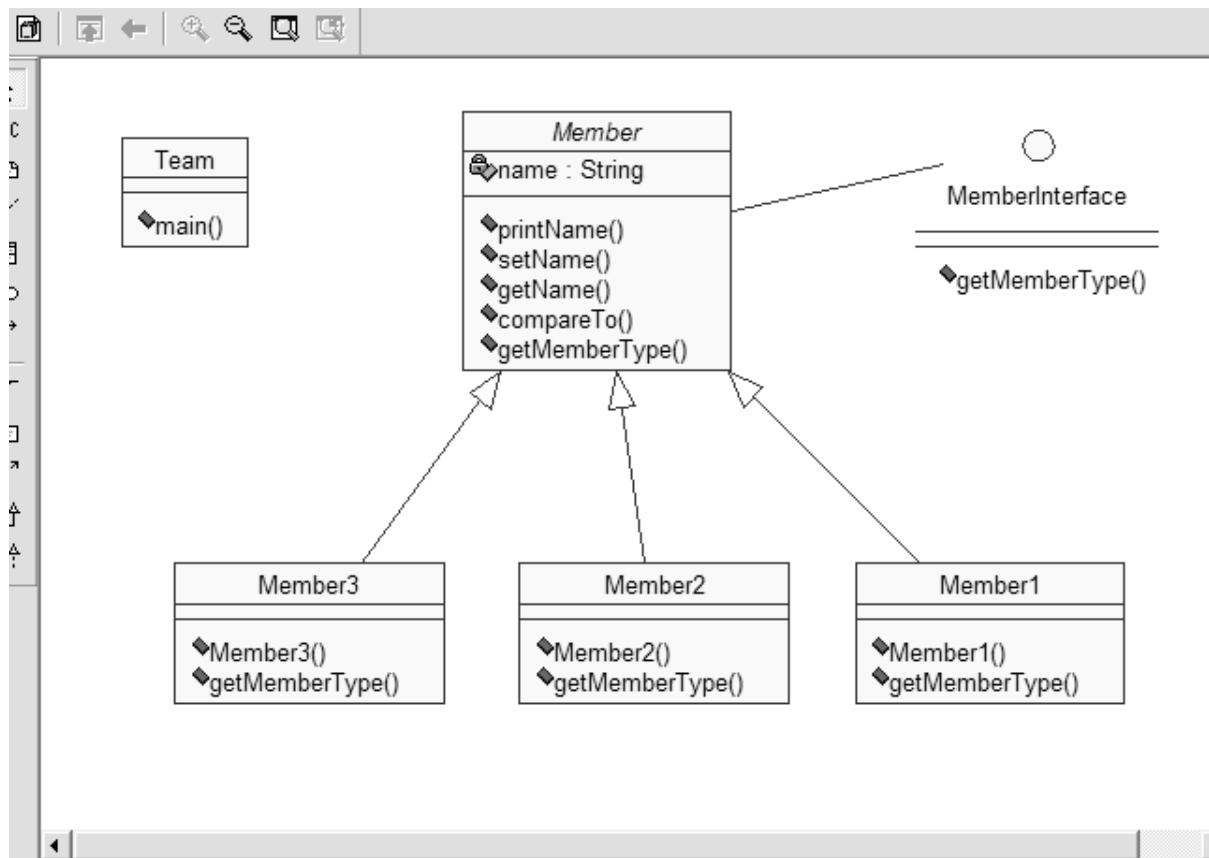
To add the .java files in the uml1.jar file you added to the classpath, you have to select the items in order indicated in the screen shot above.

- 1) select the .jar file
- 2) Add All
- 3) Select All
- 4) Reverse

The Logical View should now contain the imported java classes/interfaces:



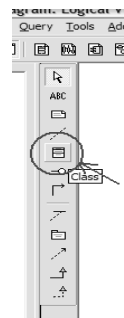
Drag the Member, Member1, Member2, Member3, MemberInterface and Team items onto the Main diagram. You should see something like this:



Look also at the source code. See how the Class, Abstract Class and Interface definitions have been represented in the diagram as UML.

3. Exercise – creating a new class

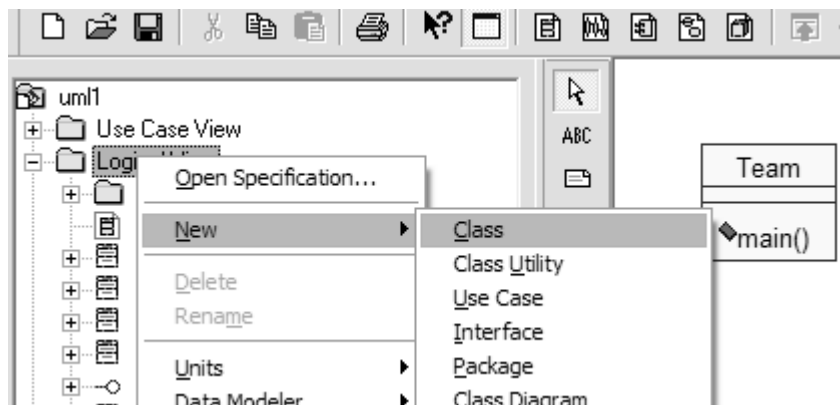
Now that we have imported some existing classes, let us add a new class to the model. NOTE: the term “Model” is used to describe the complete design. The full name is the Design Model. In the Logical view, select the class icon from the toolbox, and in the Main Class Diagram, create a new class by clicking in the diagram after selecting the Class item on the toolbar.



Define the class to have the following attributes/properties:

Class Name: InterestsImp1				
Method Name	Public Private Static	Description	Parameters	Return Type
getInterests	Public	Get a list of interest names		Array of String
addInterest	Public	Add an interest to the set of interests	String – the name of the interest, i.e., Soccer	Void
removeInterest	Public	Remove an interest from the set if the name exists	String – the name of the interest to remove	Void
hasInterest	Public	Return true if already has the given interest	String – the name of the interest to query for	boolean
getInterestCount	Private	Get a count of the number of interests		integer
Attribute Name	Public Private Static	Description	Type	
InterestList	Private	A list of interest Names	Array of String	

Now define a class with the same definition as InterestImp1 but called InterestImp2, but this time do not create it on the diagram but using the list on left hand side.



4. Exercise – creating an interface

Create the following interface:

Interface Name : IInterests				
Method Name	Public Private Static	Description	Parameters	Return Type
getInterests	Public	Get a list of interest names		String[]
addInterest	Public	Add an interest to the set of interests	String – the name of the interest, i.e., Soccer	Void
removeInterest	Public	Remove an interest from the set if the name doesn't exist	String – the name of the interest to remove	Void
hasInterest	Public	Return true if already has the given interest	String – the name of the interest to query for	

5. Exercise – creating a class that implements the interface

Have the classes InterestsImp1 and InterestImp2 implement the interface IInterests.

6. Exercise – creating a class that references an instance of the interface

Have the classes Member1 and Member2 use an instance of InterestsImp1 and have class Member3 use an instance of InterestsImp2.

7. Exercise – adding implementation notes to the diagram

Add two notes to the diagram, and associate one note to the Member1 class and the other with the Team class.

8. Exercise – Generating Code

Generate the code for the complete model and show it to the tutor.

Homework Exercise – Create a UML Class diagram for your program

Create a new Rational Project and add import (reverse engineer) one or more UML diagrams for your program.