DT228/3 Software Engineering Lab 3 (Week 4)

Rational Software Architect

Class Diagrams, Sequence Diagrams, Model Implementation

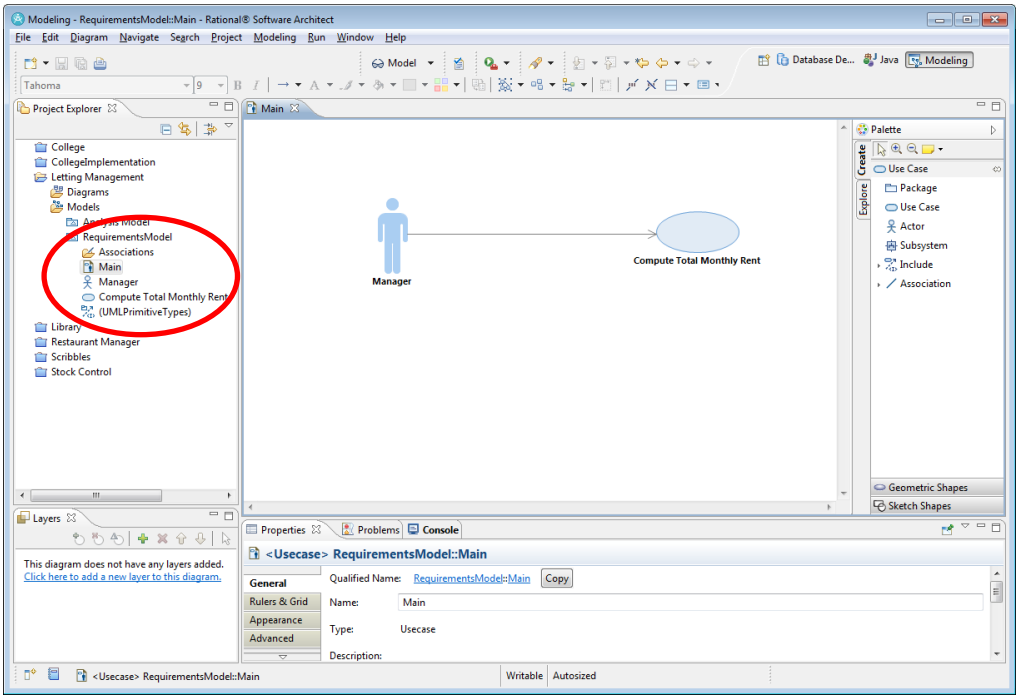
In this lab sheet, we will create a new UML project using RSA, create a simple use case diagram, create a class and sequence diagram which will consider the *Principle of Least Knowledge*. We will then, using RSA, create a *Java Project* and add code to realise the use case.

The example we will do is taken from the class notes. It is a *Letting Company* management system and we will model and implement a simple use case to calculate the monthly rent for a given property which contains a number of apartments.

Task 1 – Create a new UML project and Requirements Model

As before create a new *UML Model Project* and name it *Letting Management* - choose a *Blank Use Case Package* as the template (recall: depending on the version of RSA you are using you may not see *Blank Use Case Package* - in that case, choose *Blank (UML) Model* and choose *Use Case Diagram* as the default diagram). Call the model *Requirements Model*.

Add an actor called *Manager* and a use case called *Calculate Total Monthly Rent* to your model / diagram so that you have the following:

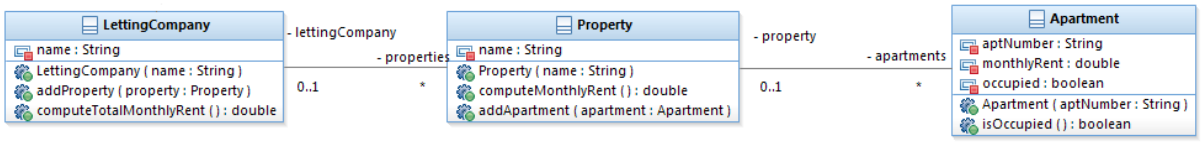


Task 2 – Create the Object Model

Consider the object model / class diagram below and its associations. Follow the instructions below.

- Add an analysis model to your project (right-click on *Models* in the Project Explorer view and choose *Create Model* – use the *Blank Analysis Package* template). Recall, use *Blank (UML) Model* with a default diagram of *Class Diagram* if those are the options you see.
- Add to the default model / diagram as appropriate to create the following class diagram. Ensure all the details are accurate.

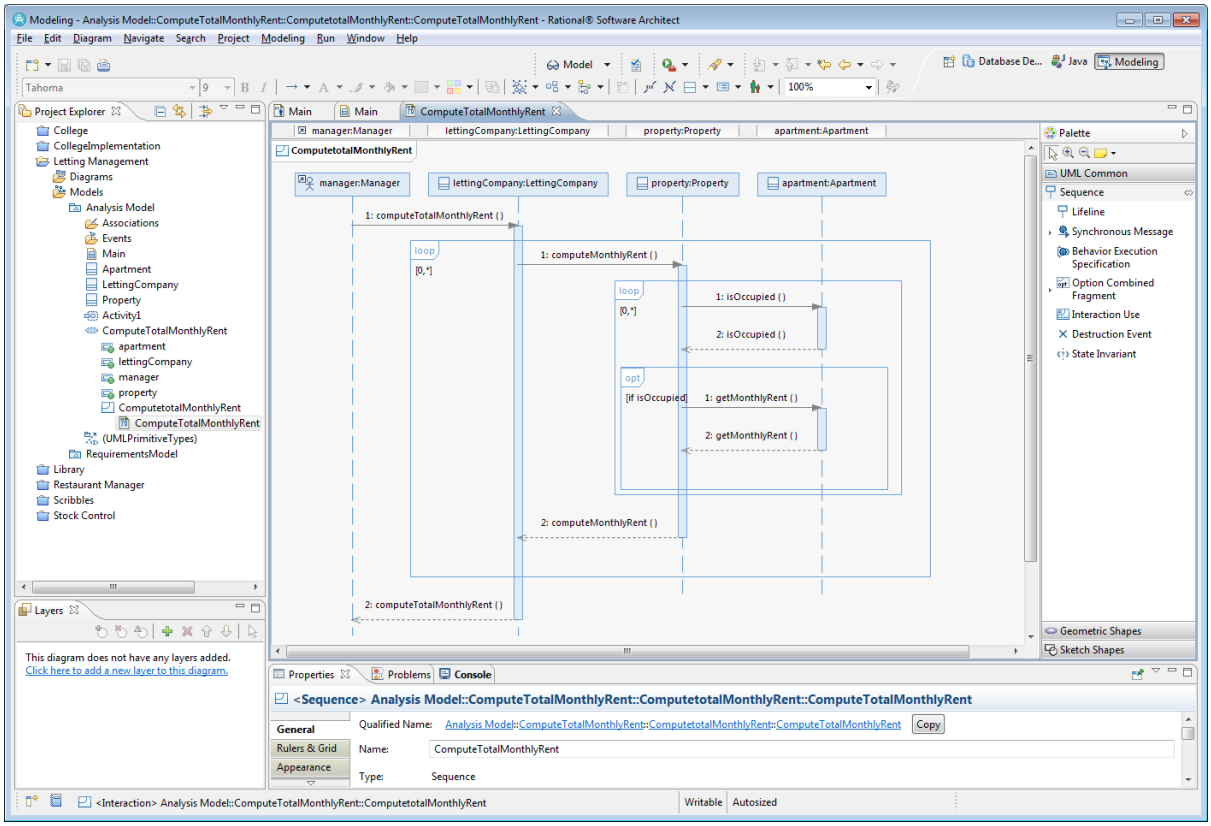
> ***Note***
> - Recall last week's lab sheet *Part II Task 1* shows how to create operations and show their signature.
> - Also, some of the operations show a *double* type – if you do not see a *double* primitive type when selecting the type just use *Integer* for now.



Task 3 – Create the Collaboration and Sequence Diagram

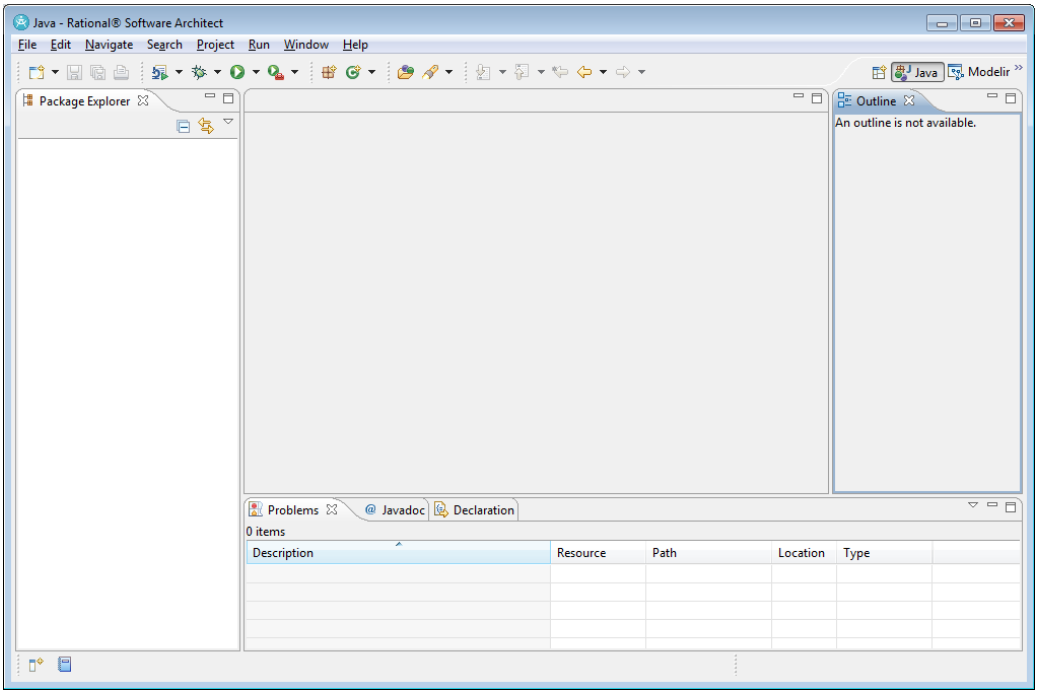
Here we add a *Collaboration* to our Analysis Model. At this point we want to show how the objects we have identified in task 2 will collaborate to realise the *Compute Total Monthly Rent* use case.

- Add a collaboration to the Analysis Model (right-click on the analysis model in project explorer and choose *Add UML -> Collaboration)* and name it *ComputeTotalMonthlyRent*.
- Add a sequence diagram to the collaboration (right-click on the collaboration and choose *Add Diagram -> Sequence Diagram*).
- Add to the sequence diagram so that you have something like the diagram below (*note the conformance to the class diagram with respect to the paths of communications between the objects*).



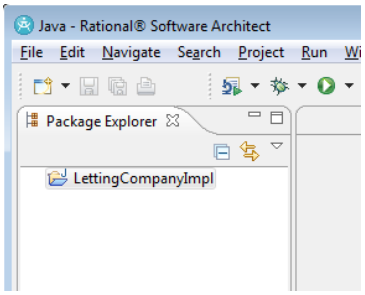
Task 4 – Create a Java Project

- Switch RSA to the Java Perspective: Choose *Window->Open Perspective->Other-> Java*
- You should see the following perspective.



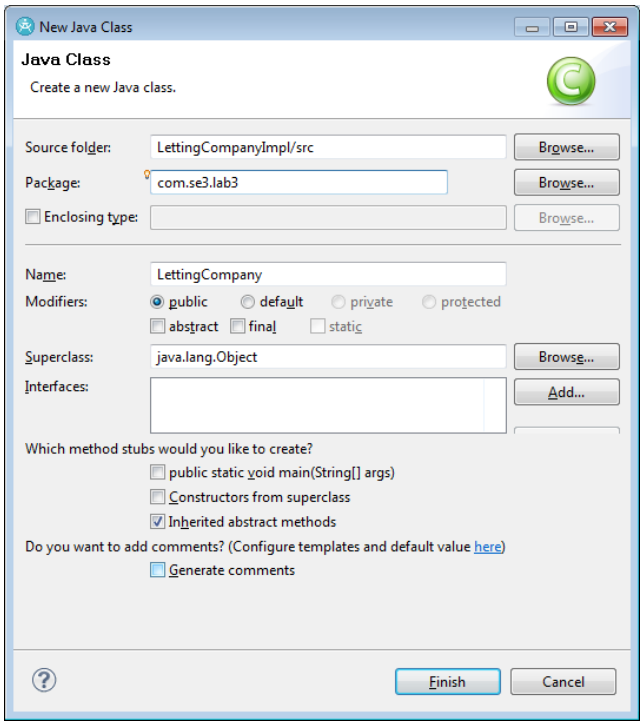
- Choose *File->New->Project->Java Project* and click *Next*.

- Name the project *LettingCompanyImpl* and click *Finish*. You should have a new java project in your Package Explorer.

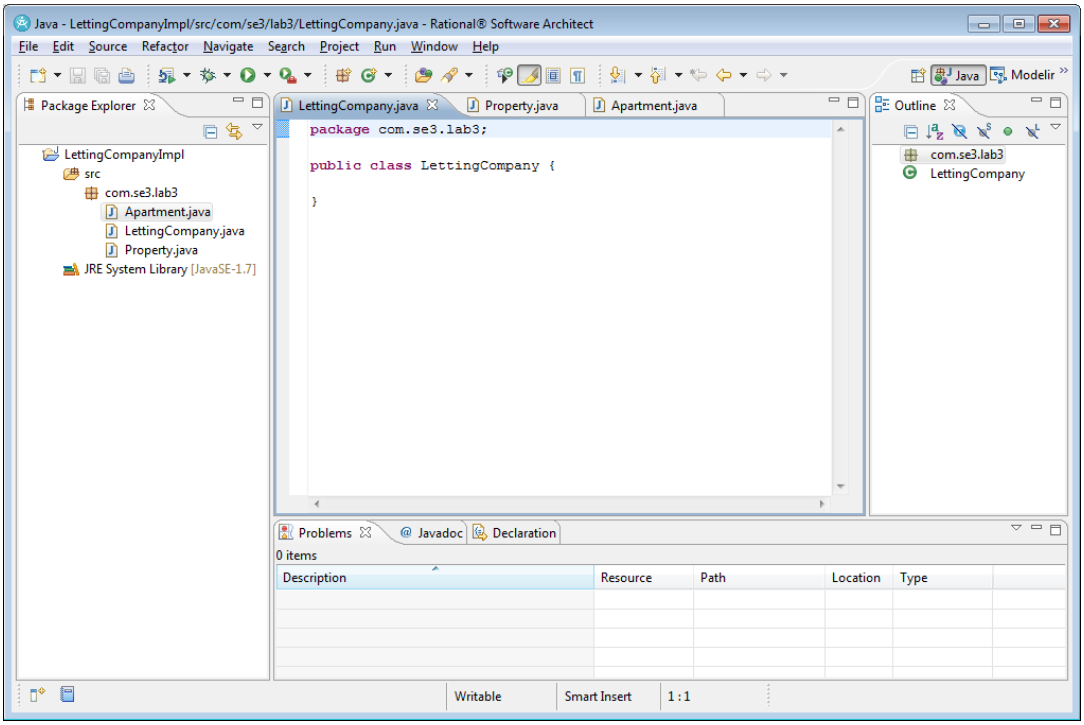


Task 5 – Create the Java Classes based on our Analysis Model

- Expand the LettingCompanyImpl project folder and you should see a source folder called *src*. Right-click on the *src* folder and choose *New->Class*.
- Set up the class as follows (note the package name) and click *Finish*:



- Create classes for *Property* and *Apartment*, you should now have the following:



Task 6 – Implement the Structural (class diagram including the Associations) and Behavioural (sequence diagram) models

- ***Note 1:*** To keep this example simpler, we will disregard the requirement for referential integrity when creating/removing object links at runtime.
- ***Note 2:*** Associations are implemented through class attributes - for a many ("*") relationship, you can use a *List*. Also, remember *List* is an interface, so you could use an *ArrayList* as the concrete implementation.

```
List<Property> properties = new ArrayList<Property>();
```

- ***Note 3:*** In the *LettingCompany* and *Property* classes, you will need to ensure that the *properties* and *apartments* attributes are initialised before you attempt to use them (this is also illustrated in note 2 above).
- ***Note 4:*** In the *LettingCompany* and *Property* classes, you will need to add code to the *addProperty()* and *addApartment()* methods respectively in order to add the *Property* and *Apartment* objects (passed in as arguments) to the *Lists*.
- ***Note 5:*** You will need to implement the loops etc. in the *LettingCompany* and *Property* classes.

Task 7 – Test the code

- Right-click on the *LettingManagementImpl* java project in the *Package Explorer* and create a new ***source folder*** called *test*.
- Right-click on the *test* folder and create a new java class called *TestClient.java* (when doing this choose to include a static *main* method). Ensure you use the same package name as in the *src* folder.

- In the *main* method, create six *Apartment* objects (name them differently and set the *occupied* and *rent* attributes as you like).

  E.g.

```
Apartment ap1 = new Apartment ("1A");
ap1.setMonthlyRent(400);
```

- Create two *Property* objects and add three of the *Apartment* objects to each.
- Create a *LettingCompany* object and add the two *Property* objects to it.
- Finish your test code in the *TestClient.java* class to test the use case realisation – i.e. invoke the *computeTotalMonthlyRent()* method on the *LettingCompany* object and verify the result is as expected.
- Execute your test (right-click on *TestClient.java -> Run As -> Java Application*).