Tools Exercise 1
02161 Software Engineering 1

Anne Haxthausen

February 1, 2008

1 Introduction

The purpose of this exercise is to get acquainted with how to use eUML and JUnit inside Eclipse.

For this purpose you should follow some of the steps in the eUML tutorial "Getting Started with Class Diagram" found on

`http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/`

Note that there might be small differences (e.g. in the exact choices in pop-up menus) in the version used in tutorial and the version installed in the databar.

Select an example that you want to model in UML and implement and test in Java.

2 Starting Eclipse and creating a class diagram editor

1. Start the Eclipse tool. In Linux use the command `eclipse &`

   Here you will be asked to fill out a workspace (that is the directory where your files will be stored by Eclipse).

2. Create a Java project: see step 1 in "Class Diagram Creation" in the eUML tutorial.

3. Now you should ensure that you use java 5.0: Select the project you have just created. Right mouse click and select **Properties**. In the pop up window, select **java compiler**, tick **enable project specific settings**, chose compiler level 5.0 and click **ok**.

4. Create a package: see step 2 in the tutorial.

5. Create a class diagram: see step 3 in the tutorial (answer no to the question about splitting the pane).

3 Making a class diagram and a Java implementation

Now you should make a class diagram and corresponding implementation in Java. You should follow the steps below, but they need not to be done in the order they are mentioned. You should note that as soon as you have made a piece of diagram a corresponding piece of Java code will be generated automatically and vice versa. In the left pane of the window, you can see the created classes (*.java* files) and diagrams (`.ucd` files). You can switch between editors for these these by clicking on them.

1. Add your classes as explained in "Classes and Interfaces" in the eUML tutorial. For abstract classes, remember to make a tick indicating this in the dialogue window that pops up.

2. Add attributes in your classes, as explained in "Attributes".

3. Add methods in your classes, as explained in "Methods". (Note that get and set methods for the attributes are automatically created, but not shown on the diagrams unless this is specially requested.)

4. Add generalization arrows, as explained in "Implementation and inheritance".

5. Add associations as explained in "Associations". For each association end fill out (in the dialogue box that pops up) the multiplicity and navigability. A tick in the navigable box means that there should be an arrow head in that end. Note that the tool requires at least one arrow head and that it does not show the multiplicity in an end where there is no arrow head. In week 2 you will learn about associations with two arrow heads. In week 1 we will just use one arrow head.

6. Complete/modify the generated Java classes. Some hints:

   - Any occurrence of `Collection` should be replaced with `Collection<...>`, where ... is the type of elements in the collection.
   - Fill out empty function bodies.
   - Remember to save your changes.

4 JUnit

You can make test cases and run these using JUnit inside Eclipse.

You should do this for your example. Note that the test cases can be defined before the implementation is completed (and hence used as a specification of the required behaviour of the methods that are going to be implemented).
Making a test case:

1. Select the package you chose to work with in the beginning of this exercise. Then right click and select **new** and then select **JUnit test case**. In the pop up window:
   (a) Fill in the name of the test case you are going to create.
   (b) Tick **New JUnit 3 test**.
   (c) In this exercise there is no need to fill out the class under test.
   (d) Tick any methods stubs (like setup) that you might like to have.
   (e) If it says that JUnit 3 is not on the path, click on **click here** and then click **ok**.
   (f) Click **Finish**. By this, a skeleton for the test case now appears.
2. Edit the body of test class, i.e. define a method for each thing you want to test. In these methods you can use assertEquals to compare objects.

Running the test:

1. Click on **Run** in the top panel. In the left pane a green color indicates that the test went through without errors, while a red color indicates errors.

5 Running a Java Program

If you have made a class having a `main` method, you can run this by selecting it and clicking on **Run** in the top panel. Output will appear in the bottom pane.

Exercise: make and run a class with a main method manipulating some objects using classes from your example.