Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III
Prof. Dr. A. B. Cremers

Aspect-Oriented Software Development
Course in the summer semester 2009
Dr. Günter Kniesel, Daniel Speicher

Exercise Sheet 5

Due: Sunday 24.05.2009, 23:59:59 via SVN

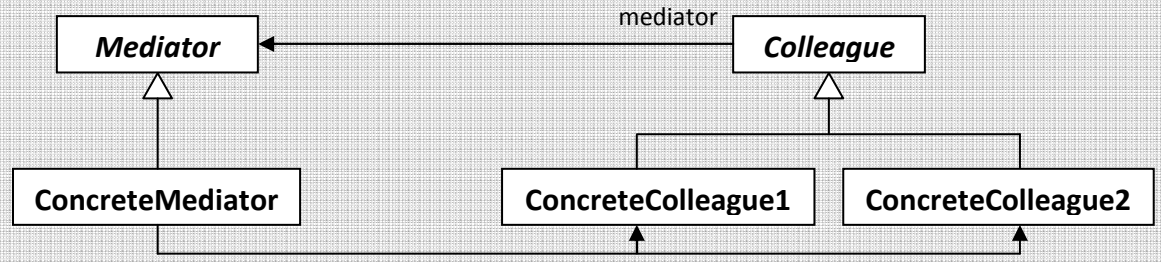
For help, contact aosd-staff@lists.iai.uni-bonn.de (staff only) or aosd-course@lists.iai.uni-bonn.de (staff and participants).

Please start working on the exercises early enough so that you can contact us in time in case of problems. Don't expect us to be available during weekend!

Exercise 1: *"Mediator Pattern, Abstractness-Stability-Diagram"* (6 Points)

Mediator Pattern

*The **Mediator** pattern is a behavioural pattern. Interaction between different colleague objects gets encapsulated in a mediator object. The Mediator defines all possible interactions between the colleagues. A ConcreteMediator then implements these interactions between the ConcreteColleagues. If a ConcreteCollegue wants to communicate with another ConcreteCollegue they will not communicate directly but through the ConcreteMediator. So there is no coupling in between the ConcreteColleagues but only between the ConcreteColleagues and the ConcreteMediator.*



a) Draw an Abstractness-Stability-Diagram for the mediator pattern. Is the position of every class unambiguous? Which assumptions do you have to make?
b) How could an aspect oriented version of this pattern look like? Draw a new Abstractness-Stability-Diagram for the aspect oriented version.
c) Find an example where the usage of the mediator pattern is useful. Identify which part of your example correspondents to the classes of the pattern.

Exercise 2: *"Observer Pattern"* (4 Points)

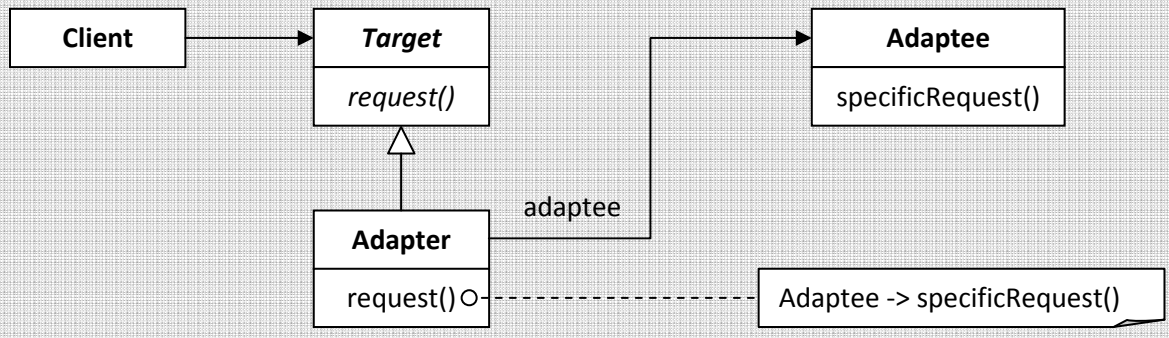
In your repository you will find the project ES05_E02_TicTacToe. The implementation is using the Model-View-Controller architecture. The communication of the model with the views should now be implemented using the aspect oriented version of the Observer pattern (2 Tier Design).

a) Identify and name all classes that include instances of the Observer pattern.
b) ***Refactor*** the project so that it uses an aspect-oriented version of the Observer pattern for the communication between the model and the view.
*(Refactoring means using all **4** steps of "Refactoring to Aspects" in every detail!)*

Exercise 3: *"Adapter Pattern"* (5 Points)

Adapter Pattern

The Adapter pattern is a structure pattern. It enables classes with incompatible interfaces to work together. The adapter provides a usable interface for a client and maps the input to the interface of the target class.



The project ES05_E03_StatusRequest in your repository is the implementation of three sensors, one for pressure, one for radiation and one for temperature. These three sensors don't have a common supertype (except Object) nor are they in any kind of hierarchy.
The values of each sensor can be grouped into three status groups:

- OK
- Critical
- Danger

a) Your job is to implement a status requester (see SensorStatusProvider) that retrieves the values of the sensors and then returns a status (Do you remember the intertype-declarations?).
The status is calculated using different values for each sensor:

	Temperature	Pressure	Radiation
OK	< 235°C	< 5 bar	< 3 rad
Critical	235° - 300°C	5 - 6.58 bar	3 - 4 rad
Danger	> 300°C	> 6.58 bar	> 4 rad

Write your adapters in AspectJ. You are not allowed to change anything in the java code except for the Main class. There you should add new sensors and write the output of the status monitor onto the console.
b) The values above are only the default values. Find a way to add individual values to each sensor and let the status requester use these values. Don't forget to adapt the Main class.