Lab 5 - Object-Oriented Software Development with Design Patterns TOUK18 - H22

Design Patterns/Principles
HARD deadline: Monday, 24 October, 23:59

Jönköping University, School of Engineering
Monday, 26 September

General instructions

In this lab we deepen our understanding of design patterns and principles. We first establish some order, structure, and relations between the patterns we have encountered. In a second part we train our skills to identify design patterns suitable to help us in a given problem scenario.

1 Establish order

We have encountered many design patterns. Most of them in one of the categories of a) OO principles, b) GRASP, c) GoF. We want to rehearse some GRASP and GoF patterns by the following exercise.

Task 1: For each of the six GoF patterns *Factory, Strategy, Composite, Adapter, Facade, Observer* identify and specify its relations to the nine GRASPatterns. You may want to represent this graphically (in UML?). The minimum requirement is that you find for each of the six GoF patterns **at least one** (2-4 are possible) related GRASPattern and that you **describe** the relation in text. To give an example of a relation, *Facade* is a way to achieve *Protected Variations*. Our book *Applying UML and Patterns* mentions a number of relations in chapter 26, *Applying GoF Design Patterns*. You might find other relations and other types of relations as well. There are also interesting relations among the GRASPatterns themselves and also among the GoF patterns themselves. It is certainly a good exercise to think about those as well, though we don't require it for this task.

2 Problem scenarios

Think about our tic tac toe game design from lab 4. We came up with the classes Controller, View, Player, Board, RuleEngine.

2.1 Scenario 1

Imagine we want to add the functionality to save/load a game to/from hard-drive.

Task 2: Which class should be assigned the responsibility of taking care of saving and loading the game's state? Use GRASP to evaluate alternatives and suggest a solution. Motivate your answers.

2.2 Scenario 2

Immagine we want to add the functionality to play the game via network, with different players located at different computers with different IP-addresses. This means in essence that the RuleEngine sends updates to the players not directly, but through the network. Similarly, the players send move requests to the RuleEngine not directly, but through the network. We want of course our game to be portable, that is, it should run on different operating systems. Using Java as implementation language provides this platform-interoperability pretty well and automatically. Furthermore, there are rather simple functions for sending and receiving data to and from a computer through the network. These functions are supported by almost all modern operating systems and are, to a certain extend, standardized (*socket* programming: *send()* and *recv()*). Unfortunately, the error messages and codes that the *send()* and *recv()* functions return are different for different operating systems. Even though we use Java and socket programming! This implies every time we call *send()* and/or *recv()*, the interpretation of the returned error code depends on the operating system we currently run on.

Task 3: Which design patterns help us find a solution to this problem? Which solutions are available? What is "the best" solution? Motivate your answers.

3 Deliverables

You shall deliver *one* pdf document that contains answers to tasks 1-3. For tasks 2 and 3 you are expected to deliver a reasonable piece of text (but in total not more than 1.5 pages). Additional UML drawings are allowed. Submit your pdf on Canvas under the assignment "Lab 5". If you have hand-made drawings, this is accepted, as long as it is **well-readable** and submitted within the pdf (scann it). Note that *no oral presentation* is required for this lab.