Design Patterns and Frameworks	**Exercise Sheet No. 5**
Dipl.-Inf. Steffen Zschaler	Software Engineering Group
INF 2097	Institute for Software and Multimedia Technology
`http://st.inf.tu-dresden.de/teaching/dpf`	Department of Computer Science
	Technische Universität Dresden
	01062 Dresden

Extensibility Patterns: Extension Access

Task 1: Lively Iteration

Cellular automata [1] are mathematical 'machines' consisting of a grid of individual so-called 'cells', each of which can be in one of k states (typically called 'colours'). A cellular automaton evolves in so-called generations: The state of all cells is updated simultaneously following rules based on the colours of the cells in a certain neighbourhood of the currently evolving cell. Different types of cellular automata exist; they vary in the shape and size of the grid, the number of possible colours per cell, the size and shape of a cell's neighbourhood, and the evolution rules.

1a)

Design and implement a class `CellGrid` that represents a grid of cells of a cellular automaton. Provide an interface for access to the individual cells, but hide the structure of the grid (number of dimensions, structure and size of neighbourhood, etc.) from clients.

What design pattern can you use? How do you use it?

1b)

Using the `CellGrid` class from above, design and implement a class `GridChanger` that realises Conway's Game of Life [2]. Life is played on a two-dimensional grid, with cells that are either black ("dead") or white ("alive"). The neighbourhood of a cell consists of the 8 cells directly adjacent to it. For each cell, the colour of the cell in the next generation depends on the cell's current colour and on the number of living (i.e., white) cells in its neighbourhood:

- The cell *dies* (i.e., turns black in the next generation) if it is alive and there are less than 2 or more than 3 living cells in the neighbourhood.
- The cell *survives* (i.e., stays white in the next generation) if it is currently alive and there are 2 or 3 living cells in the neighbourhood.
- The cell *is born* (i.e., turns white in the next generation) if it is currently dead and there are exactly 3 living cells in its neighbourhood.
- The cell *stays dead* (i.e., stays black in the next generation) if it is dead and there are more or less than 3 living cells in its neighbourhood.

1c) *

Use additional patterns (Strategy, Observer, Interpreter, ...) to implement a completely generic cellular automaton grid. Parametrize size and structure of the grid, size and structure of a cell's neighbourhood, number of colours, and the rules for determining a cell's colour in the next generation.

This task is of added complexity. We may not discuss it in the exercise, but I will be happy to comment on any solution of yours that you send me.

Bibliography

1. Eric W. Weisstein. *Cellular Automaton.* From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/CellularAutomaton.html`
2. Eric W. Weisstein. *Life.* From MathWorld—A Wolfram Web Resource. `http://mathworld.wolfram.com/Life.html`

Task 2: Extensible Insurance Contracts

Insurance contracts are very long-lived documents that are treated by many different people for many different reasons during their life-time. In a software system for the management of insurance contracts, each of these clients needs a custom interface to the insurance-contract object. Sometimes the same person needs to treat very different types of insurance contracts in the same manner. Occasionally, new types of treatment need to be added dynamically, without affecting any of the pre-existing code.

2a)

Understand the design pattern Extension Object [1]. What are its elements? How do they collaborate to support solving the above problems?

2b)

Use the Extension Object pattern to design and implement an insurance contract management system. Support the following roles:

- Initialization: The contract object has just been created and needs to be filled with the correct data.
- Conclusion: The contract has been accepted and needs to be signed by all parties.
- Termination: The contract's duration has passed, all the money goes to the company :-)

2c)

What do you have to do to support another role "Incident" (i.e., the incident against which the insurance is held, has occurred)?

2d)

How can different document types (for example, insurance contracts, but also letters, etc.) support the same role?

Bibliography

1. Gamma, E. 1997. *Extension object.* In Pattern Languages of Program Design 3, R. C. Martin, D. Riehle, and F. Buschmann, Eds. Addison-Wesley Software Pattern Series. Addison-Wesley Longman Publishing Co., Boston, MA, 79–88.
   Also at http://st.inf.tu-dresden.de/Lehre/WS06-07/dpf/gamma96.pdf
   A nice tutorial also exists at http://www.design-nation.net/en/archives/000488.php