15-214: Principles of Software System Construction

Jonathan Aldrich and William Scherlis

Assignment 6: Cellular Automaton Framework Design

Assignment Goals

- Design an object-oriented framework, demonstrating the effective use of design patterns to achieve the framework's reuse goals.
- Carefully specify the framework design through the use of Java interfaces and informal documentation.

The Backstory. Facelook is expanding from viewing faces to viewing all kinds of living things—even digital forms of life. Inspired by Conway's Game of Life, Facelook is creating a wide array of cellular automatons for the viewing pleasure of Facelookers. The developers, however, are concerned about the cost of building so many different cellular automaton programs.

Fortunately, the Facelook CEO recently went to the SPLASH conference and came back dripping with excitement about the benefits of object-oriented frameworks. She has articulated a new strategy for development: instead of writing software for each cellular automaton individually, the company will invest in developing a cellular automaton product line. The object-oriented framework at the core of the product line should provide a skeleton that can be reused and customized to support multiple different cellular automata. The application design team has come up with a specification for the functionality to be provided by this framework, described in more detail below.

Framework Platform Specification. The framework should support the following application lifecycle:

- A `main()` program written by the plugin author creates the framework and connects it to at least two plugins: one that defines the cellular automaton to be run, and one that defines the way the cellular automaton should be visualized.
- It should be possible to use the same cellular automaton plugin with several different visualization plugins set up in different `main()` programs. The reverse need not be possible—a visualization may be specific to a particular automaton.
- `main()` starts the framework, passing in the two plugins.
- The framework shows a window with the cellular automaton simulating step by step.
- When the user closes the window, the simulation halts and the program exits.

Other features of the framework should include:

- Generality
    - The framework should support a range of cellular automata. This should include Conway's Game of Life (2-state, 2-dimensional automata with an 8-cell neighborhood), as well as close variants such as Brian's Brain and

    Seeds. It should also support more than 2 states, e.g. as in Wireworld or a Von Neumann cellular automaton. For descriptions of these and other cellular automata, see http://en.wikipedia.org/wiki/Cellular_automaton and the links from this article.
  - The visualization plugin should at a minimum allow the user to visualize different cell states with colors appropriate to the state.
- Power
  - The framework should reuse as much common functionality as possible, given the range of cellular automata it targets.
  - The framework should include the basic structure of the user interface, e.g. displaying a 2-dimensional grid of cells. The visualization plugin is to be used to customize this—e.g. to pick an appropriate color. or possibly symbol for the state of each cell in the grid.
- Custom Feature. Your manager has asked you to propose N-1 features that go beyond what is described above (N==group size). This feature should enhance in some way the services provided by the framework, and should extend the way the plugin and framework interact. Example extensions might include:
  - Supporting more diverse forms of cellular automata. For example, you could support one-dimensional cellular automata such as Stephen Wolfram's Rule 110. Or, you could support a "machine state" in addition to the state of each cell, as in Langton's ant or Turmite. For the ambitious, you could support a different kind of grid (e.g. hexes, or 3-D grids).
  - Supporting a more interesting UI. For example, in addition to a color for each state, the visualization plugin could provide the ability to specify a custom symbol such as an arrow or an ant (for Langton's ant).
  - Your own idea here—the list above is not exhaustive!

The above features may be incomplete, so discussion of design goals on company Piazza forums is encouraged.

Design Task. Designing this infrastructure calls for the premier design team[1] in the company—you! Your manager has asked your team to develop a preliminary design for a framework which implements this specification. You should specify your design with Java interfaces, document it carefully using prose, UML and Javadoc, and provide sample automata and visualization plugin code for instantiating the framework into an application. After you produce your design, it will be reviewed both by your boss[2] and by the Facelook Review Team[3]. You should therefore document your framework with your peers at Facelook as the intended audience.

[1] This assignment is intended to be done in **groups of 3-4**. Individual students may do this assignment only with prior instructor/TA permission. **We highly encourage all group members to be in the same section.** Expectations will be increased slightly for larger groups (the number of custom features expected), but expectations will *not* be decreased for individuals.

[2] The TA

[3] Another group in the class

You should prepare a design document for your manager that contains the following elements:

- A specification of the functionality of your custom feature(s). Be at least as specific as the descriptions of the other features mentioned above. Also, your framework design is permitted to deviate from the specification above, if you think you can make a more compelling framework that way; if you make such deviations, explain them and their rationale.
- One or more UML diagrams (including at least one class diagram) that describe your design
- A detailed design rationale to accompany the class diagrams. The description of the design rationale should explain how the framework meets the goals above. It should identify at least two design patterns used in the framework design, and the purpose of each one; ideally, this will form a natural part of how the framework meets the design goals, rather than being a separate list. Describe the extension points that clients can plug in to, and any constraints that the framework imposes on clients that use it. It may be helpful to show how the scenarios from the goals description can be concretely realized with the framework design. The design rationale should sufficiently detailed for the Facelook Review Team to evaluate the quality of your design and ascertain that it meets its goals, and for development teams to start designing plugins to go into your framework.
- Write a set of Java interfaces, covering all communication between the framework and its game plugin(s). In some cases specifying the interface of a Java class may be necessary as well, for example in the case of a static method (static methods are not permitted in Java interfaces)—however, to the greatest extent possible, the specification should be done in terms of interfaces alone.
- Document your interfaces with Javadoc as necessary. The company standard specifies that each method should have a Javadoc comment which describes the purpose of that method, documents what each argument is for and what the result indicates, and explains what checked exceptions may be thrown and under what circumstances. Each interface should include a comment explaining what its purpose is, and how it should be used. These guidelines are a minimum; add any additional information you believe a plug-in developer would need to know in order to successfully use your framework.
- Implement a trivial demonstration plugin that compiles correctly against the specified interfaces, and uses 80% of the methods in the framework interface. The trivial plugin will, of course, not run correctly as the framework interface has not yet been implemented.

The design document should be readable by your classmates. Your design should be implementable in under 1 week of class time for a group of 3-4 people—say about 15-20 hours total. You do not have to implement your design *yet*, however.

Turning in Deliverables

Your files **MUST BE IN THE REQUIRED FORMAT**.
Put all documents into a file hierarchy with the organization below, and zip them together. Submit to Frontdesk.

Your zip should include an Eclipse project (ie a `.project` file), with the following directory structure:

`doc/specification.pdf`	: your custom feature specification document (PDF);
`doc/rationale.pdf`	: your design rationale document (PDF) possibly including your UML design diagrams;
`doc/diagrams/`	: UML diagrams: **ONLY** if not a part of the rationale file;
`src/interfaces/`	: the specified Java interfaces, and the trivial plugin implementation;
`doc/javadoc/`	: Javadoc HTML documentation generated from the code in the interfaces/ directory (see the `javadoc` console command for details on how to generate these).

Grading Expectations *(subject to change)* – 100 points total

Group

10 points for the custom feature specification, based on clarity.

60 points for the design. Grading criteria include:

- Readability of the design diagrams
- Correct use of design patterns in the framework (there must be at least 2), identifying the patterns used in the document and clearly explaining their purpose (which should be appropriate to the pattern).
- Adequate description of how the design of the framework fulfils the goals in the specification
- Description of how clients should plug into the framework and a description of any necessary constraints on those plugins

20 points for the interfaces and associated javadoc. The code should compile without errors. Javadoc should be used appropriately as described above.

10 points for the trivial plagin. Again, it should compile without errors.