SWE 265P
Reverse Engineering and Modeling

Lecture 8

Duplication of course material for any purpose without the explicit written permission of the professor is prohibited.
“That said, anything about architecture, design patterns, or language conventions helps in interpreting code and creating a mental model of the code.” – Lee Martie [Research staff engineer, MIT-IBM Watson AI Lab]
“Good programmers will write the code according to these patterns.” – Eric Dashofy [General Manager & Deputy CIO, The Aerospace Corporation]
“Well designed APIs are also super important.” – Ping Chen
[Software Engineer, Google]
Today

- Last week’s material
- Key expert practices
- Design patterns
- In-class practice
- Alberto Krone-Martins (UCI)
Last week’s material

- Key expert practices
  - know how things work
  - address knowledge deficiencies
  - design elegant abstractions

- Architecture

- Social context

- Any questions?
Last week’s homework – architecture

• How difficult was it to document the architecture as-implemented?

• What approach(es) did you use to do so?

• What did you learn about your system?

• Any questions?
Last week’s homework – social context

• How difficult was it to document the social context of your system?

• What approaches did you use to do so?

• What did you learn about expectations on you as future contributors?

• Any questions?
Last week’s homework – issues

• How difficult was it to document five interesting pull requests and five interesting issues?

• How did you decide what was interesting (and what was not)?

• What did you learn about your system / how to contribute to your system?

• Any questions?
KEP #13: invest now to save effort later
KEP #14: socially embed and reinforce good practice

*Cool design, some flexibility needed though*

*Still to test w/JMeter*

You should add: `Lock()`, `Unlock()`

Add: `read`, `write`

*Triggers should use observer pattern*
KEP #15: use analogy

Apple? Orange?

- Round ✓
- Juicy ✓
- Edible skin ✗
A regularity in the world, in human-made design, or in abstract ideas. As such, the elements of a pattern repeat in a predictable manner.
Design pattern

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice.

name – problem – solution – consequences
Software design pattern

A general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations.
Software design pattern

Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.
# Original catalogue of patterns

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Creational</th>
<th>Structural</th>
<th>Behavioral</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Abstract factory</td>
<td>Adapter</td>
<td>Chain of responsibility</td>
</tr>
<tr>
<td></td>
<td>Builder</td>
<td>Bridge</td>
<td>Command</td>
</tr>
<tr>
<td></td>
<td>Factory method</td>
<td>Composite</td>
<td>Interpreter</td>
</tr>
<tr>
<td></td>
<td>Prototype</td>
<td>Decorator</td>
<td>Iterator</td>
</tr>
<tr>
<td></td>
<td>Singleton</td>
<td>Façade</td>
<td>Mediator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flyweight</td>
<td>Memento</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proxy</td>
<td>Observer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>State</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Template method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Visitor</td>
</tr>
</tbody>
</table>
Underlying design principles

Program to an interface, not an implementation

Favor object composition/delegation over inheritance

Find what varies and encapsulate it
Sample pattern: factory
Sample pattern: strategy
Sample pattern: adapter
Sample pattern: template method
Let’s practice

• Can you find the Factory pattern in JPacMan4?
• Can you find the Listener pattern in JPacMan4?
• Can you find the Builder pattern in JPacMan4?
• Any others?
Further reading

- [https://refactoring.guru/design-patterns/catalog](https://refactoring.guru/design-patterns/catalog)

- [https://www.youtube.com/playlist?list=PLrhzvlci6GNjpARdnO4ueTUA9R9eMBpc](https://www.youtube.com/playlist?list=PLrhzvlci6GNjpARdnO4ueTUA9R9eMBpc)

- [http://hillside.net/patterns/patterns-catalog](http://hillside.net/patterns/patterns-catalog)

- [https://www.youtube.com/playlist?list=PLrhzvlci6GNjpARdnO4ueTUA9R9eMBpc](https://www.youtube.com/playlist?list=PLrhzvlci6GNjpARdnO4ueTUA9R9eMBpc)
Homework (team)

• With your team, identify five different design patterns that are used in your system
  – provide an example per pattern
  – explain why the pattern is in use – what benefit does it provide in this specific example?
Homework (team)

- With your team, code up your first issue and submit a pull request
- Start thinking about coding up your second issue
  - issue should be of non-trivial complexity
  - issue should be approved by TA
Homework (individual)

• Make sure to regularly update your personal diary, including an entry for today’s lecture
Break
And now...

• ...welcome Alberto!