Object-Oriented Design:
CRC cards and design principles

15-413: Introduction to Software Engineering

Jonathan Aldrich

Announcements

- Iteration reports/plans due Wednesday
  - You should already be working on last iteration
    - 2 weeks, 12 hours/person
    - Focus on quality assurance, final value to clients
- Please fill out faculty course evaluations
  - I will supplement the university form with an anonymous Blackboard survey with more detailed questions
  - We appreciate your input!
- Next week
  - Mon/Wed: present tool evaluation results
  - Wed (second half): review for exam
    - Please send me what you’d like to cover!
  - Friday: final exam

28 November 2005
OO Design Challenge

- What classes should be in a system?
- What should each one do?
- How should they collaborate?

- Principles
  - Hide things likely to change
  - Maximize cohesion
    - The data and operations of a class should all be related
    - Often helpful to look at concepts from problem domain
  - Minimize coupling
    - Simplify interactions between classes as much as possible
  - Protect user from errors
  - Iterate by considering alternative designs

CRC Cards

- A lightweight collaborative design method
- **Class, Responsibility, Collaboration**
  - Responsibilities: tasks to perform, data to track
  - Collaborations: other objects this object works with

<table>
<thead>
<tr>
<th>Class name:</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
<td>Collaborations:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CRC Cards

- A lightweight collaborative design method
- **Class, Responsibility, Collaboration**
  - Responsibilities: tasks to perform, data to track
  - Collaborations: other objects this object works with

<table>
<thead>
<tr>
<th>Class name: Shape</th>
<th>Superclass: Drawable</th>
<th>Subclasses: Circle, Rectangle, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
<td>Collaborations:</td>
<td></td>
</tr>
<tr>
<td>Knows location, size</td>
<td>Point, BoundingBox</td>
<td></td>
</tr>
<tr>
<td>Draw self on canvas</td>
<td>Canvas</td>
<td></td>
</tr>
<tr>
<td>Move self</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

28 November 2005

CRC Cards

- A lightweight collaborative design method
- **Class, Responsibility, Collaboration**
  - Responsibilities: tasks to perform, data to track
  - Collaborations: other objects this object works with

<table>
<thead>
<tr>
<th>Class name: Canvas</th>
<th>Superclass: Window</th>
<th>Subclasses: Drawable, List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities:</td>
<td>Collaborations:</td>
<td></td>
</tr>
<tr>
<td>Knows drawables</td>
<td>Drawable, List</td>
<td></td>
</tr>
<tr>
<td>Draw self</td>
<td>Drawable</td>
<td></td>
</tr>
<tr>
<td>Drawing operations</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

28 November 2005
Serialization Framework Design

- Allow developers to use the framework to aid them in serializing data into a file or database

<table>
<thead>
<tr>
<th>Class name:</th>
<th>Superclass:</th>
<th>Subclasses:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canvas</td>
<td>Window</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities:</th>
<th>Collaborations:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knows drawables</td>
<td>Drawable, List</td>
</tr>
<tr>
<td>Draw self</td>
<td>Drawable</td>
</tr>
<tr>
<td>Drawing operations</td>
<td></td>
</tr>
</tbody>
</table>

Refining CRC into a design

- How to go from CRC cards to OO interfaces (e.g. UML)?
  - Walk through scenarios
    - Operations performed
    - Communication between classes
    - UML: Use case → Sequence diagram
  - Infer interface from operations in scenarios
  - Iterate according to design criteria
    - Design for change
    - Cohesion
    - Coupling
    - Robustness to errors
Framework Design Issues

• Interfaces vs. classes
  • Use interfaces when there are multiple implementations, or when implementors may want to inherit code from elsewhere
  • Use classes when there is useful code to inherit
  • If both apply, provide an interface and an abstract class that implements it

• Correct extension
  • Use final for all methods that are not specifically intended for extension

Framework Design Issues

• Correct framework use
  • If possible, ensure that all constructors result in fully initialized object
    • Don’t require calls to init or set methods later
  • Check invariants on input and throw meaningful error
    • Better to increase usability even at cost of performance
    • Use assert to make this efficient
  • Use the type system
    • Ensure type dependencies are captured in interface types
    • Parameterize where necessary
Framework Design Issues

- Prohibit illegal calls
  - Hide methods clients shouldn’t be able to touch
    - Internal to framework
    - Only safe to call under certain conditions that other methods check
    - Note: Java, C# make this hard because packages are flat
  - Hide internal objects
    - Make the classes private, or just don’t return the instance from public methods
  - Ordering constraints
    - If method A returns an object needed for method B, then can’t accidentally call B first
  - Provide read-only access to data structures that clients shouldn’t modify directly