Design patterns

CSE 403
What is a design pattern?

• A **standard solution** to a common programming problem
  – a design or implementation structure that achieves a particular purpose
  – a high-level programming idiom

• A technique for making code **more flexible**
  – reduce coupling among program components

• **Shorthand** for describing program design
  – a description of connections among program components
  – the shape of a heap snapshot or object model
Why should you care?

• You could come up with these solutions on your own
  – You shouldn't have to!

• A design pattern is a known solution to a known problem
Example design patterns

• Encapsulation (data hiding)
• Subclassing (inheritance)
• Iteration
• Exceptions
• Generics
Creational patterns

Constructors in Java are inflexible
  1. Can't return a subtype of the class they belong to
  2. Always return a fresh new object, never re-use one

• Factories
  – Factory method
  – Factory object
  – Prototype
  – Dependency injection

• Sharing
  – Singleton
  – Interning
  – Flyweight
Structural patterns: Wrappers

A wrapper translates between incompatible interfaces.

Wrappers are a thin veneer over an encapsulated class.
- modify the interface
- extend behavior
- restrict access

The encapsulated class does most of the work.

Subclassing vs. delegation

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Functionality</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter</td>
<td>same</td>
<td>different</td>
</tr>
<tr>
<td>Decorator</td>
<td>different</td>
<td>same</td>
</tr>
<tr>
<td>Proxy</td>
<td>same</td>
<td>same</td>
</tr>
</tbody>
</table>
Composite pattern (part-whole relations)

A client can manipulate the whole or any part
Example: AST (abstract syntax tree)

Question: Should we group together the code for a particular operation (procedural pattern) or the code for a particular expression (interpreter pattern)?

(A separate issue: given an operation and an expression, how to select the proper piece of code?)
When (not) to use design patterns

• Rule 1: delay
  – Understand the problem & solution first, then improve it

• Design patterns can increase or decrease understandability
  – Add indirection, increase code size
  + Improve modularity, separate concerns, ease description

• If your design or implementation has a problem, consider design patterns that address that problem

• References:
  – *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Addison-Wesley, 1995.