Recap

Design patterns
- What is a design pattern?
- Pros and cons of design patterns.
- Categories of design patterns:
  - Structural patterns
  - Behavioral patterns
  - Creational patterns
- Two design problems & potential solutions
  - Complex/composed View -> Composite pattern
  - Buffered input streams -> Decorator pattern

Logistics

Homework 1
- **Deadline:** 09/20/2016, 11:55pm.

Paper reading 1
- Group selection
  - 4 students per group,
  - opens: 09/20/2016, 5:30pm,
Recap: Composite pattern

<<interface>>
Component
+operation()
CompA
+operation()
CompB
+operation()
+operation()
-comps:Collection<Component>
+operation()
+addComp(c:Component)
+removeComp(c:Component)
1..n

Recap: Decorator pattern

<<interface>>
Component
+operation()
CompA
+operation()
CompB
+operation()
+operation()
+Decorator(d:Component)
+operation()

Recap: Composite vs. Decorator

<<interface>>
Component
+operation()
CompA
+operation()
CompB
+operation()
+operation()
-comps:Collection<Component>
+operation()
+addComp(c:Component)
+removeComp(c:Component)
1..n
Decorator
-decorated:Component
+Decorator(d:Component)
+operation()

Today

More on design patterns
- Behavioral patterns
  - Template method pattern
  - Strategy pattern
  - ...
- Live coding
Find the median in an array of doubles

Examples:
- median([1, 2, 3, 4, 5]) = 3
- median([1, 2, 3, 4]) = 2.5

Algorithm:
Input: array of length $n$     Output: median
1. Sort array
2. if $n$ is odd return \((n+1)/2\)th element
   otherwise return arithmetic mean of
   \((n/2)\)th element and \((n/2)+1\)th element

Live coding: naïve solution

See code examples (online)
- naive
  - 1: Monolithic version, static context.
  - 2: Extracted sorting method, non-static context.
  - 3: Proper package structure and visibility, extracted main method.
  - 4: Proper testing infrastructure and build system.

Use ant to compile and test the code:
- $ant -p$ => list all targets
- $ant compile$ => compile the code
- $ant test$ => run all tests
Live coding: template method pattern

See code example (online)

- template
  - Abstract superclass with template method and abstract method `sort`.
  - Two subclasses with concrete implementations for the method `sort`.

Use `ant` to compile and test the code:

- `$ant -p` => list all targets
- `$ant compile` => compile the code
- `$ant test` => run all tests (note the run-time differences between `SimpleMedianTest` and `QuickMedianTest`!)

One possible solution: template method pattern

<table>
<thead>
<tr>
<th>AbstractMedian {abstract}</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ median(a:double[]):double # sort(a:double[])</td>
</tr>
</tbody>
</table>

- The template method (`median`) implements the algorithm but leaves the **sorting** of the array undefined.

- The concrete subclass only needs to implement the actual **sorting**.

Should the median method be final?

One possible solution: template method pattern

<table>
<thead>
<tr>
<th>AbstractMedian {abstract}</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ median(a:double[]):double # sort(a:double[])</td>
</tr>
</tbody>
</table>

- The template method (`median`) implements the algorithm but leaves the **sorting** of the array undefined.

- The concrete subclass only needs to implement the actual **sorting**.