CS 520/620
Advanced Software Engineering
Fall 2016

September 22, 2016
Logistics

Homework 2

- **Goal:** Code review, (re)design, and refactoring.
- **Deadline:** 10/04/2016, 11:55pm.

![Simple stats image](image)
Recap

- Why and when to use packages (in Java)?
- Why and when to use a build system?
- Improving a naïve implementation
  - Separation of concerns and proper visibility.
  - Static vs. non-static context.
  - Template method pattern.
Recap: the problem

Find the median in an array of doubles:
- median([1, 2, 3, 4, 5]) = 3
- median([5, 3, 1, 4, 2]) = 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
Recap: template method pattern

AbstractMedian
{abstract}

+ median(a:double[]):double
# sort(a:double[])

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

SimpleMedian

# sort(a:double[])

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

- Improving a naïve implementation cont.
  - Basic testing.
  - Strategy pattern.
  - Template method pattern vs. strategy pattern.
- Observer pattern
- Iterator pattern
- MVC revisited
Live coding: strategy pattern

See code example (online)

- strategy
  - Interface *Sorter* for sorting strategies that defines the method *sort*.
  - Two implementations of this interface (*BubbleSort* and *QuickSort*).
  - *StrategyMedian* delegates the sorting to a sorting strategy, which can be configured and changed at run time.

Use `ant` to compile and test the code:

- `$ant -p` => list all targets
- `$ant compile` => compile the code
- `$ant test` => run all tests (note how `testSwapSorter` in `StrategyMedianTest` changes the sorter at run time!)
Strategy pattern

<<interface>>
Median
+median(a:double[]):double

<<interface>>
Sorter
+sort(array:double[])
Strategy pattern

```
<<interface>>
Median
+median(a:double[]):double
```

```
<<interface>>
Sorter
+sort(array:double[])
```

```
StrategyMedian
-sortStrategy:Sorter
+median(a:double[]):double
+setSorter(s:Sorter)
```

```
Median
def median(a:double[]):double
```

```
Sorter
def sort(array:double[])
```

```
StrategyMedian
```

```
HeapSort
+sort(...)
```

```
QuickSort
+sort(...)
```

“median” delegates the sorting of the array to a “sortStrategy”, which can be configured and changed at runtime.
Template method pattern vs. strategy pattern

Two solutions to the same problem

Template method
- Behavior selected at compile time.
- Don’t call us, we’ll call you.
- Template method is usually final.

Strategy
- Behavior selected at runtime.
- Composition/aggregation over inheritance.
Observer pattern

- Models a “one to many” dependency.
- Decouples state and action:
  Notify registered observer(s) about state change.
Observer pattern

```java
interface Observer
    + update()

abstract class Observable
    # observers:Set<Observer>
    + register(o:Observer)
    + unregister(o:Observer)
    + stateChanged()

class MyObservable
    - state:State
    + getState():State
    + setState(state:State)
```
Observer pattern

**Observable**

<table>
<thead>
<tr>
<th>{abstract}</th>
</tr>
</thead>
<tbody>
<tr>
<td># observers:Set&lt;Observer&gt;</td>
</tr>
<tr>
<td>+ register(o:Observer)</td>
</tr>
<tr>
<td>+ unregister(o:Observer)</td>
</tr>
<tr>
<td>+ stateChanged()</td>
</tr>
</tbody>
</table>

**Observer**

<table>
<thead>
<tr>
<th>&lt;&lt;interface&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observer</td>
</tr>
<tr>
<td>+ update()</td>
</tr>
</tbody>
</table>

**MyObservable**

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>- state:State</td>
</tr>
<tr>
<td>+ getState():State</td>
</tr>
<tr>
<td>+ setState(state:State)</td>
</tr>
</tbody>
</table>

Variations exist that pass incremental changes or the state to the update method.
Iterator pattern

- Provides (sequential) access to a data structure.
- Does not reveal implementation details.
Iterator pattern

- Provides (sequential) access to a data structure.
- Does not reveal implementation details.
- Examples
  - for (element : collection){
    System.out.println(element);
  }
  - Iterator iter = collection.iterator();
    while (iter.hasNext()) {
      element = iter.next();
      System.out.println(element);
    }
MVC revisited

Model View Controller (MVC)

Client sees View

View uses Controller

Controller manipulates Model

Model updates View
MVC revisited

Design patterns in an MVC architecture

View
Controller
Model

Composite
Strategy
Observer
Iterator

Client sees
uses

view updates
controller manipulates
Homework 2

The simple stats application

1, 2, 2, 2, 1, 10,