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

- 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.

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[])

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

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

Strategy pattern

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

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

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

```
StrategyMedian
  - sortStrategy:Sorter
    +sort(...)
```

```
Median
  +median(a:double[]):double
  +setSorter(s:Sorter)
```

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

```
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

```
<<interface>>
Observer
+ update()

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

MyObservable
+ getState():State
+ setState(state:State)
- state:State
```

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

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

Observable
{abstract}

#{ observers:Set<Observer> }
+ register(o:Observer)
+ unregister(o:Observer)
+ stateChanged()

MyObservable
- state:State
+ getState():State
+ setState(state:State)

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

Observer pattern

<<interface>>
Observer
+ update()

Observable
{abstract}
MyObservable
+ getState():State
+ setState(state:State)

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 Controler (MVC)

Client sees

View updates

Controller uses

Model manipulates
MVC revisited

Design patterns in an MVC architecture

View

Controller

Model

Client

sees

uses

manipulates

updates

Composite

Strategy

Observer

Iterator

Homework 2

The simple stats application