

## Programming Tools

David Rabinowitz

### This Lecture

- Personal Productivity Tools
  - And how to use them
- Refactoring
- Static Analysis & Metrics
- Profiling

April 26, 2006

Object Oriented Design Course

2

### Refactoring

- Improving the design of existing code, without changing its observable behavior
- Here's the *Extract Method* refactoring:

Before:	After:
<pre>void f(int[] a) {   ...   // compute score   score = initial_score;   for (int i=0; i&lt;a.length; i++)     score += a[i] * delta; }</pre>	<pre>void f() {   ...   computeScore(); } computeScore(int[] a) {   // code cut &amp; pasted here }</pre>

April 26, 2006

Object Oriented Design Course

3

### Why?

- Why Refactor?
  - Improve software design
  - Make software easier to understand
  - Help find bugs
  - Help program faster
- Preconditions
  - Working code
  - Good set of unit tests

April 26, 2006

Object Oriented Design Course

4

### When?

- When to refactor
  - Before adding functionality
  - Before fixing a bug
  - During code review
- When not to refactor
  - During adding functionality
  - During fixing a bug
  - No good set of unit tests
  - Small programs (usually)

April 26, 2006

Object Oriented Design Course

5

### Code Smells

- "If it stinks, change it"
  - Duplicate code
  - Switch statements
  - Long method
  - Data class
  - Long parameter list
  - Primitive obsession
  - Temporary field
  - ...

April 26, 2006

Object Oriented Design Course

6

## Documented Refactorings

- There's a catalog
  - Fowler's book
  - [www.refactoring.com/catalog](http://www.refactoring.com/catalog)
- There are many others
- Way to learn good OOD principles
- Pay attention to the mechanics

April 26, 2006

Object Oriented Design Course

7

## Automated Refactorings

- Eclipse's 'Refactor' menu automates thing
  - Undo & Redo
  - Physical Structure
  - Class Level Structure
  - Structure Inside a Class
- Wizards & Preview Windows Included
- Other tools exist: see [refactorit.com](http://refactorit.com)

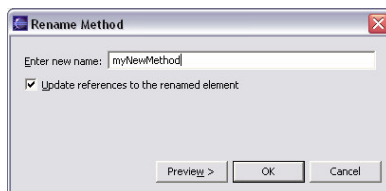
April 26, 2006

Object Oriented Design Course

8

## Automated Refactoring Example

- The 'Rename' Refactoring renames any Java element, and references to it:



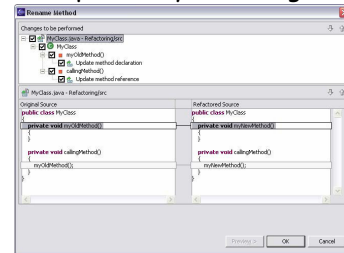
April 26, 2006

Object Oriented Design Course

9

## Automated Refactoring Example II

- You can preview your changes:



April 26, 2006

Object Oriented Design Course

10

## Encapsulate Field

Before:

```
public String name;
```

After:

```
private String name;
```

```
public String getName() {  
    return name;  
}
```

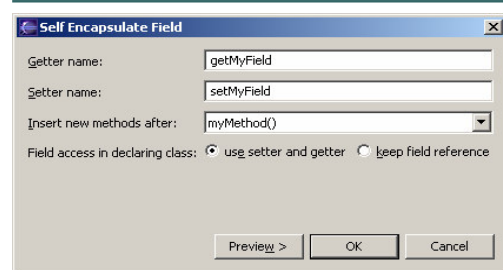
```
public void setName(String n) {  
    name = n;  
}
```

April 26, 2006

Object Oriented Design Course

11

## Encapsulate Field in Eclipse



April 26, 2006

Object Oriented Design Course

12

### Introduce Null Object

Before:

```
if (project == null)
    plan = Plan.default();
else
    plan = project.getPlan();
```

After:

```
class NullProject
    implements Project {
    public Plan getPlan() {
        return Plan.default();
    }
    // other Project methods
}
```

• This is the *Null Object Pattern*

April 26, 2006

Object Oriented Design Course

13

### Parameterize Method

Before:

```
class Server {
    handleGet(...)
    handlePut(...)
    handleSet(...)
}
```

After:

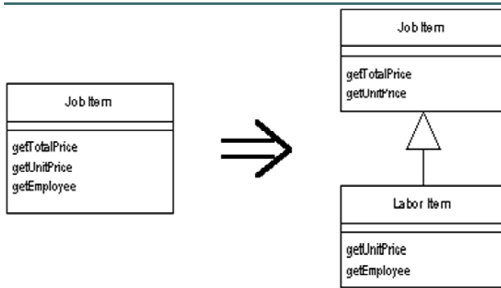
```
class Server {
    handle(EventType et, ...)
}
```

April 26, 2006

Object Oriented Design Course

14

### Extract Subclass

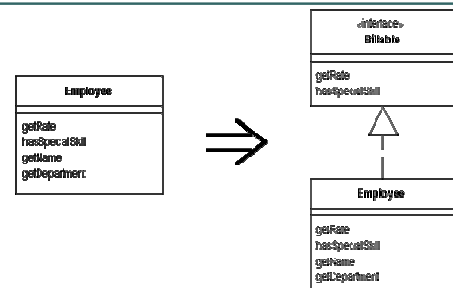


April 26, 2006

Object Oriented Design Course

15

### Extract Interface

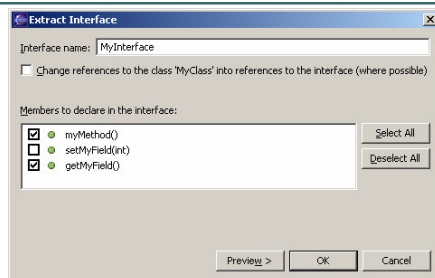


April 26, 2006

Object Oriented Design Course

16

### Extract Interface in Eclipse

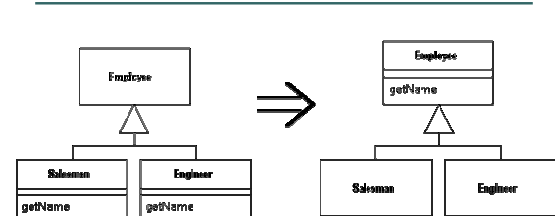


April 26, 2006

Object Oriented Design Course

17

### Pull Up Method

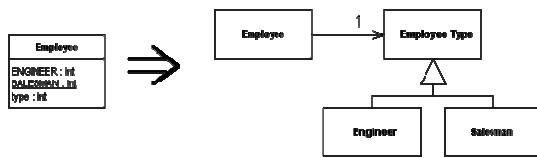


April 26, 2006

Object Oriented Design Course

18

### Replace Type Code with State/Strategy

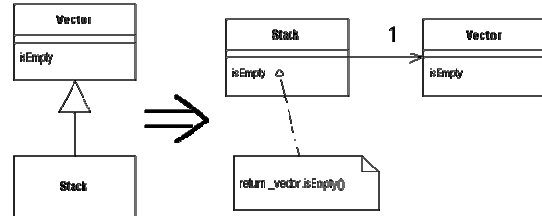


April 26, 2006

Object Oriented Design Course

19

### Replace Inheritance with Delegation

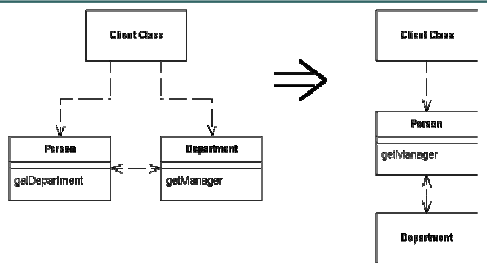


April 26, 2006

Object Oriented Design Course

20

### Hide Delegate



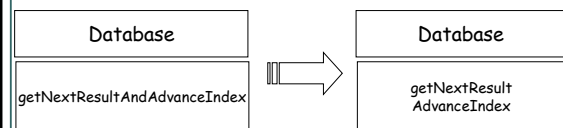
- Obeys the *Law of Demeter*

April 26, 2006

Object Oriented Design Course

21

### Separate Query from Modifier



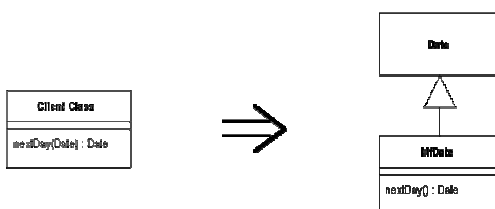
- Obeys *Command-Query Separation*

April 26, 2006

Object Oriented Design Course

22

### Introduce Local Extension



- Alternative: Introduce Foreign Method

April 26, 2006

Object Oriented Design Course

23

### The opposites are there too

- Inline method (extract method)
- Replace Parameter with Explicit Methods (Parameterize Method)
- Collapse Hierarchy (Extract subclass)
- Remove middle man (Hide delegate)
- Push down method (pull up method)
- Replace delegation with inheritance

April 26, 2006

Object Oriented Design Course

24

### More useful Refactorings in Eclipse

- Rename
- Move
- Change Method Signature
- Use Supertype where possible
- Extract Constant
- Introduce Factory
- ...

April 26, 2006

Object Oriented Design Course

25

### How to Refactor

- Recognize the smells
- Refactor in small discrete steps
- Test after each step
- Refactor in pairs
- Use documented refactorings
- Don't mix with adding functionality or fixing a bug

April 26, 2006

Object Oriented Design Course

26

### Static Code Analysis

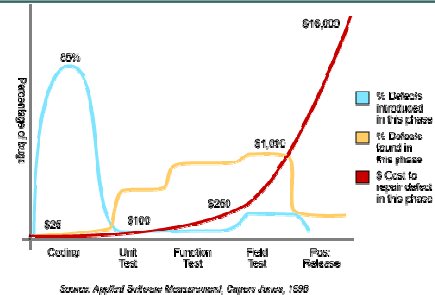
- Programs that help gain understanding of your code
- Find areas in the code with
  - Possible Bugs
  - "Fishy" Design
  - Inconsistent Style
- It's no replacement for testing
  - Finding (non-trivial) bugs is undecidable

April 26, 2006

Object Oriented Design Course

27

### Why is it so important?



April 26, 2006

Object Oriented Design Course

28

### Available Tools

- Commercial
  - Lint for C and C++ (see [gimpel.com](http://gimpel.com))
  - JTest ([parasoft.com](http://parasoft.com))
- Free Eclipse Plugins
  - JLint ([artho.com](http://artho.com))
  - CPD - Copy Paste Detector
  - PMD
  - CheckStyle
  - JDepend - Metrics

April 26, 2006

Object Oriented Design Course

29

### Lint

- Looks for over 800 C/C++ Issues
- Things that compilers either miss or allow
- Specific C++ Errors, for example:
  - Throwing from a destructor
  - Not checking for NULL argument in 'delete'
  - Order of initializations / constructors
  - Non-virtual over-riden methods
- Macro scanning
  - Incorrect parameter passing, Side effects, ...

April 26, 2006

Object Oriented Design Course

30

## Lint II

- Value Tracking
  - Division by zero, null dereference, out-of-bounds, memory leaks, double deallocation, ...
- Casting & Values
  - Loss of sign, truncations, Assignment in 'if', ...
- Specific C Issues
  - printf() arguments, order of evaluation: a[i] = i++;
- Style
  - Indentation, suspicious semi-colons (a > b); , ...
- Hundreds of other issues

April 26, 2006

Object Oriented Design Course

31

## JTest

- Checks for 380 Java & Style Issues
- Can automatically correct 160 of these
- Extensible by user-defined issues
- Supports metrics as well
  - Number of bytes, classes, lines, methods, ...
  - Issue = Deviation from acceptable metric range
- Some issues are shared with C/C++
  - Values, Casting, Unreachable code, Indentation, Comments, Initialization, Exceptions, ...

April 26, 2006

Object Oriented Design Course

32

## JTest II

- Other Java Specific Issues
  - Portability
  - Security
  - Optimization
  - Garbage Collection
  - Threads and Synchronization
  - Internationalization
  - Servlets / EJBs
  - Naming Conventions
  - ...

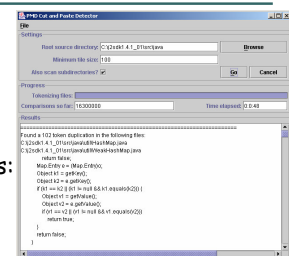
April 26, 2006

Object Oriented Design Course

33

## CPD - Copy Paste Detector

- Works with Java, C, C++ and PHP
- <http://pmd.sourceforge.net/cpd.html>
- From the examples:
  - A 307 lines(!) of duplicated code in Apache 2



April 26, 2006

Object Oriented Design Course

34

## PMD



- For Java code
- Checks
  - Unused local variables / parameters / private methods
  - Empty catch blocks
  - Empty 'if' statements
  - Duplicate import statements
  - Classes which could be Singletons
  - Short/long variable and method names
  - And many many more ...

April 26, 2006

Object Oriented Design Course

35

## CheckStyle

- Similar to PMD
- Javadoc Comments, Naming Conventions, Headers, Imports, Size Violations, Whitespace, Modifiers, Blocks, Coding Problems, Class Design, Duplicate Code

Id	Description	Resource	In Folder	Location
1	Name 'hashCode()' must match pattern '^[a-z]+(\\.[a-z0-9_]+)?\$'	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 2
1	Missing a Javadoc comment.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 5
1	'/' should be on the previous line.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 6
1	Missing a Javadoc comment.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 7
1	Missing a Javadoc comment.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 8
1	Name 'hashCode()' must match pattern '^[a-z]+(\\.[a-z0-9_]+)?\$'	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 10
1	Name 'hashCode()' must match pattern '^[a-z]+(\\.[a-z0-9_]+)?\$'	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 10
1	Variable 'hashCode()' must be private and have accessor methods.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 10
1	Missing a Javadoc comment.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 11
1	Variable 'hashCode()' must be private and have accessor methods.	FieldNames.java	JavaTest\\test\\src\\main\\java\\...	line 11

April 26, 2006

Object Oriented Design Course

36

### JDepend

- Calculates metrics for java packages
- Calculated metrics
- **CC** - Concrete Class Count
  - The number of concrete classes in this package.
- **AC** - Abstract Class Count
  - The number of abstract classes or interfaces in this package.

April 26, 2006

Object Oriented Design Course

37

### JDepend (2)

- **Ca** - Affluent Couplings
  - The number of packages that depend on classes in this package.
  - "How will changes to me impact the rest of the project?"
- **Ce** - Efferent Couplings
  - The number of other packages that classes in this package depend upon.
  - "How sensitive am I to changes in other packages in the project?"

April 26, 2006

Object Oriented Design Course

38

### JDepend (3)

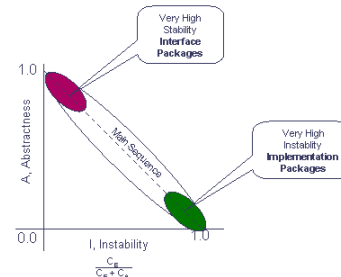
- **A** - Abstractness (0-1)
  - Ratio (0.0-1.0) of Abstract Classes (and interfaces) in this package.
  - $AC/(CC+AC)$
- **I** - Instability (0-1)
  - Ratio (0.0-1.0) of Efferent Coupling to Total Coupling.
  - $Ce/(Ce+Ca)$
- **D** - Distance from the Main Sequence (0-1)
- **Cyclic** - If the package contains a dependency cycle

April 26, 2006

Object Oriented Design Course

39

### The main sequence



April 26, 2006

Object Oriented Design Course

40

### Examples - Pet Store

Used By - Affluent Dependencies (58 Packages)

com.sun.j2ee.blueprints.catalog.exceptions (CC: 1 AC: 0 Ca: 0 Ce: 0 A: 0 I: 0 D: 1)
com.sun.j2ee.blueprints.catalog.model (CC: 4 AC: 0 Ca: 4 Ce: 0 A: 0 I: 0 D: 1)
com.sun.j2ee.blueprints.catalog.util (CC: 2 AC: 0 Ca: 0 Ce: 0 A: 0 I: 0 D: 1)
com.sun.j2ee.blueprints.contactinfo.ejb (CC: 2 AC: 3 Ca: 6 Ce: 5 A: 0.6 I: 0.45 D: 0.05)
com.sun.j2ee.blueprints.creditcard.ejb (CC: 1 AC: 3 Ca: 6 Ce: 2 A: 0.75 I: 0.25 D: 0)
com.sun.j2ee.blueprints.customer.account.ejb (CC: 0 AC: 3 Ca: 3 Ce: 2 A: 1 I: 0.4 D: 0.4)
com.sun.j2ee.blueprints.customer.ejb (CC: 0 AC: 3 Ca: 4 Ce: 2 A: 1 I: 0.33 D: 0.33)
com.sun.j2ee.blueprints.customer.profile.ejb (CC: 1 AC: 3 Ca: 6 Ce: 0 A: 0.75 I: 0 D: 0.25)
com.sun.j2ee.blueprints.encodingfilter.web (CC: 1 AC: 0 Ca: 0 Ce: 0 A: 0 I: 0 D: 1)
com.sun.j2ee.blueprints.item.ejb (CC: 1 AC: 2 Ca: 2 Ce: 2 A: 0.67 I: 0.5 D: 0.17)
com.sun.j2ee.blueprints.petstore.controller.ejb (CC: 3 AC: 8 Ca: 2 Ce: 6 A: 0.73 I: 0.75 D: 0.48 Cyclic)
com.sun.j2ee.blueprints.petstore.controller.ejb.actions
com.sun.j2ee.blueprints.petstore.controller.web
com.sun.j2ee.blueprints.petstore.controller.web.actions
com.sun.j2ee.blueprints.petstore.controller.web.actions
com.sun.j2ee.blueprints.petstore.controller.web.actions (CC: 6 AC: 0 Ca: 0 Ce: 24 A: 0 I: 1 D: 0 Cyclic)
com.sun.j2ee.blueprints.petstore.controller.events (CC: 12 AC: 0 Ca: 3 Ce: 4 A: 0 I: 0.57 D: 0.43)
com.sun.j2ee.blueprints.petstore.controller.ejb (CC: 3 AC: 8 Ca: 2 Ce: 6 A: 0.73 I: 0.75 D: 0.48 Cyclic)

April 26, 2006

Object Oriented Design Course

41

### Examples - Pet Store (2)

Depends Upon - Efferent Dependencies (51 Packages)

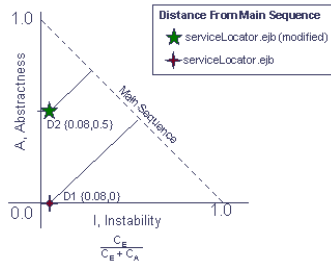
com.sun.j2ee.blueprints.petstore.controller.ejb (CC: 3 AC: 8 Ca: 2 Ce: 6 A: 0.73 I: 0.75 D: 0.48 Cyclic)
com.sun.j2ee.blueprints.cart.ejb
com.sun.j2ee.blueprints.customer.ejb
com.sun.j2ee.blueprints.servicelocator
com.sun.j2ee.blueprints.servicelocator.ejb
com.sun.j2ee.blueprints.waf.controller.ejb
com.sun.j2ee.blueprints.waf.controller.ejb.action
com.sun.j2ee.blueprints.waf.controller.ejb
com.sun.j2ee.blueprints.waf.event
com.sun.j2ee.blueprints.waf.event
com.sun.j2ee.blueprints.waf.exceptions
com.sun.j2ee.blueprints.petstore.controller.ejb.actions (CC: 6 AC: 0 Ca: 0 Ce: 24 A: 0 I: 1 D: 0 Cyclic)

April 26, 2006

Object Oriented Design Course

42

### How to improve the rating?



April 26, 2006

Object Oriented Design Course

43

### Profiling

- A profiler is a program that can track the performance of another program
- Used to solve performance problems
  - "How come a simple file viewer take 30 seconds to start, and over 2 minutes to find text in a medium text file?"
- Used to solve memory problems
  - "Why does my text editor take 50MB on startup, and 300MB after a hour of work?"

April 26, 2006

Object Oriented Design Course

44

### Performance Tuning

- How can I make my program faster?
- The 80 / 20 Principle
  - 80% of the time is spent in 20% of the code
  - Key Issue: Find the bottlenecks
- Classic Mistake: Assume the bottlenecks
  - You can't know where they'll be
- Classic Mistake II: Optimize in Advance
  - Start with the right design, then optimize

April 26, 2006

Object Oriented Design Course

45

### Performance Tuning Process

- Step 1: Identify the bottlenecks
  - Use a profiler!
  - Find & measure the bottlenecks
- Step 2: Decide how to solve bottlenecks
  - Make them faster (new algorithm, data str.)
  - Call them less often (caching, lazy execution)
- Step 3: Measure again
  - Only way to make sure improvement happened

April 26, 2006

Object Oriented Design Course

46

### Eclipse Profiler Plugin

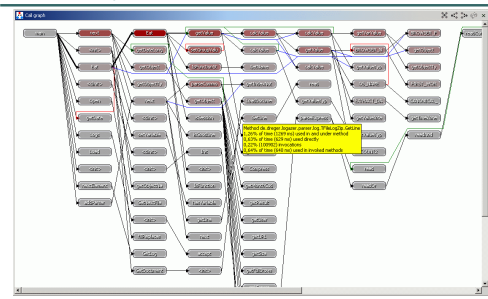
- We'll demonstrate on the (free!) Eclipse Profiler Plugin
- What is tracked
  - CPU
  - Memory usage
  - Number of objects
  - Object graph
  - Call graph

April 26, 2006

Object Oriented Design Course

47

### Call Graph



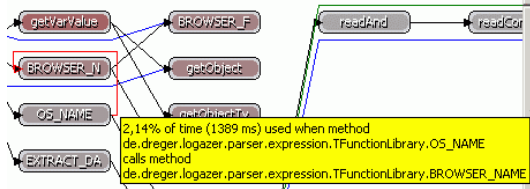
April 26, 2006

Object Oriented Design Course

48



### Call hint



April 26, 2006

Object Oriented Design Course

49

## Callers

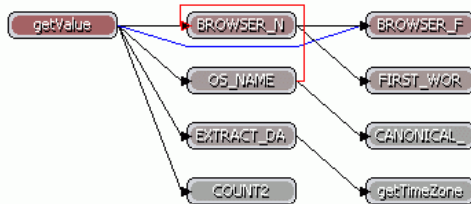


April 26, 2006

Object Oriented Design Course

50

## Callees

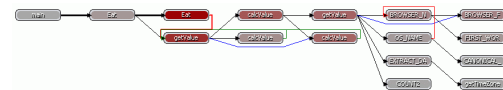


April 26, 2006

Object Oriented Design Course

51

## Callers and callees



April 26, 2006

Object Oriented Design Course

52

## CPU Profiling

- How many invocations were?
- How much time have we spent in a package / class / method?
- Finds the bottlenecks
  - Just sort by time or number of invocations

April 26, 2006

Object Oriented Design Course

53

## Packages

[illegible]

April 26, 2006

Object Oriented Design Course

54

## Classes

[illegible]

April 26, 2006

Object Oriented Design Course

55

## Methods

[illegible]

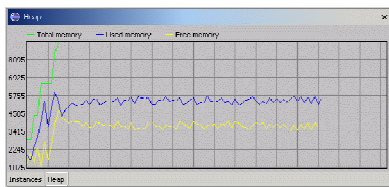
April 26, 2006

Object Oriented Design Course

56

## Memory

- How much memory does the program take?
- Are there memory leaks?

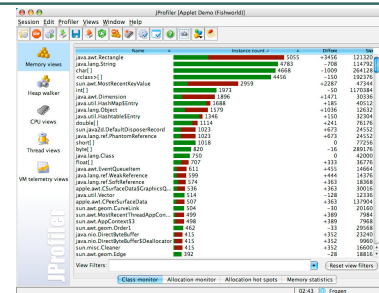


April 26, 2006

Object Oriented Design Course

57

## Memory Monitor



April 26, 2006

Object Oriented Design Course

58

## Profiling - summary

- How does my application behave?
- What are the critical paths?
- Where are the bottlenecks?
- Do I have memory leaks?
  - Java users - you are not exempted!

April 26, 2006

Object Oriented Design Course

59

## Summary

- Personal Productivity Tools
  - Refactoring
  - Static Analysis & Metrics
  - Profilers
- Use them!
- There's more - see [Eclipse Plugins](#)

April 26, 2006

Object Oriented Design Course

60