

Kenneth M. Anderson Software Methods and Tools CSCI 3308 - Fall Semester, 2003

### Credit where Credit is Due

- The material for this lecture is based on content from "Refactoring: Improving the Design of Existing Code" by Martin Fowler
- As such, some of this material is copyright © Addison Wesley, 1999

#### November 10, 2003

© University of Colorado, 2003

# Goals for this lecture

- (Very) Briefly introduce the concept of design
- Introduce Refactoring and cover a few examples

# Software Design (I)

- Software design is the process of creating a software system that meets a set of customer requirements
  - Designs require conceptual integrity
- Traditional software design consists of
  - high-level design (architecture, modules)
  - low-level design (interfaces, algorithms)
    - with these two pieces, implementation is often much simpler than it would be if you start coding from scratch

# Software Design (II)

- Many different techniques to choose from
  - Structural
    - Stepwise Refinement; "Top Down" vs. "Bottom Up"
    - Abstractions used in design are often different from those used in requirements
    - Typically result in procedural solutions that share data structures; the shared data structures is how modules "communicate"
  - Object-Oriented
    - World consists of objects; Thus, systems should consist of objects that "model" their real-world counterparts
    - Objects appear in all phases (requirements; design; implementation)
    - Typically result in "federations" of objects that work together to achieve system functionality; data and algorithms "live" in objects; communication (data sharing) occurs via "message passing"

November 10, 2003 © University of Colorado, 2003	5
--	---

# Software Design (III)

- Good design requires experience
  - also depends on talent (great designers ala Brooks)
- We can't teach experience (you just have to earn it); we can however teach good design techniques
- Example: Refactoring
  - Useful because its focus is on source code not a specific design notation (so you do not need to learn a new notation to learn this technique)

```
November 10, 2003
```

```
© University of Colorado, 2003
```

```
6
```

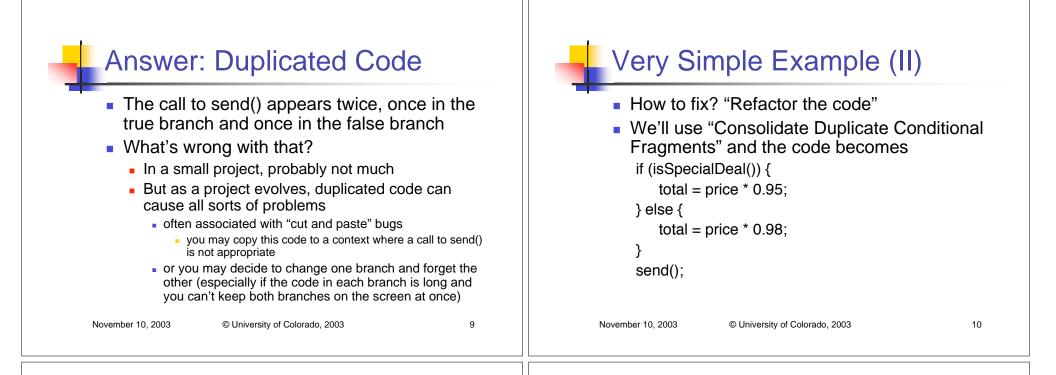
# What is Refactoring

- Refactoring is the process of changing a software system such that
  - the external behavior of the system does not change
    - e.g. functional requirements are maintained
  - but the internal structure of the system is improved
- This is sometimes called
  - "Improving the design after it has been written"

# Very Simple Example (I)

What's wrong with this code?

if (isSpecialDeal()) {
 total = price \* 0.95;
 send()
} else {
 total = price \* 0.98;
 send()
}

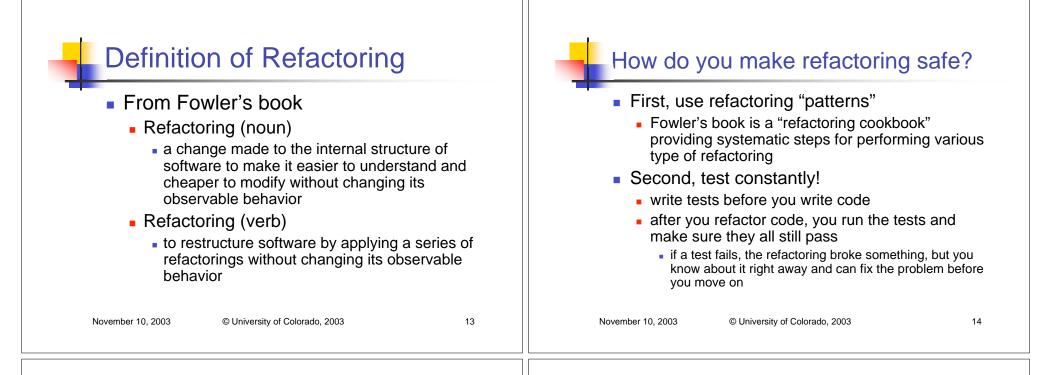


# A Rose is a Rose...

- Why is it so important to give a stuffy sounding name to something so simple?
  - Answer: to improve the "state of practice" in the software development industry
  - As we add standardized vocabulary which all professional developers are required to know; we improve the professionalism of the entire field
  - Refactoring vocabulary is especially important since its improving developer's "design skills"
- Also: some refactorings are NOT simple and giving them a name, makes it easier to discuss the technique with other developers

#### **Benefits of Refactoring**

- The idea behind refactoring is to acknowledge that it will be difficult to get a design right the first time
  - and as a program's requirements change, the design may need to change
  - thus, refactoring provides techniques for evolving the design in small incremental steps
- Benefits
  - Often code size is reduced after a refactoring
  - Confusing structures are transformed into simpler structures
    - thus, these new structures are easier to maintain and understand

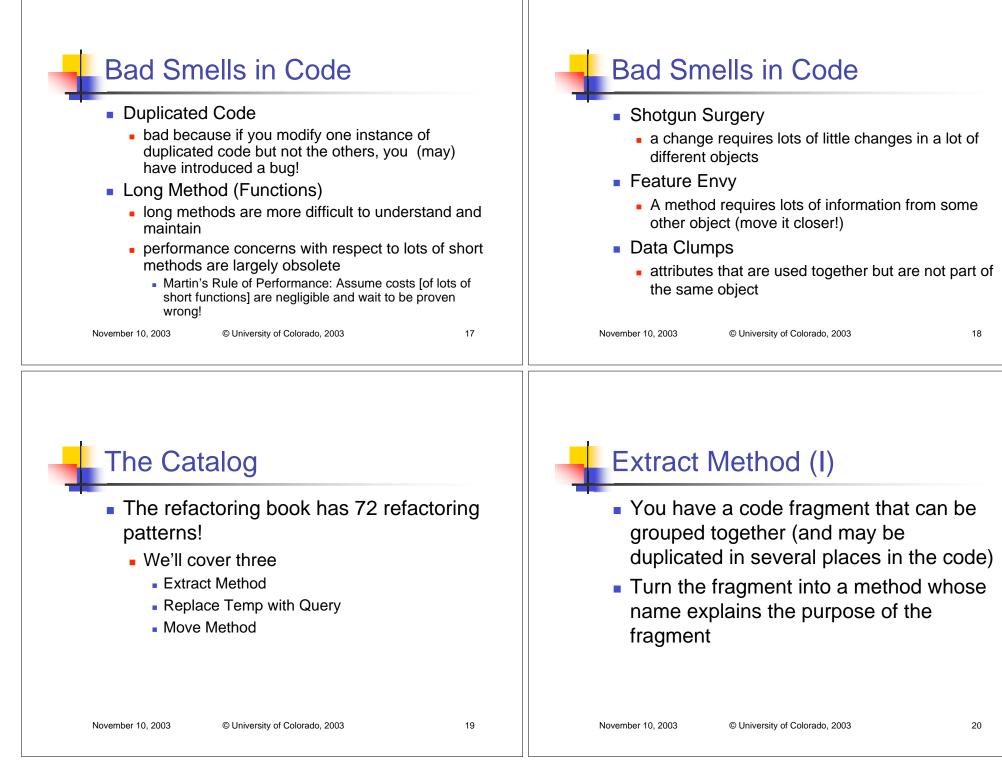


# Why should you refactor?

- Refactoring improves the design of software
  - without refactoring, a design will "decay" as people make changes to a software system
- Refactoring makes software easier to understand
  - because structure is improved, duplicated code is eliminated, etc.
- Refactoring helps you find bugs
  - Refactoring promotes a deep understanding of the code at hand, and this understanding aids the programmer in finding bugs and anticipating potential bugs
- Refactoring helps you program faster
  - because a good design enables progress

# Refactoring: Where to Start?

- How do you identify code that needs to be refactored?
  - Fowler uses an olfactory analogy (attributed to Kent Beck)
  - Look for "Bad Smells" in Code
    - A very valuable chapter in Fowler's book
    - It presents examples of "bad smells" and then suggests refactoring techniques to apply



Extract Method (II)		Replace	e Temp with Que	əry (I)
<pre>void printOwing(double amount) {     printBanner()     //print details     System.out.println("name: " + _name);     System.out.println("amount: " + amount); }</pre>		<ul> <li>You are using a temporary variable to hold the result of an expression</li> <li>Extract the expression into a method; Replace all references to the temp with the expression</li> </ul>		
<pre>void printDetails(double amount) {     System.out.println("name: " + _name);     System.out.println("amount: " + amount); }</pre>		The new other me	r method can then be ethods	used in
November 10, 2003 © University of Colorado, 2003	21	November 10, 2003	© University of Colorado, 2003	22
double basePrice = _quantity * _itemPrice if (basePrice > 1000) return basePrice * 0.95; else	ery (II)	some oth	l is using more data and f er object than its own obj	ect
double basePrice = _quantity * _itemPrice if (basePrice > 1000) return basePrice * 0.95;	ery (II)	<ul> <li>A method some oth</li> <li>Create a the class</li> </ul>	I is using more data and f er object than its own obj new method with a simila it uses most	ect ar body in
double basePrice = _quantity * _itemPrice if (basePrice > 1000) return basePrice * 0.95; else return basePrice * 0.98; 	ery (II)	<ul> <li>A method some oth</li> <li>Create a the class</li> <li>Have the or remove</li> </ul>	l is using more data and f er object than its own obj new method with a simila	ect ar body in old method