Lecture. Refactoring

Dr. Miryung Kim

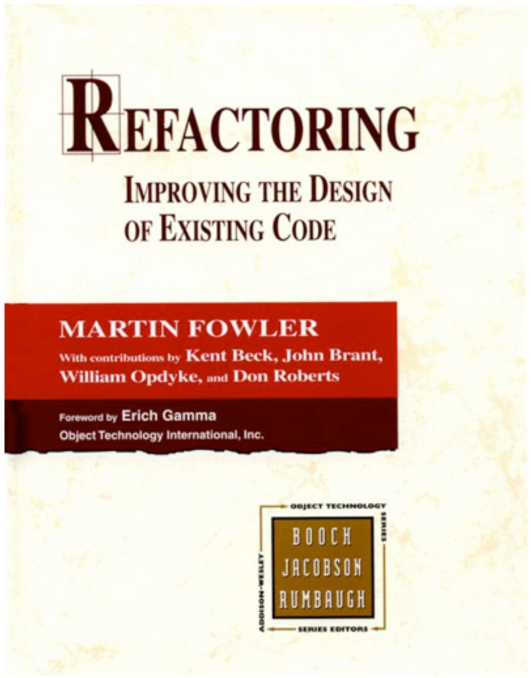
Topics

- Topics for today's lecture
  - What is refactoring?
  - Bad code smells/ When should I refactor code?
  - Refactoring types & transformations
  - Refactoring research projects at SEAL

Announcement

- Refactoring research projects at SEAL
  - A field study of refactoring benefits & challenges at Microsoft
  - SYDIT: example-driven automated refactoring

Refactoring


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH

- semantic-preserving program transformations
- a change made to the internal structure to ~~the structure to~~ make it easier to understand and cheaper to modify without changing its observable behavior

Why do we need Design Patterns?

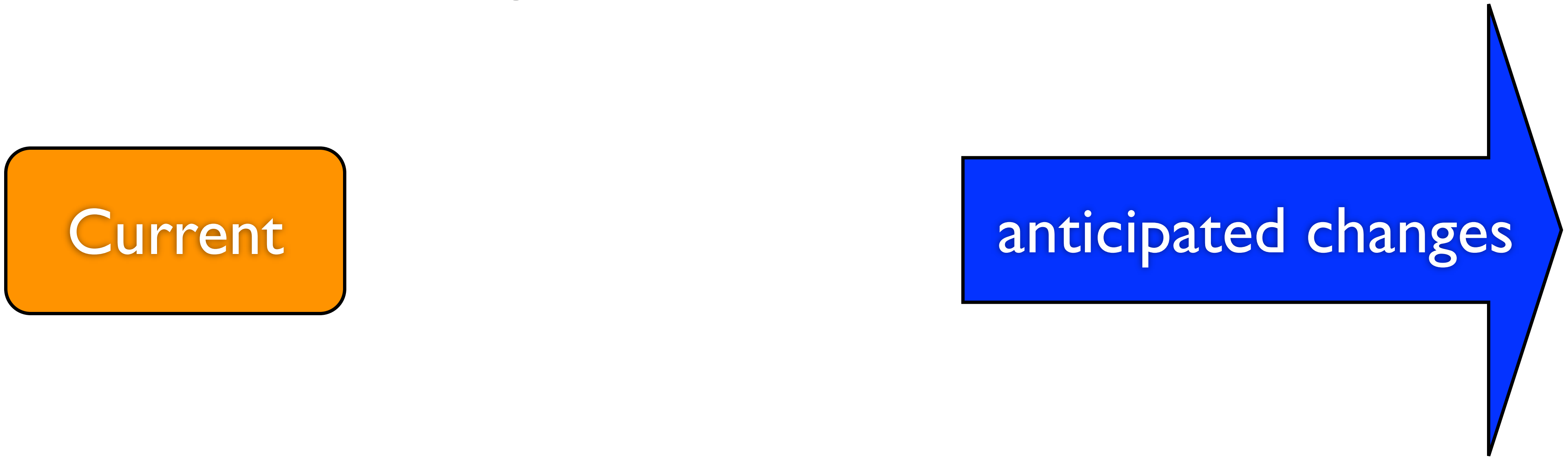
skip

1. Abstract design experience => a reusable base of experience
2. Provide common vocabulary for discussing design
3. Reduce system complexity by naming abstractions => reduce the learning time for a class library / program comprehension

Why do we need Design Patterns?

skip

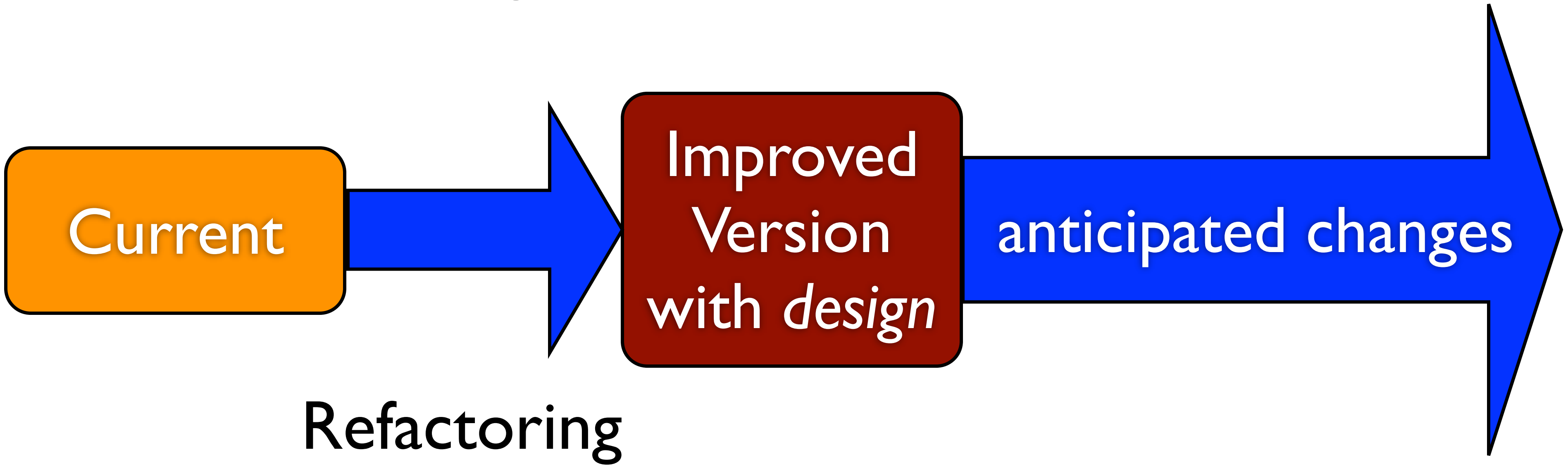
4. Provide a target for the reorganization or refactoring of class hierarchies



Why do we need Design Patterns?

skip

4. Provide a target for the reorganization or refactoring of class hierarchies

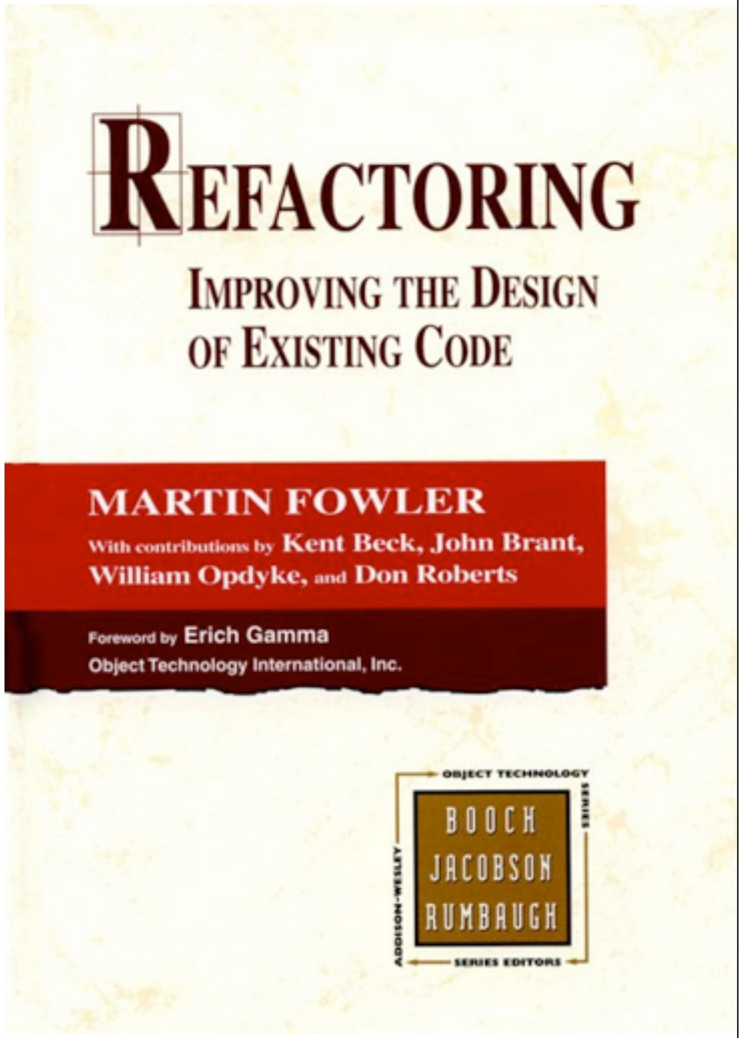


Reasons to Refactor

- Sometimes code degenerates under maintenance, and sometimes the code just wasn't very good in the first place.

Bad Code Smells

- What are reasons to refactor code?
- Fowler termed "code smells" to indicate the symptoms of bad software design


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

What are examples of bad *code smells*?

Bad Code Smells

- Duplicated code
- Long method
- Large class
- Long parameter list
- Divergent change
- *Shotgun surgery*

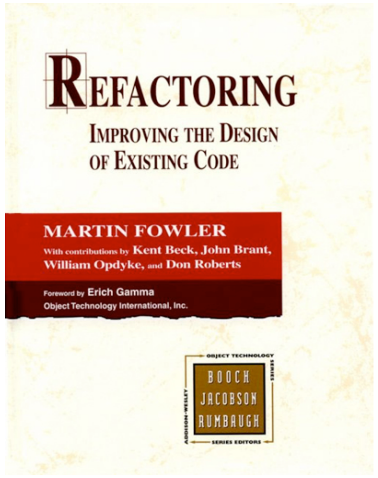
Bad Code Smells

- *Feature envy*
- *Data clumps*
- *primitive obsession*
- *switch statements*
- *parallel inheritance hierarchies*
- *lazy class*

Bad Code Smells

- speculative generality
- temporary field
- message chains
- middle man
- inappropriate intimacy
- alternative classes with different interfaces

Refactoring (Fowler 2000)


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International, Inc.
BOOCH JACOBSON RUMBAUGH

- It is a catalogue of common refactorings in object-oriented programs.
- It is not formally defined (there's no way to check semantics preservation.)
- However, just like a design pattern, it provides a common vocabulary to refer to common refactoring types.

Problem: Divergent Change
Solution: *Extract Class*

- when one class is commonly changed in different ways for different reasons.
  - I have to change mA(), mB(), and mC() every time I get a new database, and mD(), mE(), mF(), and mG() every time there's a new financial instrument.
  - *Extract Class refactoring to separate different concerns*

Shotgun Surgery

- Shotgun surgery is similar to divergent change but the opposite.
  - Divergent change is one class that suffers many kinds of changes, and shotgun survey is one change that alters many classes.
- You have to make a lot of little changes to a lot of different classes.
- Solution: Move Method, Move Field, Inline Class

Feature Envy

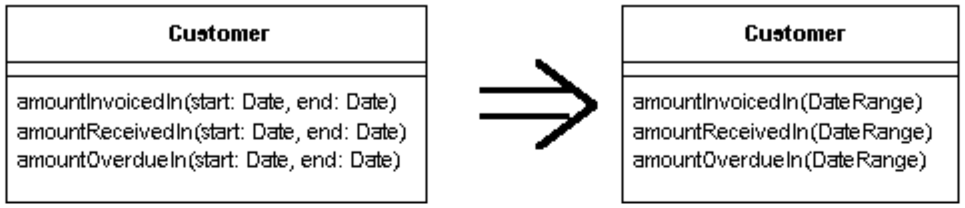
- A method that seems more interested in a class other than the one it actually is in.
- The most common focus of the envy is the data
  - e.g. a method that invokes half-a-dozen getter methods to another object to calculate some value.

Data Clumps

- Bunches of data that hang around together really ought to be made into their own object
- Solutions:
    - Extract class
    - Introduce parameter objects
    - Preserve whole objects

Introduce Parameter Object

You have a group of parameters that naturally go together.
=>Replace them with an object


Customer
amountInvoicedIn(start: Date, end: Date)
amountReceivedIn(start: Date, end: Date)
amountOverdueIn(start: Date, end: Date)
Customer
amountInvoicedIn(DateRange)
amountReceivedIn(DateRange)
amountOverdueIn(DateRange)

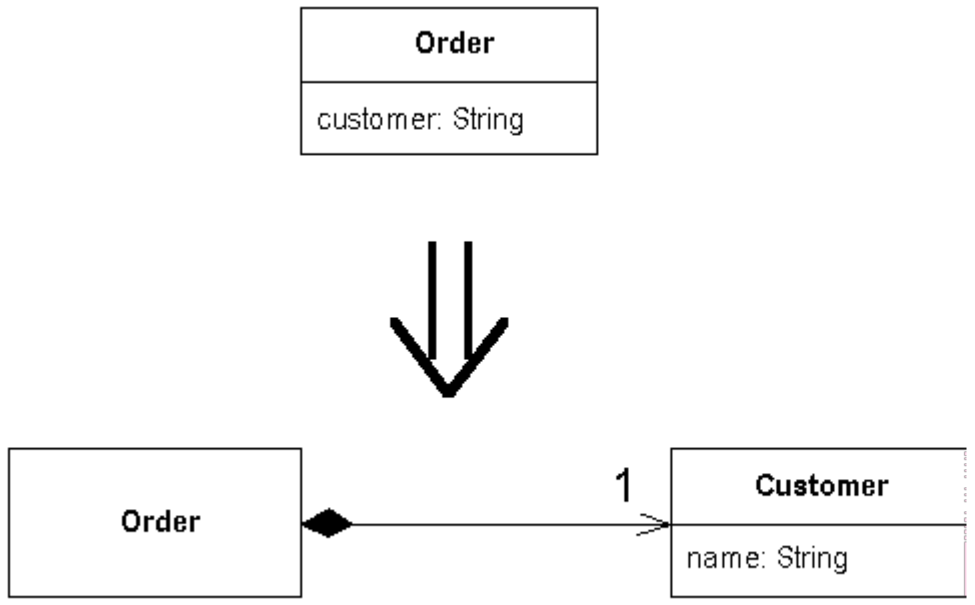
See Handout

Primitive Obsession

- Record types allow you to structure data into meaningful groups
- Primitive types are your building blocks
- Solutions
    - replace data value with object
    - replace type code with class

Replace Data Value with Object

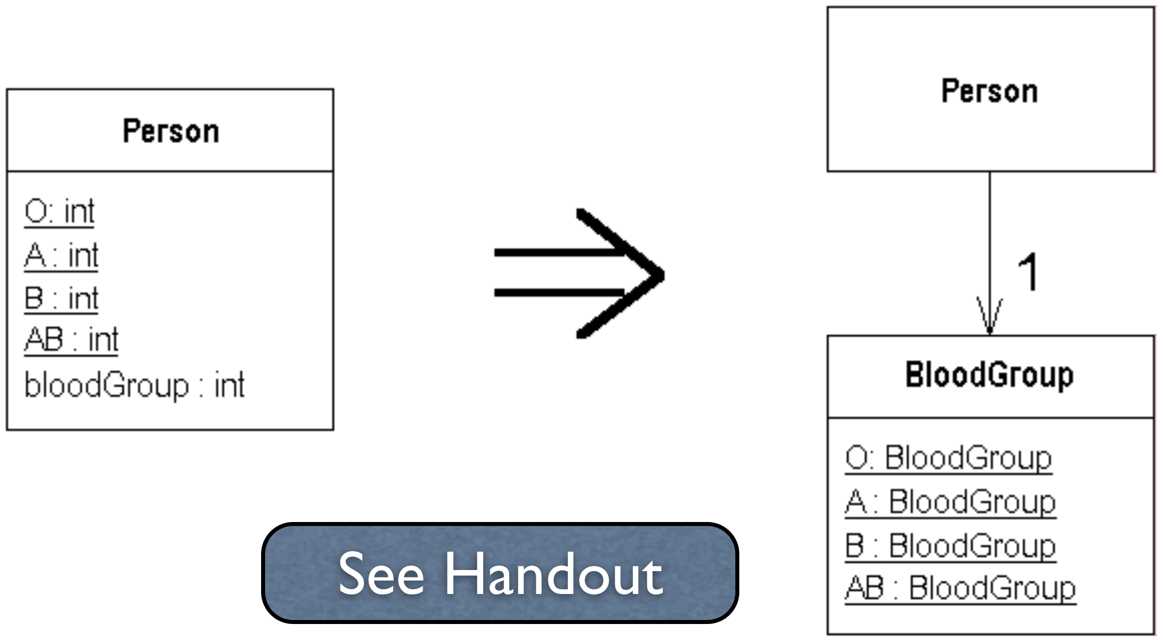
You have a data item that needs additional data or behavior.
Turn the data item into an object.



Replace Type Code with Class

A class has a numeric type code that does not affect its behavior.

=> replace the number with a new class



See Handout

Parallel Inheritance Hierarchies

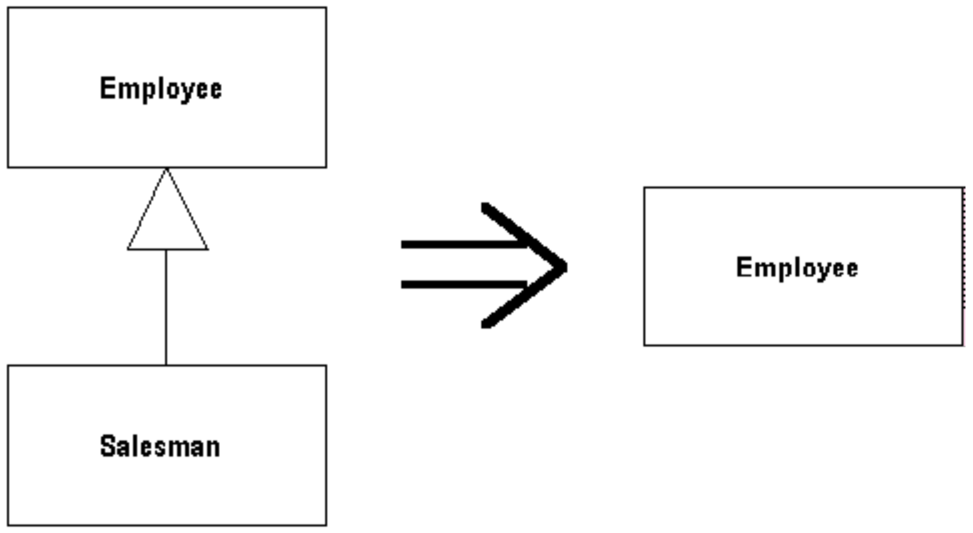
- Parallel inheritance hierarchies is a special case of shotgun surgery.
- Every time you make a subclass of one class, you also have to make a subclass of another.
- Solution: move method or move field

Lazy Class

- Each class you create costs money to maintain and understand.
- A class that isn't doing enough to pay for itself should be eliminated.
- If you have subclasses that aren't doing enough, try to use *Collapse Hierarchy.*
- Nearly useless components should be subjected to *Inline Class*

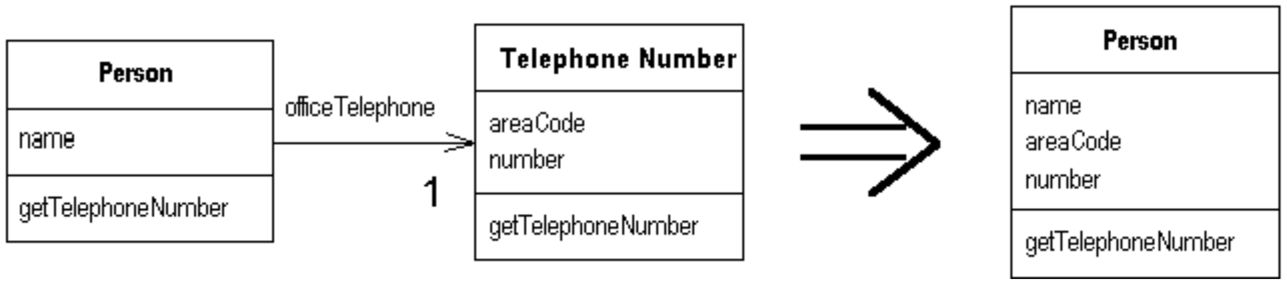
Collapse Hierarchy

A superclass and subclass are not very different. Merge them together



Inline Class

A class isn't doing very much
=> Move all its features into another class and delete it



Speculative Generality

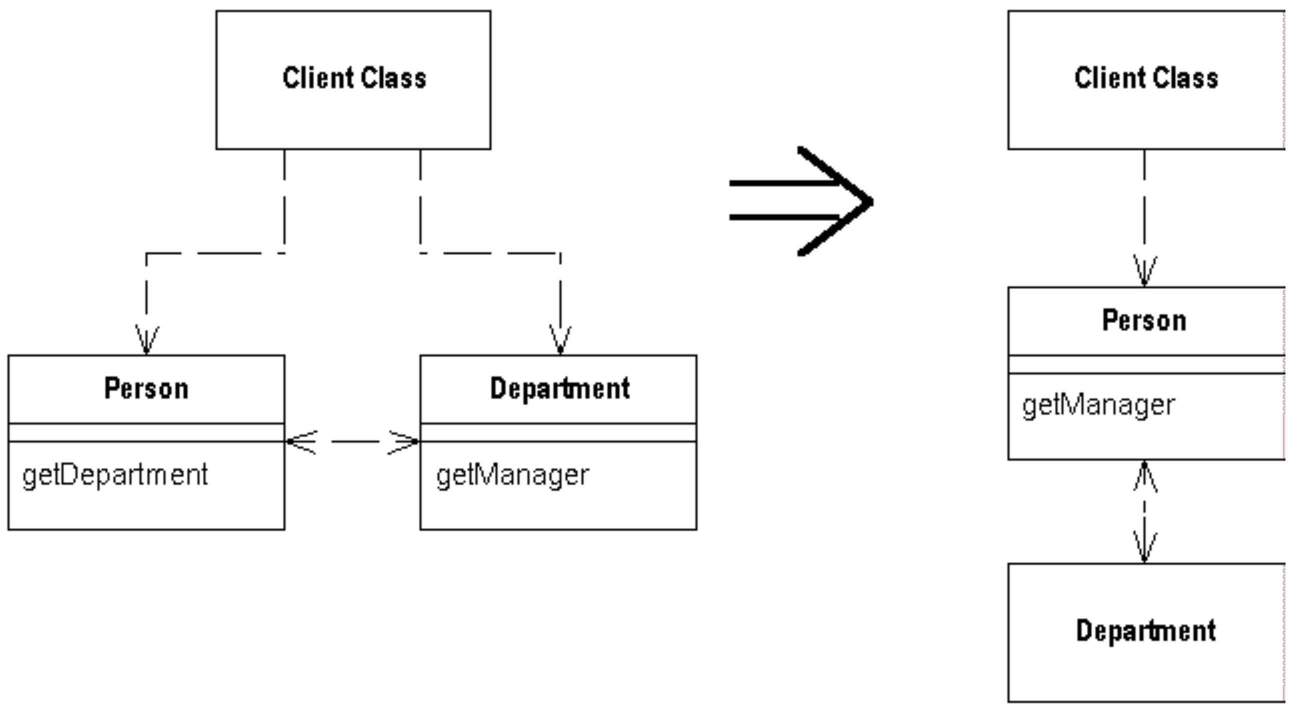
- "Oh, I think we need the ability to this kind of thing someday."
- If you have abstract classes that aren't doing much, use *Collapse Hierarchy.*
- Unnecessary delegation can be removed with *Inline class*. Methods named with odd abstract names should be brought down to earth with *Rename Method.*

Inappropriate Intimacy

- Sometimes classes become far too intimate and spend too much time delving in each other's private data
- *Change Bidirectional Association to Uni-direction.*
- If the classes do have common interests, use *Extract Class* to put the commonality in a safe place.
- *Hide Delegate* to let another class act as go-between.

Hide Delegate

A client is calling a delegate class of an object.
=> Create methods on the server to hide the delegate



Replace Conditional with Polymorphism

- You have a conditional that chooses different behavior depending on the type of an object.
- *Move each leg of the conditional into an overriding method in a subclass. Make the original method abstract.*

See Handout

Refactoring Categories

- Data-Level Refactorings
- Statement-Level Refactorings
- Routine-Level Refactorings
- Class Implementation Refactorings
- Class Interface Refactorings
- System Level Refactorings

Eclipse Demo

skip

Refactoring Safely

- Save the code you start with
- Keep refactorings small
- Do refactorings one at a time
- Make a list of steps you intend to take
- Make a parking lot--- for changes that aren't needed immediately, make a "parking lot."

Refactoring Safely

- Make frequent checkpoints
- Use your compiler warnings
- Retest
- Add test cases
- Review the changes
- Adjust your approach depending on the risk level of the refactoring

Recap

- Bad code smells indicate the symptoms of poor design.
- Fowler's catalog lists code transformations to address individual bad code smells.
- It is important to apply refactoring safely and to validate the correctness of refactoring.

Research Projects at SEAL

- A field study of refactoring benefits & challenges at Microsoft
- Sydit: Learning program transformations from an example