Refactoring
March 6, 2004

Lots of material taken from Fowler, Refactoring: Improving the Design of Existing Code

Administrivia

• Project deadline Saturday, March 13
  – Usual late rules apply

• Minor (but important) project changes:
  – set_field methods should return the value of their parameter
    • Helps with EncapsulateField y in x = y = 1
  – Can ignore +=, -=, *=, /=

Switch Statements

• Usually not necessary in delegation-based OO programming

• Replace Type Code with State/Strategy
  – Define a class hierarchy, a subclass for each type code

• Replace Conditional with Polymorphism
  – Call method on state object to perform the check; switching is based on dynamic dispatch

Replace Type Code with State/Strategy

Replace Conditional with Polymorphism

Duplicated Code

• The same expression used in different places in the same class
  – Use Extract Method to pull it out into a method

• The same expression in two subclasses sharing the same superclass
  – Extract Method in each, then
    – PullUp method into parent

• Duplicated code in two unrelated classes
  – Extract Class - Break a class that does many things into smaller classes
Pull Up Method

- Might do other refactorings if methods don’t quite match
- What if doesn’t appear in all subclasses?

Extract Class

- How do we decide what goes in new class?
- Do fields still need to be accessed in orig class?

Long Parameter List

- Lots of parameters occlude understanding
- Replace Parameter with Method
  - Remove method parameters and instead use some other way to get the parameter value (e.g., method call)
- Introduce Parameter Object
  - Group parameters that go together into a container object

Replace Parameter with Method

- discountedPrice can call getDiscount() itself
- double basePrice = num * price;
  double discount = getDiscount();
  double finalPrice = discountedPrice(basePrice, discount); 

Introduce Parameter Object

Divergent Change

- One class is commonly changed in different ways for different reasons
  - To add a new database, change these three methods
  - To add a new financial currency, change these four
- Suggests maybe this shouldn’t be one object
- Apply Extract Class to group together variations
Shotgun Surgery

- Every time I make change X, I have to make lots of little changes to different classes
  - Opposite of Divergent Change

- Move Method
- Move Field
  - Switch field from one class to another
- Inline Class
  - A class isn’t doing very much, so inline its features into its users (reverse of Extract Class)

Other Bad Smells

- Data Clumps
  - Objects seem to be associated, but aren’t grouped together
- Primitive Obsession
  - Reluctance to use objects instead of primitives
- Parallel Inheritance Hierarchies
  - Similar to Shotgun Surgery; every time we add a subclass in one place, we need to add a corresponding subclass to another

Other Bad Smells (cont’d)

- Lazy Class
  - A class just isn’t useful any more
- Speculative Generality
  - “Oh, I think we need the ability to do this kind of thing someday.”
- Temporary Field
  - Instance variable only used in some cases. Confusing to figure out why it’s not being set everywhere.

Other Bad Smells (cont’d)

- Message Chains
  - Long sequences of gets or temporaries; means client is tied to deep relationships among other classes
- Middle Man
  - Too much delegation. If a class delegates lots of its functionality to another class, do you need it?
- Inappropriate Intimacy
  - Classes rely on too many details of each other

Other Bad Smells (cont’d)

- Alternative Classes with Different Interfaces
  - Methods do the same thing but have different interfaces
- Incomplete Library Class
  - Library code doesn’t do everything you’d like
- Data Class
  - Classes that act as “structs,” with no computation
- Refused Bequest
  - Subclass doesn’t use features of superclass

Other Bad Smells (cont’d)

- Comments!
  - If code is heavily commented, either
    - It’s very tricky code (e.g., a hard algorithm), or
    - The design is bad, and you’re trying to explain it
  - “When you feel the need to write a comment, first try to refactor the code so that any comment becomes superfluous.”
Refactoring with Tools

- Many refactorings can be performed automatically
- This reduces the possibility of making a silly mistake
- Eclipse provides support for refactoring in Java
  - http://www.eclipse.org

More information

- Textbook: Refactoring by M. Fowler
- Catalog of refactorings:
- Refactoring to patterns