ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 27 — March 19, 2013

Patrick Lam *version 1*

Goal of Refactoring. Refactoring aims to improve *non-functional properties* of the code; performance may get better, worse, or stay the same. The idea is to get easier-to-maintain code. For instance, you might split one loop into two separate loops, each of which does one logically sensible thing.

List of Refactorings. I've augmented Wikipedia's non-exhaustive list of refactorings[1]. These refactorings are organized differently than the list you saw last time.

- Techniques that allow for more abstraction:
  - *Encapsulate Field*: force code to access the field with getter and setter methods.
  - *Generalize Type*: create more general types to allow for more code sharing.
  - *Replace conditional with polymorphism*: use inheritance and virtual dispatch instead of a conditional.
- Techniques for breaking code apart into more logical pieces:
  - *Extract Method*: as seen above, pull out part of a larger method into a new method.
  - *Extract Class*: moves code from an existing class into a new class.
- Techniques for integrating code that's needlessly spread apart:
  - *Inline Method*: integrate a copy of the body of a method into its calling method.
  - *Inline Class*: put all of the fields and methods of a class into another class and erase the original.
- Techniques for improving names and location of code:
  - *Move Method/Field*: move to a more appropriate class or source file.
  - *Rename Method/Field*: changing the name into a new one that better reveals its purpose.
  - *Pull Up*: in OOP, move to a superclass.
  - *Push Down*: in OOP, move to a subclass.

More examples

We'll continue by talking about the refactorings in more detail; perhaps this will help you understand the goals of refactoring, so that you can do it in your own code.

[1] `http://en.wikipedia.org/wiki/Code_refactoring`, accessed March 22, 2011.

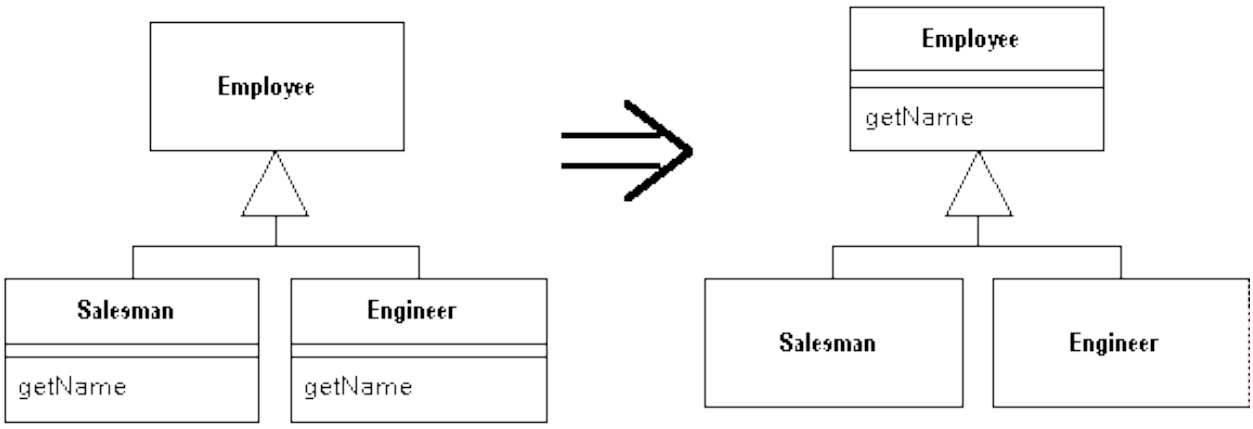
Encapsulate field. A common Java idiom is to have private fields, public accessor methods, and replace all field reads `x.foo` and writes `x.foo = y`, respectively, with calls to `x.getFoo()` and `x.setFoo(y)`.

The goal of this refactoring is to enhance the modularity of the code. Only the class that defines `x` is allowed to read or write `x` directly.

One implication is that you can change the way you store the internal data inside a class, and it won't affect any other classes. For instance, you can have a `Point` class which you initially implement using x/y coordinates, and then transparently change it to using polar coordinates without having to change any of the users of the `Point`. Or, as in Assignment 8, you can rewrite the `CardDatabase` and any of its users don't need to care.

Not every field needs to be encapsulated, though. For instance, if you have a class which represents an RGB colour, it's probably OK to have public fields for the red, green and blue fields.

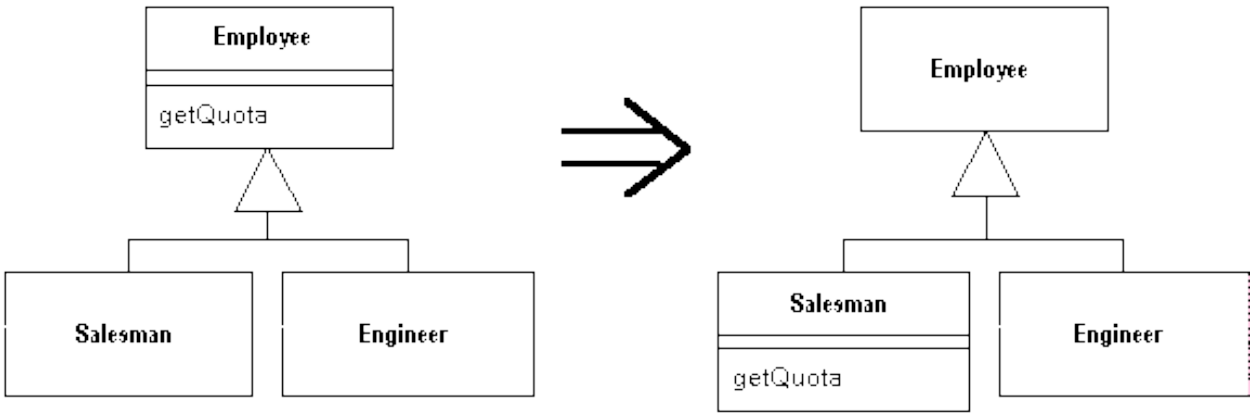
Pull Up Method. This refactoring is an example of a generalization. Consider the following UML class diagrams[2]:



What we can see here is that the two subclasses `Salesman` and `Engineer` both have a common method `getName`, with identical behaviour. It makes sense to move the method to the common superclass `Employee`.

Push Down Method. Conversely, sometimes it makes sense to push methods down to subclasses[3]:

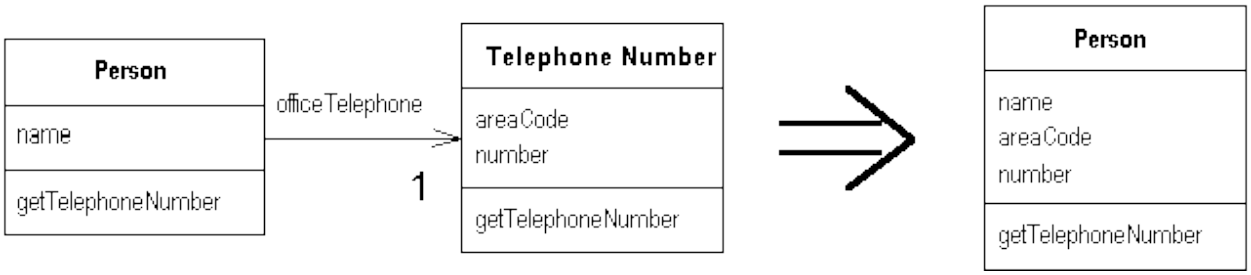
[2] `http://www.refactoring.com/catalog/pullUpMethod.html`, accessed July 4 '11.
[3] `http://www.refactoring.com/catalog/pushDownMethod.html`, accessed July 4, '11.



In this case, the original design was flawed: `Engineer`s don't have quotas, only `Salesman`. So we improve the design by putting `getQuota` where it should be.

I didn't show you the code, only the UML, but the transformation to the code should be obvious. Note that in both cases, it's not supposed to change the behaviour of the program, only improve its design.

Inline Class. Here's one final refactoring example[4]. In this case, we get rid of helper classes that aren't that useful.



The UML transformation corresponds to a code transformation where we fold the `areaCode` and `number` fields, along with the `getTelephoneNumber` method, into the containing `Person` class. Maybe we found that no one else was using the `Telephone Number` class. Be sure to delete the class after you've inlined it.

[4] `http://www.refactoring.com/catalog/inlineClass.html`, accessed July 4, '11.