Refactoring: Improving the Design of Existing Code

CS 4354
Summer II 2014

Jill Seaman

What is Refactoring?

- Refactoring: disciplined technique for changing a software system: altering its internal structure without changing its external behavior
  - To improve readability.
  - To improve structure.
  - Reduce complexity.
  - Easier to modify in the future
- No added functionality!!
- Preventative maintenance: reduces future (or current) maintenance costs

Refactoring process

- Required during Iterative Development to maintain the quality of the code as new code is added.
  - ✦Ongoing process, from start of development.
  - ✦Applied on a small scale
  - ✦Avoids structure degradation from the start
- Steps of the process:
  1. write new code and new tests
  2. test code using **all** the tests, make sure all tests pass
  3. refactor the code (in a series of steps)
  4. test code using all the tests. If any tests fail, repair or rollback changes.
  5. repeat from step 1 or step 3

Refactoring process

- **Refactoring** (noun): a <small> change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- **Refactor** (verb): to restructure software by applying a series of refactorings without changing its observable behavior.
- I'm often asked about how it should be scheduled. Should we allocate two weeks every couple of months to refactoring?
- In my view refactoring is not an activity you set aside time to do. Refactoring is something you do all the time in little bursts. You don't decide to refactor, you refactor because you want to do something else, and refactoring helps you do that other thing.

from: sourcemaking.com/refactoring

Some Refactorings

- Rename Method/Field/Class/Variable: change the name and all references to it.
- Extract Method: Replace some inline code with a call to a (new) method containing that code.
- Encapsulate Field: Make a public field private, generate getters and setters, replace references to the field with calls to these.
- Move Method/Field: Move element to the new class, use delegation to replace references to these elements in the original class.
- Pull Up Method/Field: If two subclasses use the same method, move the method to the superclass.
- Push Down Method/Field: If a method is used for only some of the subclasses, move it to those subclasses.

Where to apply refactoring (bad smells)

- Duplicate code
  - ✦Same or very similar code found at various places in a program.
  - ✦[Extract method]: put similar code into a single method/function
- Long method
  - ✦Long methods are difficult to understand, modify.
  - ✦Redesign as many shorter methods [Extract method]
- Switch (case) statements
  - ✦Multiple switch statements with same cases.
  - ✦Make subclasses, move each case into appropriate subclass.
  - ✦[Replace Type Code with Subclasses]
- Data clumping
  - ✦The same group of items occur in several places in a program.
  - ✦Replace with an class that encapsulates all of the data [Extract Class]
- Speculative generality
  - ✦Unused parameters, classes, included “just in case”. [Remove Parameter]

Refactoring example

- Note: classes are incomplete: constructors, getters/setters are not shown.
- What is the bad smell here?

```
class Employee
   double monthlySalary;
   double commission;
   double bonus;
   int getType() { … }
   int payAmount() {
      switch (getType()) {
         case ENGINEER:
            return monthlySalary;
         case SALESMAN:
            return monthlySalary + commission;
         case MANAGER:
            return monthlySalary + bonus;
         default:
            throw new RuntimeException("Incorrect Employee");
      }
   }
```


Refactoring example

- Move cases into (new) subclasses [Replace Type Code with Subclasses]:

```
class Employee…
   double monthlySalary;
   double commission;
   double bonus;
   int payAmount();
}
class Engineer : Employee
   int payAmount() {
      return monthlySalary;
   }
class Salesman : Employee
   int payAmount() {
      return monthlySalary + commission;
   }
class Manager : Employee
   int payAmount() {
      return monthlySalary + bonus;
   }
```

- Now we can clean this up further using another refactoring.

Refactoring example

- [Push down field]: when a field is used only by some subclasses

```
class Employee… {
   double monthlySalary;
   int payAmount();
}
class Engineer : Employee {
   int payAmount() {
      return monthlySalary;
   } }
class Salesman : Employee  {
   double commission;
   int payAmount() {
      return monthlySalary + commission;
   } }
class Manager : Employee  {
   double bonus;
   int payAmount() {
      return monthlySalary + bonus;
   }
}
```


Another example from Assignment 2

- Recall the Products sold by the Online Store
- Now the owner is adding a new product type: Electronics.
- Exercise: create a new subclass of Product for the Electronics, then apply some refactorings to clean up your code.
- Use a new refactoring [Extract Superclass].

Product Type	Shipping credit	Commission
Movie (dvd)	$2.98	12% of sale price
Book	$3.99	15% of sale price
Toys	$4.49 + .50/lb	15% of sale price
Electronics	$4.49 + .50/lb	8% of sale price