Design Defects and Restructuring

Lecture 6

Sat, Oct 23, 2021

Refactoring

- Refactoring is the process of changing a software system
  - It does not alter the external behavior of the code
  - It improves its internal structure
- It is a disciplined way to clean up code that minimizes the chances of introducing bugs
- When you refactor, you are improving the design of the code after it has been written

Refactoring

- **Refactoring**: A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior
- **Refactor**: To restructure software by applying a series of refactorings without changing its observable behavior

Refactoring Tips

- When you find you must add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature
- Before you start refactoring, check that you have a solid suite of tests; these tests must be self-checking
- Refactoring changes the programs in small steps; if you make a mistake, it is easy to find the bug
- **Any fool can write code that a computer can understand; good programmers write code that humans can understand**

Refactoring

- Why Should You Refactor?
  - Refactoring Improves the Design of Software
  - Refactoring Makes Software Easier to Understand
  - Refactoring Helps You Find Bugs
  - Refactoring Helps You Program Faster
- When Should You Refactor?
  - Refactor When You Add Function
  - Refactor When You Need to Fix a Bug
  - Refactor As You Do a Code Review

Refactoring

- Problems with Refactoring
  - Databases
  - Changing Interfaces
  - Design Changes That Are Difficult to Refactor
  - When Shouldn't You Refactor?
- Refactoring and Design
- Refactoring and Performance
- Where Did Refactoring Come From?

Bad Smells in Code

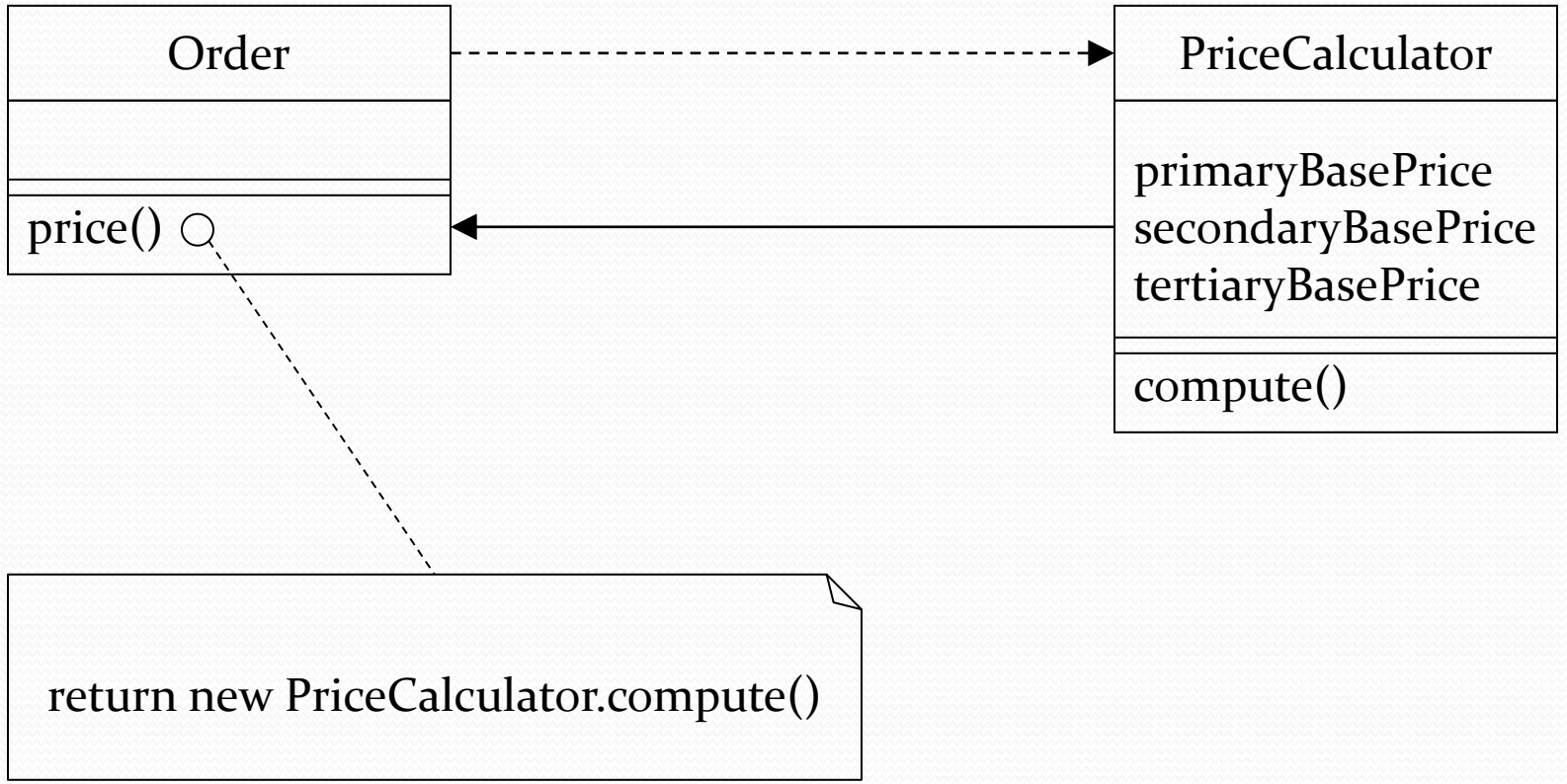
- Duplicate Code
- Long Method
- Large Class
- Long Parameter List
- Divergent Change
- Shotgun Surgery
- Feature Envy
- Data Clumps
- Primitive Obsession
- Switch Statements
- Parallel Inheritance Hierarchy
- Lazy Class
- Speculative Generality
- Temporary Field
- Message Chains
- Middleman
- Inappropriate Intimacy
- Alternative Classes with Different Interfaces
- Incomplete Library Class
- Data Class
- Refused Bequest
- Comments

Extract Method

- You have a code fragment that can be grouped together
- *Turn the fragment into a method whose name explains the purpose of the method*
- ```
  void printOwing(double amount) {
       printBanner();
       //print details
       System.out.println("name:" + _name);
       System.out.println("amount" + amount);
  }
  ```
  ```

Extract Method (Continued)

```
void printOwing(double amount) {
     printBanner();
     printDetails(amount);
}
void printDeatils(double amount) {
     System.out.println ("name:" + _name);
     System.out.println ("amount" + amount);
}
```

Inline Method

- A method's body is just as clear as its name
- *Put the method's body into the body of its callers and remove the method*
- ```
  int getRating() {
       return (moreThanFiveLateDeliveries()) ? 2 : 1;
  }

  boolean moreThanFiveLateDeliveries() {
       return _numberOfLateDeliveries > 5;
  }
  ```
- ```
  int getRating() {
       return (_numberOfLateDeliveries > 5) ? 2 : 1;
  }
  ```

Inline Temp

- You have a temp that is assigned to once with a simple expression, and the temp is getting in the way of other refactorings
- *Replace all references to that temp with the expression*
- ```
  double basePrice = anOrder.basePrice();
  return (basePrice > 1000)
  ```
- ```
  return (anOrder.basePrice() > 1000)
  ```

Replace Temp with Query

- You are using a temporary variable to hold the result of an expression
- *Extract the expression into a method. Replace all references to the temp with the expression. The new method can then be used in other methods*
- ```
  double basePrice = _quantity * _itemPrice;
  if (basePrice > 1000)
       return basePrice * 0.95;
  else
       return basePrice * 0.98;
  ```
- ```
  if (basePrice() > 1000)
       return basePrice() * 0.95;
  else
       return basePrice() * 0.98;
  double basePrice() {
       return _quantity * _itemPrice;
  }
  ```

Introduce Explaining Variable

- You have a complicated expression
- *Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose*
- ```
  if((platform.toUpperCase().indexOf("MAC") > -1) &&
  (browser.toUpperCase().indexOf("IE") > -1) &&
  wasInitialized() && resize > 0 ) {
       // do something
  }
  ```
- ```
  final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
  final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;
  final boolean wasResized = resize > 0;
  if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {
       // do something
  }
  ```

Split Temporary Variable

- You have a temporary variable assigned to more than once, but is not a loop variable nor a collecting temporary variable.
- *Make a separate temporary variable for each assignment.*
- ```
  double temp = 2 * (_height + _width);
  System.out.println (temp);
  temp = _height * _width;
  System.out.println (temp);
  ```
- ```
  final double perimeter = 2 * (_height + _width);
  System.out.println (perimeter);
  final double area = _height * _width;
  System.out.println (area);
  ```

Remove Assignments to Parameters

- The code assigns to a parameter
- *Use a temporary variable instead*
- ```
  int discount(int inputVal, int quantity, int yearToDate) {
      if (inputVal > 50) inputVal -= 2;
  ```
- ```
  int discount(int inputVal, int quantity, int yearToDate) {
      int result = inputVal;
      if (inputVal > 50) result -= 2;
  ```

Replace Method with Method Object

- You have a long method that uses local variables in such a way that you cannot apply Extract Method
- *Turn the method into its own object so that all the local variables become fields on that object. You can then decompose the method into other methods on the same object*
- ```
  class Order...
  double price() {
       double primaryBasePrice;
       double secondaryBasePrice;
       double tertiaryBasePrice;
  // long computation;
  ...
  }
  ```
  ```

Replace Method with Method Object (Continued)



Substitute Algorithm

- You want to replace an algorithm with one that is clearer
- *Replace the body of the method with the new algorithm*
- ```
  String foundPerson(String[] people) {
      for (int i = 0; i < people.length; i++) {
          if (people[i].equals ("Don"))
              return "Don";
          if (people[i].equals ("John"))
              return "John";
          if (people[i].equals ("Kent"))
              return "Kent";
      }
      return "";
  }
  ```
  ```

Substitute Algorithm (Continued)

- ```
  String foundPerson(String[] people){
       List candidates = Arrays.asList(new String[] {"Don", "John", "Kent"});
       for (int i = 0; i < people.length; i++)
              if (candidates.contains(people[i]))
                     return people[i];
       return "";
  }
  ```
  ```