Refactoring by Examples: Coding Horrors and Remedies

Course of Software Engineering II

A.A. 2010/2011

Valerio Maggio, PhD Student
Prof. Sergio Di Martino

Before to start..

- Tools:
  - Eclipse IDE 3.6
    - eclipse.org
  - IBM Rational Software Architect
    - Ibm.com/downloads

Example

- Sample Program to print out a statement of customer's charges at a video store.
- Let's see the UML Class Diagram...
- ... and read the Source Code
  - Program Comprehension!

Starting Point

- Impressions about the design of the Program
  - `statement()` ?
    - Long Statement
  - [Requirement]
    Define `htmlStatement()`
- Solutions ?
  - Copy & Paste
- What happens when *Charging Rules* Change?
  - Change both methods

What to do?

- Common feeling:
  Don't touch method `statement`
- *Old engineering adage:*
  “If it ain't broke, don't fix it”
- It's not broken but is does hurt!
- **Refactoring**
  - You have to add a feature and Program's code is not structured in a convenient way!

Step 1: Extracting Amount Calculation

- First step: Aided...
- Obvious target: *long method* `statement()`
- What refactoring solution do you suggest?
- **Refactorings:**
  - **Extract Method**
  - **Change Method Signature**
  - **Rename Variables**

Step 2: Amount calculation

- Let's look at the refactored code
- `amountOf()` method
- What's wrong in your opinion?
- **Refactoring:**
  - **Move Method**

Step 3: Refinements

- Let's go back to `Customer.statement()`
- Next thing that strikes me is:
  - `thisAmount`
- **Refactoring:**
  - **Best Practice: Replace a temp with a query**
  - **Inline Refactoring**

Step 4: Extracting Renter Points

- Next step: Do similar thing for frequent renter point
- Again, What refactoring do you suggest?
- **Refactoring:**
  - **Extract Method**

Step 4.1: Refinement

- Let's look at the refactored code
- `frequentRenterPoints()` method
- Any ideas?
- **Refactoring:**
  - **Best Practice: Replace Temp with Query**
  - **Move Method**

Step 4.2: Another Refinement

- We can take all the refactored code and replace all temporary variables with queries
- Local Variables become Method Calls
- Let's see how...

Step 4.2: Observations

- More refactoring reduce the amount of code but this one increases it!
- That's because Java requires a lot of statements to set up a summing loop.
  - Java Idiom
  - Arises the needs of Java Closures
  - **http://martinfowler.com/bliki/Closure.html**
- **Performances?**

Step 5: `htmlStatement()`

- Now we are able to implement the **new** method `htmlStatement()`
- Let's see the source code...

Step 6: Rental Calculation

- Change the target:
  - `Rental Class`
- Focus on:
  - `Rental Calculation`
- What's wrong?
- **Refactoring:**
  - **Move Method**
  - **Extract Method**

Step 7: Movie Class

- Let's look at Movie Class UML
- Movie has:
  - Three Constants!
  - What about constructors?
- **Refactoring:**
  - **Extract Constructors**

Step 8: At last....

- We have different types of Movies
- So we have different ways of answering the same question
- This sounds like a job for... ?
  - **Subclasses and Inheritance**

State Pattern

- The **State Pattern** is a *behavioral software* design pattern.
- This pattern is used in computer programming to represent the state of an object.
- This is a clean way for an object to partially change its type at runtime [Gang of Four]

Final Thoughts

- Hope this simple examples gives you the feeling of what refactoring is like.
- Used Techniques:
  - Moving Behaviour, Extract Method
  - Replacing case statements
- Improve responsibilities distribution
- Facilitate code maintenance
- Most important lesson: Rhythm of Refactoring
  - Test, small change, test, small change, test, ….

References

- [Fowler]
  - Fowler M with Scott K, UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, Reading MA, 1997
- [Gang of Four]
  - Gamma E, Helm R, Johnson R, and Vlissides J, Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley, Reading MA, 1995
- www.cs.toronto.edu/~arnold/407/06s/assignments/02/fowler04refactoring.pdf