School of Computing
FACULTY OF ENGINEERING & PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

COMP5911M
Advanced Software Engineering

5: Refactoring Case Study

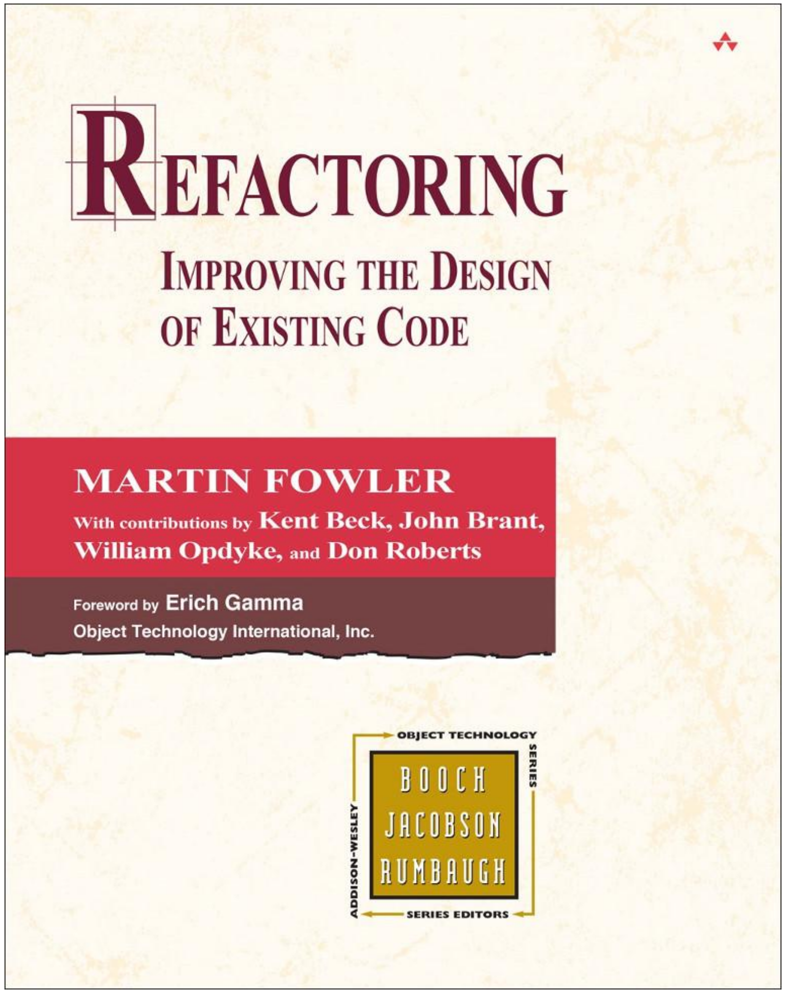
Nick Efford

https://comp5911m.info

Objectives

- To introduce refactoring via a case study
- To get a feel for what the refactoring process is like

‘The Bible’



REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

Foreword by Erich Gamma
Object Technology International, Inc.

OBJECT TECHNOLOGY SERIES

BOOCH JACOBSON RUMBAUGH

ADDISON-WESLEY

SERIES EDITORS

Published 18 years ago!

2nd edition came out a couple of years ago, but is JavaScript-based…

Our Approach

- Following Fowler, Chapter 1:
    - "It is with examples that I can see what is going on"
    - Aim is to provide a sense of what the process is like, then discuss principles next time
- Code example is very similar to Fowler's
- We will discuss the code and work through the first few refactorings here
- … then you finish off the process in Exercise 3

Car Rental Example

UNIVERSITY OF LEEDS

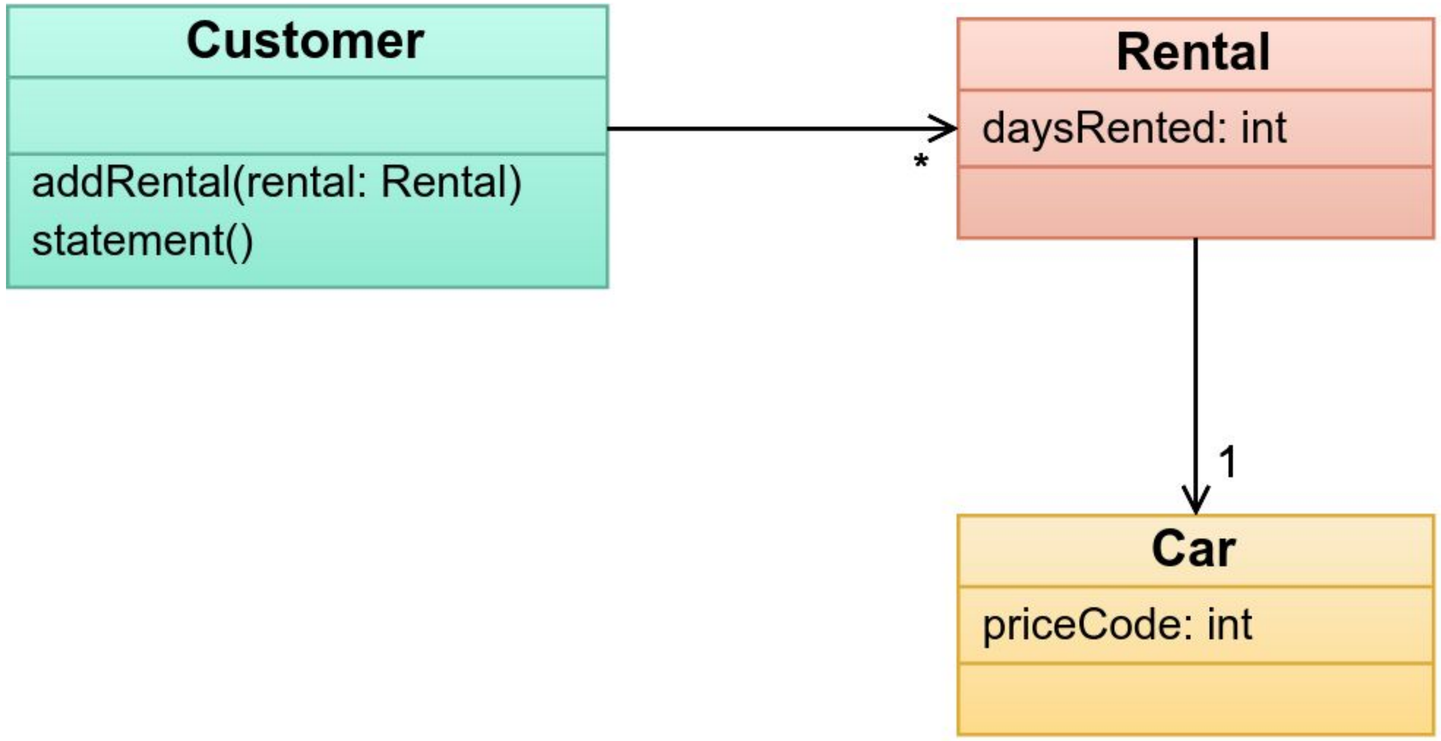
- `Car` class stores model of car and its **price code**
  - Different price codes (and therefore rental costs) for Standard, New Model and Luxury cars
- `Rental` class uses a `Car` object to represent the car that was rented and stores time period of the rental
- `Customer` class represents a customer using their name and rental record (a list of `Rental` objects)
- Customers can accumulate ‘frequent renter points’
  - 1 point for each rental, or 2 points if a new model has been rented for at least 3 days

Initial Tasks

- Study the `step1` code
- Sketch a UML class diagram (Exercise 1 task!)
    - Important fields and methods
    - Class relationships
- What are your impressions of the code?
    - Is it well designed?
    - Is it properly object-oriented?
    - Are there obvious flaws?

Class Diagram

UNIVERSITY OF LEEDS



Impressions

Questions

UNIVERSITY OF LEEDS

- If the program works, does any of this really matter?
- Imagine we needed to extend the system so that it can generate and print statements in HTML
    - We could add an `htmlStatement()` method to the `Customer` class
    - What issues would this cause?

Extract Method

UNIVERSITY OF LEEDS

- One of the most common refactorings
- Applied when you have a method that is too long or that needs a comment to understand its purpose
- Moves a coherent chunk of code into its own method, with a good name that describes its intention
- Existing method becomes shorter, need for comments is reduced (IF the new methods have good names), code clarity increases…

Detailed Mechanics

UNIVERSITY OF LEEDS

1. Run tests; make sure they all pass
2. Create new **target method**, named appropriately
3. Copy code from source method to target
4. Determine whether variables referenced in the code need to be local variables of the target, or parameters
5. Run tests; make sure they all pass
6. Replace extracted code in the source method with a call to the target method (**delegation**)
7. Run tests; make sure they all pass

Next Steps

UNIVERSITY OF LEEDS

- Study the `step2` code; compare the `Customer` class with the version from `step1`
- Is computation of rental charge in the right place?

Move Method

UNIVERSITY OF LEEDS

- Used when classes have too much behaviour, or when classes collaborate too much (high coupling)
- Specifically: a method is using, or is used by, more features of another class than the class on which it is defined
- We create a new method with a similar body in the class that uses it most, then turn the old method into a **delegation**, or remove it altogether

Detailed Mechanics

1. Run tests; make sure they all pass
2. Check sub- and superclasses for other declarations of the source method, which might prevent the move
3. Check for any features in source class that can also be moved with the source method
4. Declare the method in target class and copy code from source method to target, adjusting to fit
5. Compile target class
6. Turn source method into a delegating method
7. Run tests; make sure they all pass
8. If removing source method, replace calls to it with calls to the new method in target class, and run tests again

Inline Temp

UNIVERSITY OF LEEDS

- Variable `thisAmount` is set to the result of calling `getCharge` and is not changed afterward
- … so we can replace occurrences of `thisAmount` with a direct call to `getCharge`, eliminating the temporary variable
    - This is known as **inlining**

Detailed Mechanics

1. Run tests; make sure they all pass
2. Declare temp variable as `final` and compile
   (to check that it is assigned only once)
3. Find all references to temp and replace them with the right-hand side of the assignment
4. Run tests; make sure they all pass
5. Remove declaration and assignment of temp
6. Run tests; make sure they all pass

Questions

UNIVERSITY OF LEEDS

- What are the advantages of this small change?
- What are the disadvantages?

Inlining

UNIVERSITY OF LEEDS

Advantages:

- Fewer local variables in the code
- Easier to track what's going on in long methods

Disadvantages:

- Inefficiency: repeating a calculation, rather than doing it once and storing the result

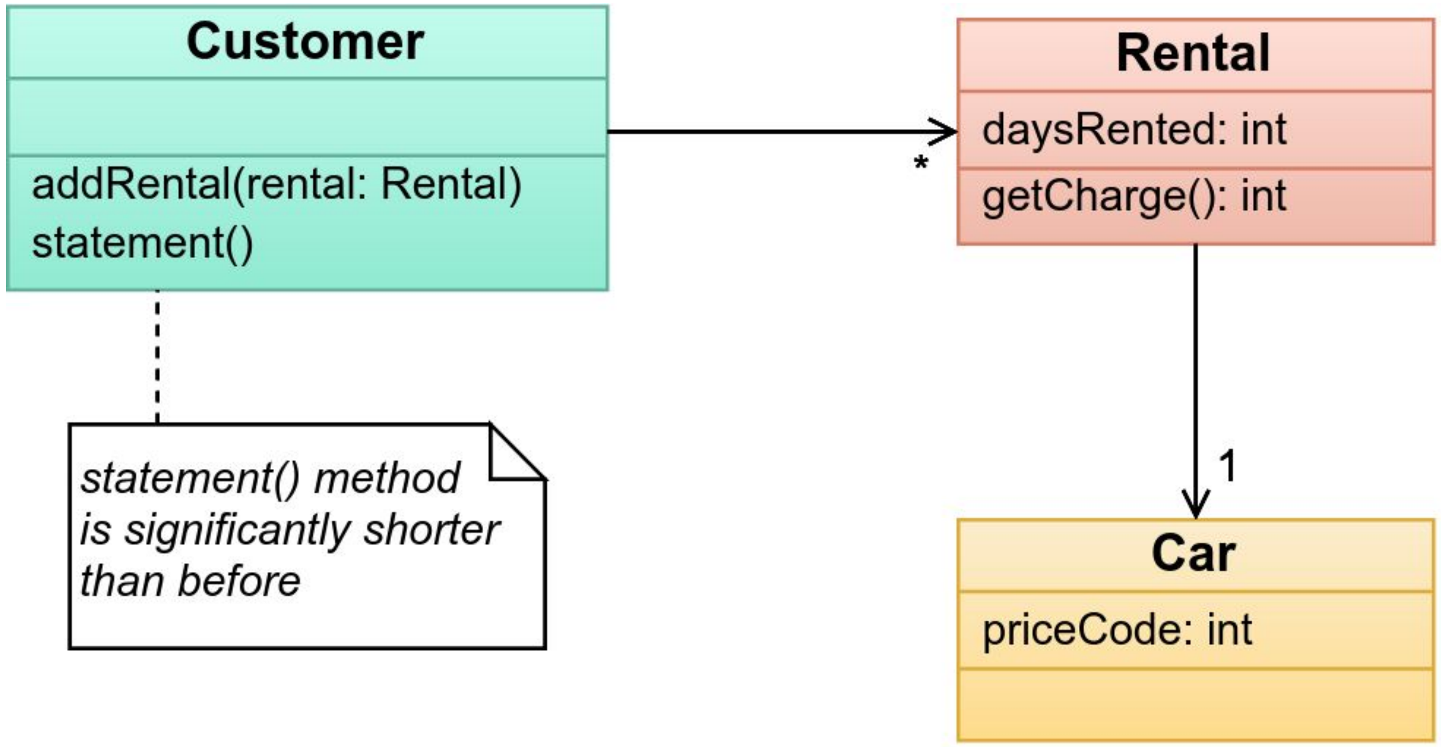
Performance vs Clarity

UNIVERSITY OF LEEDS

- Intuition is a bad guide when it comes to optimization!
- Profile code with tools to see where it is actually slow, and focus optimization efforts on those parts
- Even if refactoring makes code temporarily less efficient, improved clarity can simplify performance tuning
- To sum up: **focus on trying to write simple and clear code**, rather than always trying to ‘be efficient’

Where Are We Now?

UNIVERSITY OF LEEDS


Customer
addRental(rental: Rental)
statement()
*
Rental
daysRented: int
getCharge(): int
1
Car
priceCode: int
statement() method is significantly shorter than before

Where Next?

UNIVERSITY OF LEEDS

- Study the `step3` code
- Can we continue to simplify `statement()` by further use of the refactorings we've seen already?
- If we apply these, how easy will it be to add a new method to generate HTML statements?

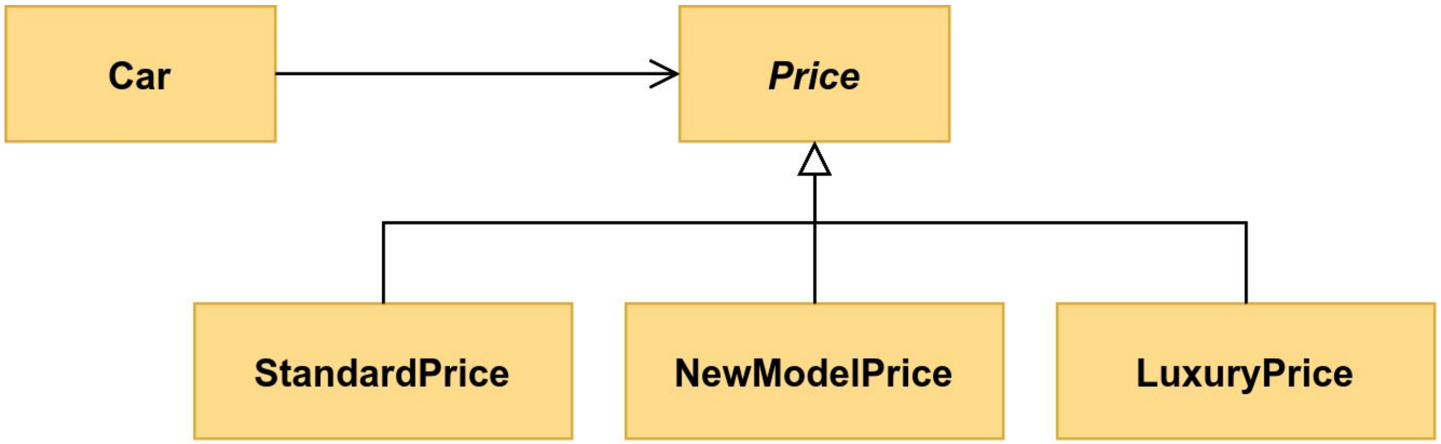
Ideas

- Extract calculation of frequent renter points into its own method, simplifying `statement()` further
- Move calculation of frequent renter points to `Rental`
- Use **Replace Temp with Query** to eliminate the variables that hold total amount charged and total number of frequent renter points

Better Object Orientation

UNIVERSITY OF LEEDS

- Switching on car type to compute amount charged or number of frequent renter points is inflexible
- We can make the system much easier to extend by adding some extra classes and exploiting polymorphism



Summary

UNIVERSITY OF LEEDS

We have

- Examined a small application consisting of three classes and identified the weaknesses in its design
- Applied three refactorings to improve the design in a careful and controlled way
- Discussed the nature of the improvements and considered further design improvement steps

Follow-Up / Further Reading

UNIVERSITY OF LEEDS

- **IMPORTANT**: do Exercise 3 to continue the refactoring!
- See Refactoring Guru's Refactoring Techniques page for further information on 66 different refactorings, including those discussed here