# Code Smells 2
## Refactoring with Fowler

FIT2099: SEMESTER 1 2018

## Recap: code smells and refactoring

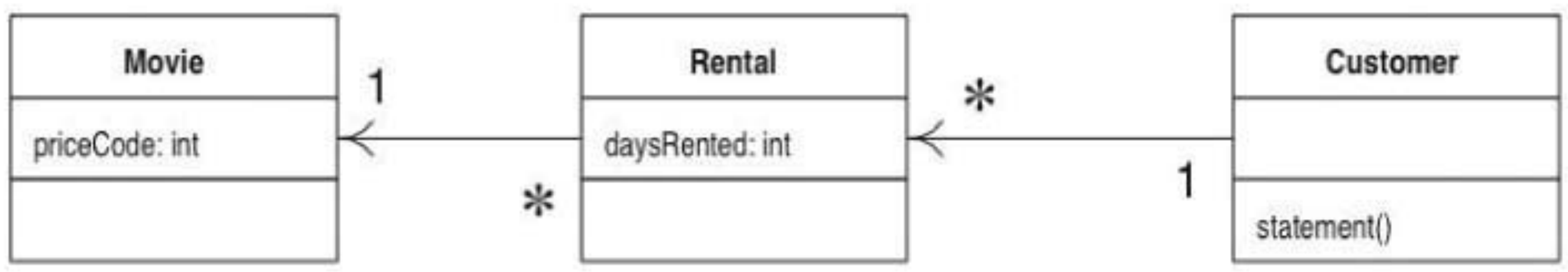
- Code smells: small things in code that indicate design problem
- Refactoring: changing code without changing functionality to improve design

## Fowler's refactoring example

- Fowler came up with an excellent example of refactoring
- Shows off his method

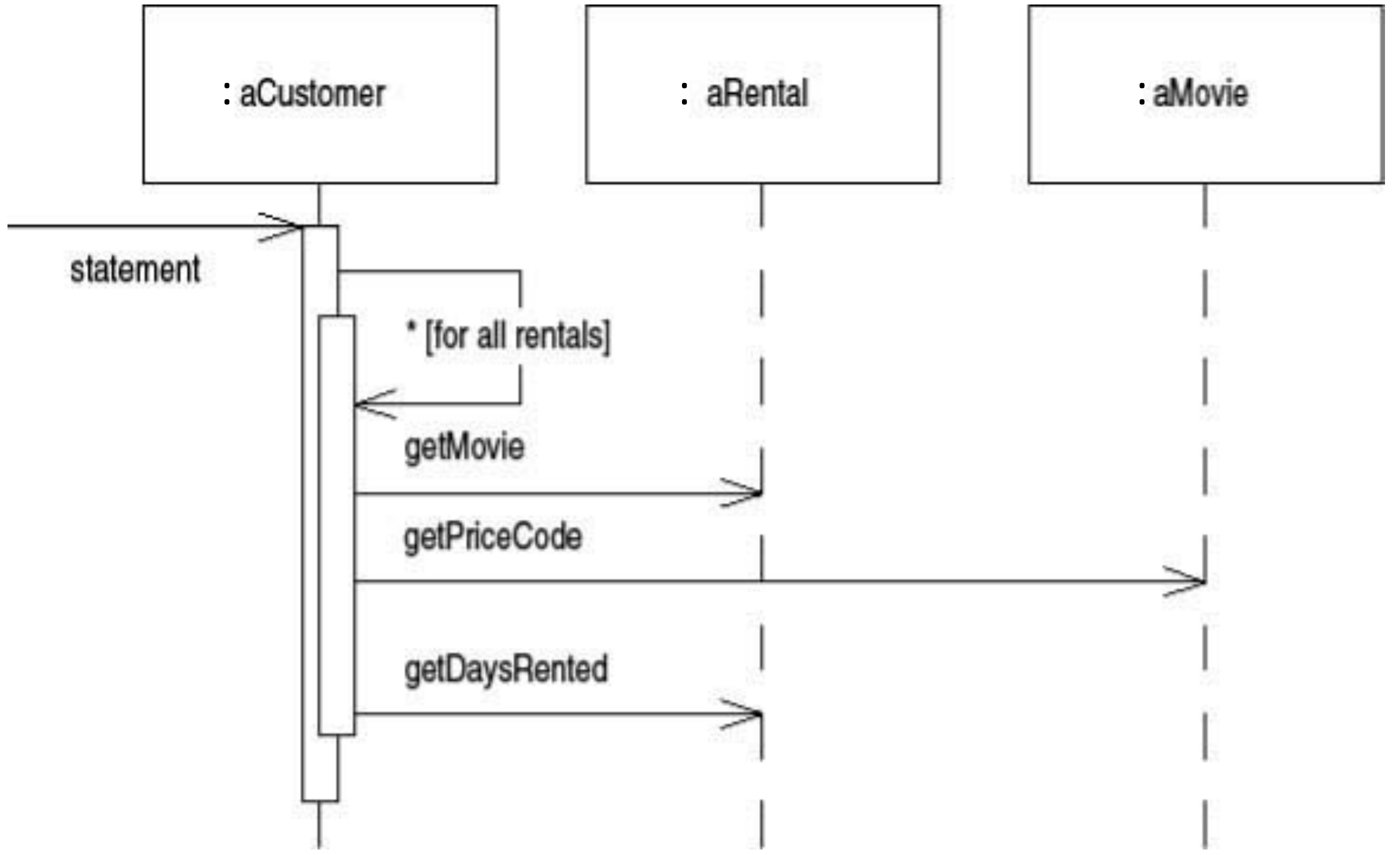
# The video store

## The starting point



- Rental and Movie are plain data classes with trivial setters and getters.
- Customer has trivial setters and getters and a statement() method that produces a customer statement:

# Statement sequence diagram



```java
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator<Rental> rentals = _rentals.iterator();
    String result = "Rental Record for " + getName() + "\n";
    while (rentals.hasNext()) {
        double thisAmount = 0;
        Rental each = rentals.next();

        //determine amounts for each line
        switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2)
                thisAmount += (each.getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            thisAmount += each.getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.getDaysRented() > 3)
                thisAmount += (each.getDaysRented() - 3) * 1.5;
            break;
        }
    // add frequent renter points
    frequentRenterPoints ++;
    // add bonus for a two day new release rental
    if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
          each.getDaysRented() > 1) frequentRenterPoints ++;

    //show figures for this rental
    result += "\t" + each.getMovie().getTitle()+ "\t" +
          String.valueOf(thisAmount) + "\n";
    totalAmount += thisAmount;

}
//add footer lines
result +=  "Amount owed is " + String.valueOf(totalAmount) + "\n";
result += "You earned " + String.valueOf(frequentRenterPoints) +
       " frequent renter points";
return result;
```

## Where to start?

- Statement()

## Long method

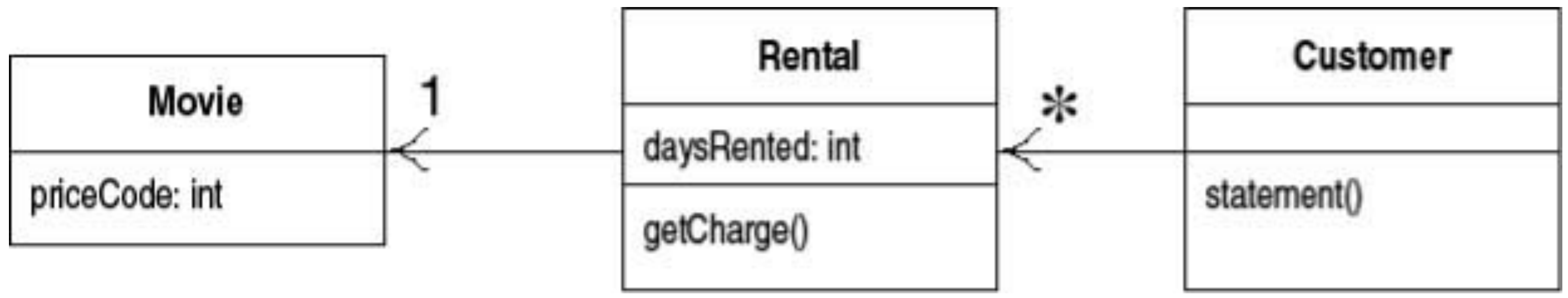
- Fix long methods by extracting parts…

```java
private double amountFor(Rental each) {
    double thisAmount=0;
    switch (each.getMovie().getPriceCode()) {
    case Movie.REGULAR:
        thisAmount = 2;
        if (each.getDaysRented() > 2)
            thisAmount += (each.getDaysRented() - 2) * 1.5;
        break;
    case Movie.NEW_RELEASE:
        thisAmount = each.getDaysRented() * 3;
        break;
    case Movie.CHILDRENS:
        thisAmount = 1.5;
        if (each.getDaysRented() > 3)
            thisAmount += (each.getDaysRented() - 3) * 1.5;
        break;
    }
    return thisAmount;
}
```

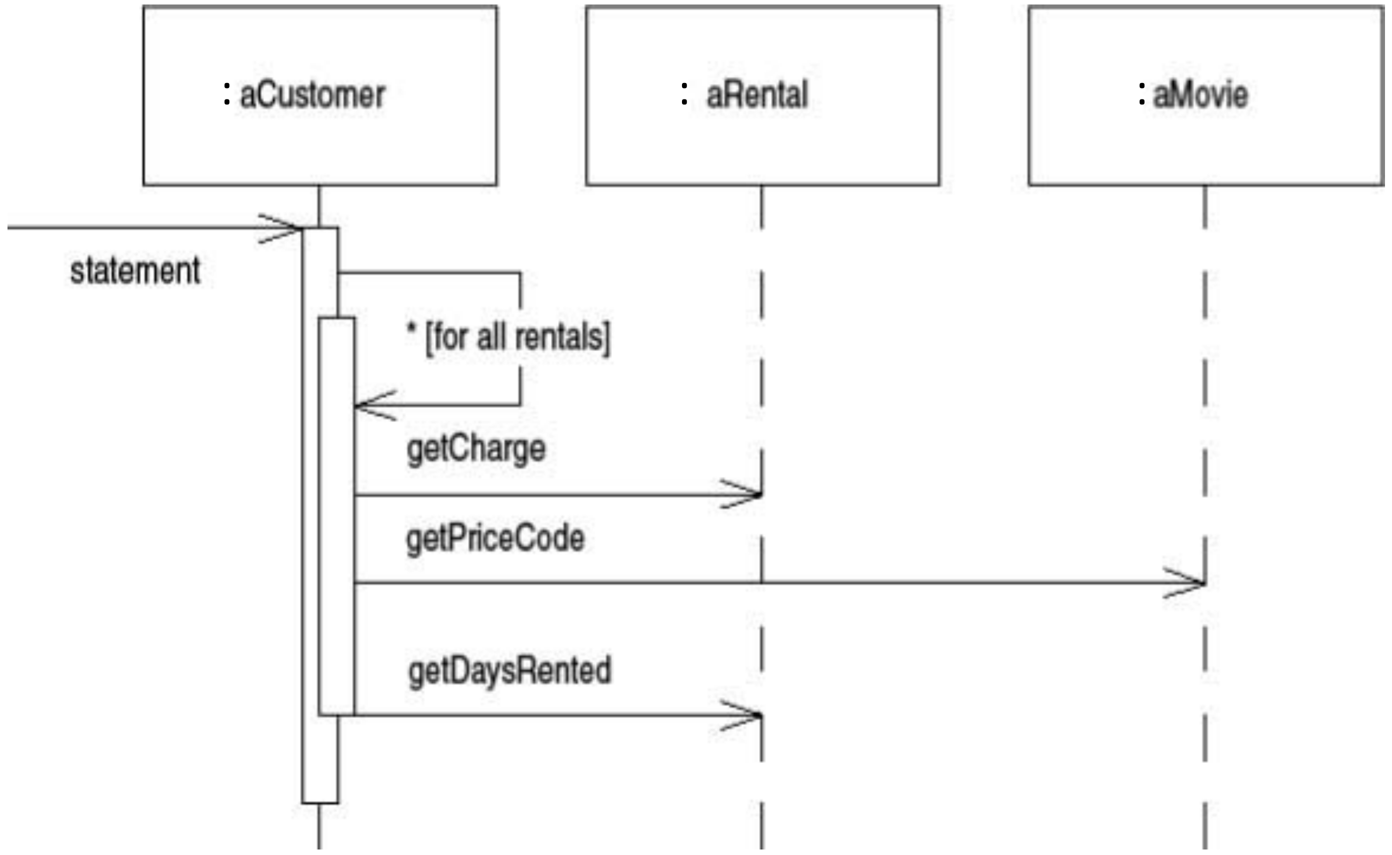
## Move method

- Fowler has a detailed procedure for moving methods across
- We're going to skip over some of the details…

# Result: getCharge in Rental



## GetCharge in Rental



## What about calculating frequent renter points?

- Two step fix!
- Extract Method
- Move Method (to rental)

## Temporary variables

Fowler doesn't like them!

- Only useful within their own routine
- Encourage long, complex routines
- Easy to lose track of

Replace them with queries

- accessible to any method in the class
- encourage a cleaner design without long, complex methods

We have two for computing totals

Fowler suggests factoring them into queries as follows:

```
private double getTotalAmount() {
        double total = 0;
        for (Rental a: _rentals) {
            total += a.getCharge();
        }
        return total;
    }

    private int getTotalFrequentRenterPoints() {
        int total = 0;
        for (Rental a: _rentals) {
            total += a.getFrequentRenterPoints();
        }
        return total;
    }
```

## Replace temp with queries

- Code is now:
  - Longer
  - Slower
- But…
  - Those query are much more easily reusable
  - If we needed to optimize, we could
  - 99% of the time code runs faster than you can imagine
  - Optimizing for human readers rather than absolute speed is almost always what we want to do

# The other hat: adding features.

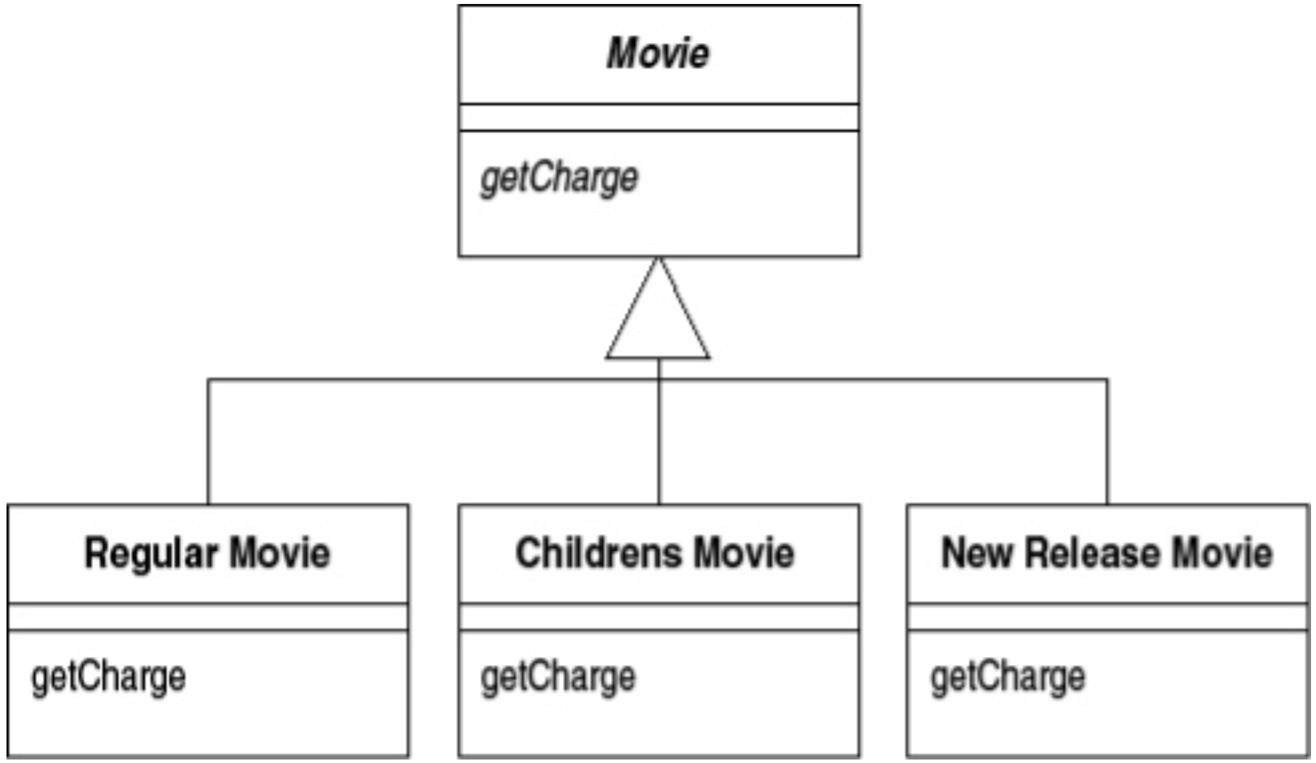
```
public String htmlStatement() {
        String result = "<H1>Rentals for <EM>" + getName() + "</EM></H1><P>\n";
            for (Rental each: _rentals) {
            result += each.getMovie().getTitle()+ ": " +
                String.valueOf(each.getCharge()) + "<BR>\n";
        }
            //add footer lines
            result +=  "<P>You owe <EM>" + String.valueOf(getTotalAmount()) +
"</EM><P>\n";
            result += "On this rental you earned <EM>" +
                String.valueOf(getTotalFrequentRenterPoints()) +
        "</EM> frequent renter points<P>";
            return result;
      }
```

## Simple vs complex refactorings.

- These refactorings were pretty simple
- Not so simple that we could just get a machine to do them…
- But I didn't have to think about them much

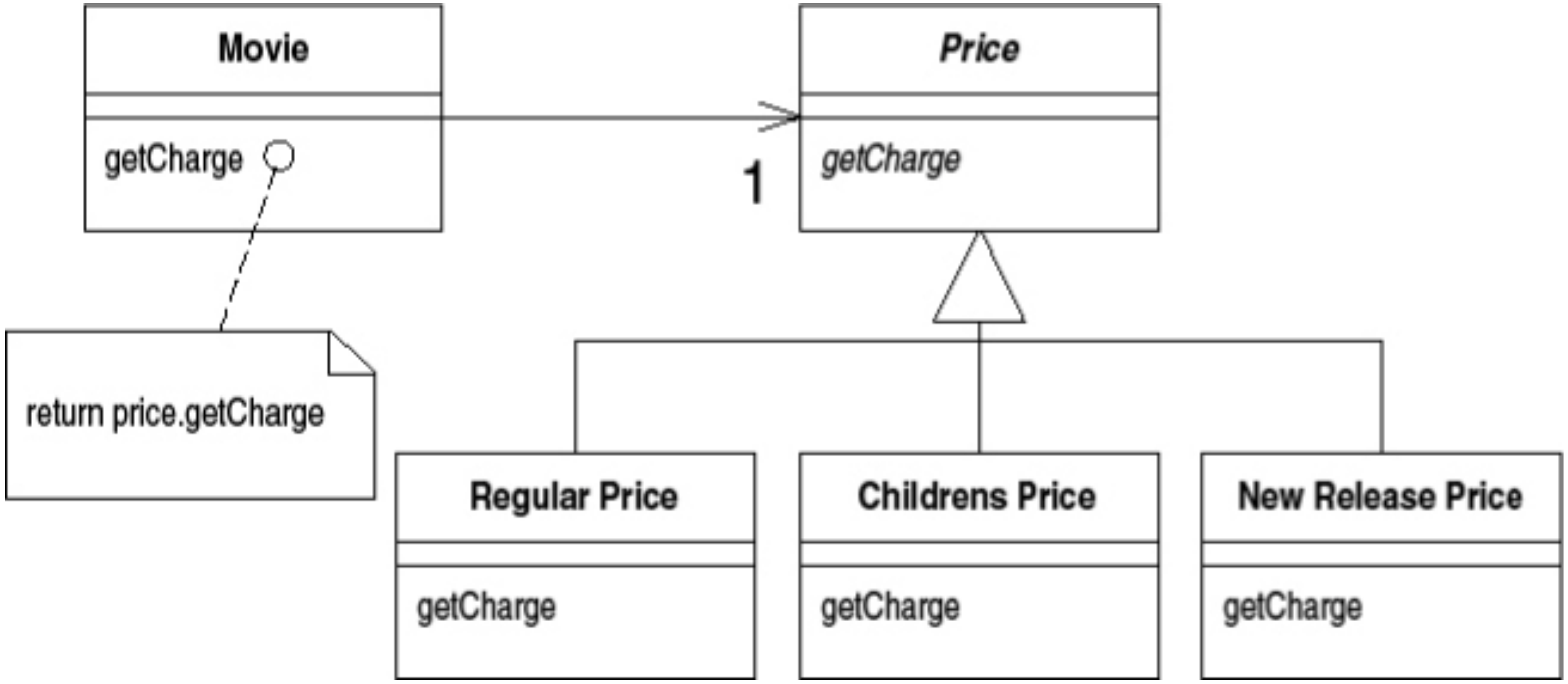
```java
public double getCharge() {
        double thisAmount=0;
        switch (getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount = 2;
            if (getDaysRented() > 2)
                thisAmount += (getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            thisAmount = getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            thisAmount = 1.5;
            if (getDaysRented() > 3)
                thisAmount += (getDaysRented() - 3) * 1.5;
            break;
        }
        return thisAmount;
    }
```

## Fowler Solution #1



210
STEAL
LEASE
RENT
4

# Fowler's solution #2



## More to this example

- Read Chapter 1 of Fowler (link on Moodle)!
- The later refactorings can’t just be applied blindly

## Summary

- Often, implementation reveals your design is imperfect
- Even if it starts off good, modifications and extensions may introduce“technical debt”
- Refactoring – modifying design without changing functionality
- Fowler presents  some standard techniques for refactoring
- **Read Fowler**