Refactoring:
Improving the Design of Existing Code

Or: you can't get there from here…

Yogi Kulkarni
Chris Stevenson
Mujiruddin Shaikh

www.thoughtworks.com

ThoughtWorks®
The art of heavy lifting.℠

What we will cover

- **An example of refactoring**
  - Blow by blow example of reworking an application
  - Motivations
- **Background of refactoring**
  - Where it came from
  - Tools
  - Why and When

ThoughtWorks®
The art of heavy lifting.℠

What is Refactoring

A series of *small* steps, each of which changes the program's internal structure without changing its external behavior

- **Verify no change in external behavior by**
  - Testing
  - Formal code analysis by tool
  - Being very, very careful

Why Refactor

- **To make room for new functionality**
  - Getting ready for structural change
- **To make the program easier to change**
  - Remove duplication
  - Put behaviour in the right place
- **To make the software easier to understand**
  - Express intent
  - Understand unfamiliar code
- **To "Fix broken windows"**
  - The Pragmatic Programmers

A stitch in time…

Three Golden Rules

- » Once and only once
- » Express intent
- » Tell, don't ask

Video Rental Example

- » Sample Output

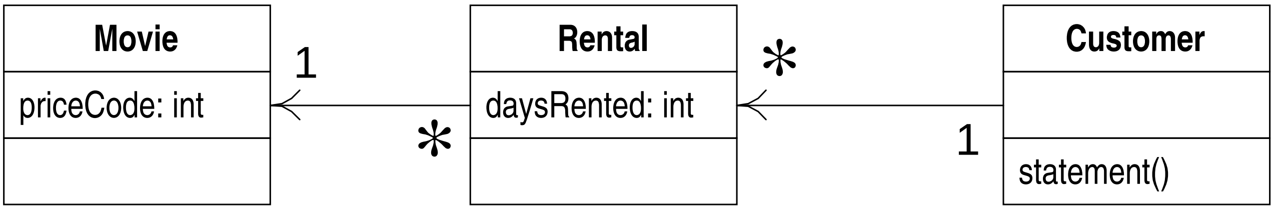
```
Rental Record for Dinsdale Pirhana
   Monty Python and the Holy Grail  3.5
   Ran                              2
   Star Trek 27                     6
   Star Wars 3.2                    3
   Wallace and Gromit               6
Amount owed is                      20.5
You earned 6 frequent renter points
```

Eclipse Lab
Running Unit Tests

Requirements Changes

- **Produce an html version of the statement**
- **The movie classifications will soon change**
  - together with the rules for charging and for frequent renter points

Initial Class diagram



Movie

priceCode: int

1

*

Rental

daysRented: int

*

1

Customer

statement()

Class Movie

```
public class Movie {
    public static final int  CHILDRENS = 2;
    public static final int  REGULAR = 0;
    public static final int  NEW_RELEASE = 1;

    private String title;
    private int priceCode;

    public Movie(String title, int priceCode) {
        this.title = title;
        this.priceCode = priceCode;
    }

    public int getPriceCode() {
        return priceCode;
    }

    public void setPriceCode(int arg) {
    priceCode = arg;
    }

    public String getTitle () {
        return title;
    };
}
```

Class Rental

```
class Rental {
    private Movie movie;
    private int daysRented;

    public Rental(Movie movie, int daysRented) {
                  this.movie = movie;
                  this.daysRented = daysRented;
    }
    public int getDaysRented() {
                  return daysRented;
    }
    public Movie getMovie() {
                  return movie;
    }
}
```

Class Customer (partial)

```
public class Customer {
    private String name;
    private ArrayList rentalList = new ArrayList();

    public Customer(String name) {
        this.name = name;
    }

    public void addRental(Rental arg) {
        rentalList.add(arg);
    }

    public String getName() {
        return name;
    }

       public String statement() // see next slide
```

Customer.statement() i

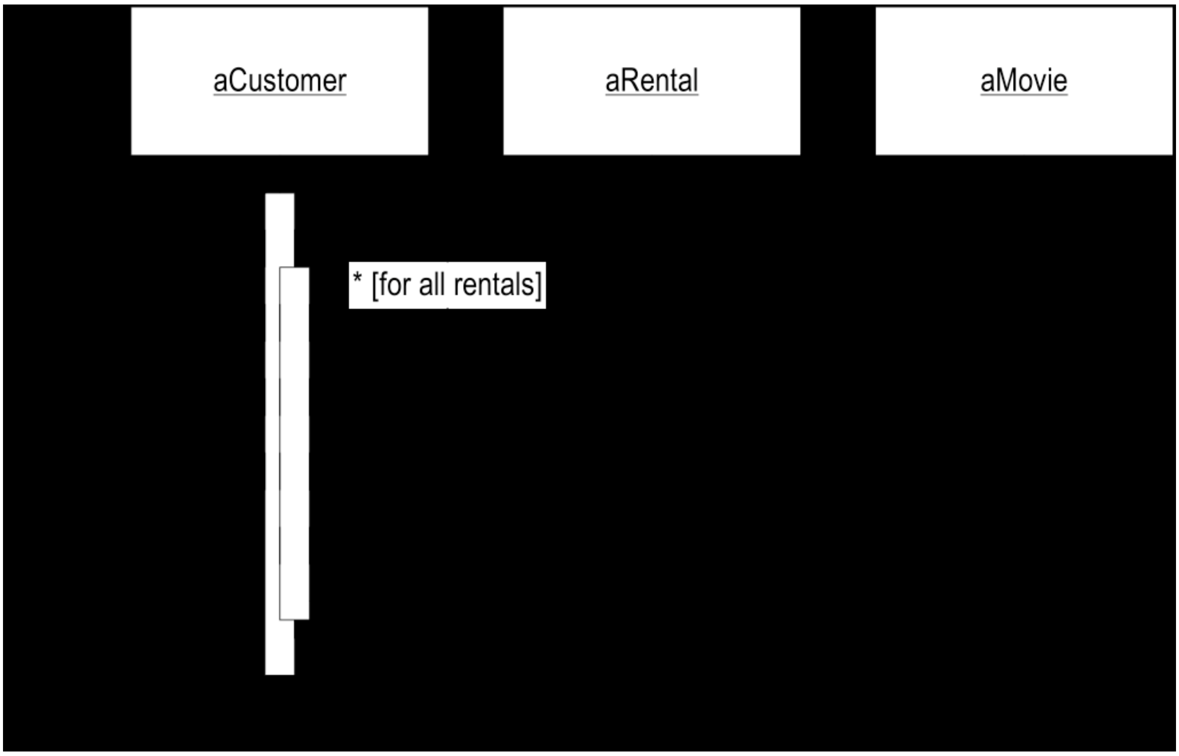
```
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator rentals = rentalList.iterator();
    String result = "Rental Record for " + getName() + "\n";
    while (rentals.hasNext()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.next();

      //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;

      }
```

continues on next slide

Customer.statement() ii

```
      // 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"
             + thisAmount + "\n";
      totalAmount += thisAmount;

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

Interactions for statement



Code smell: Long Method

- Long methods are hard to read and understand
- Often have multiple side effects
- Related smells:
  - **Explaining comment** – move the commented code into a new method and remove the comment

Code smells: recognising refactorings

- » **Code smells are an attempt to describe why code looks 'wrong' or awkward or is resistant to change**
- » **Code smells suggest particular refactorings**
- » **eg. statement() in the example**
  - – smell: **long method**
  - – refactoring: **extract method**

A pattern language for Refactoring

- » **A language to communicate between developers**
- » **Allows us to have a discussion at a more abstract level**
- » **Communication between developers is more efficient**

And you'll need tests

- Use a simple test framework to write and organize tests
  - http://www.junit.org
  - http://xprogramming.com/software
- Small fast tests for code you're working on
- Complete tests for build
  - Run full test suite as part of build process
  - http://martinfowler.com/articles/continuousIntegration.html
- Build tests as you go for legacy code

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() {
  printBanner();

  // printDetails
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```

```
void printOwing() {
  printBanner();
  printDetails(getOutstanding());
}

void printDetails(Amount outstanding) {
  System.out.println("name: " + name);
  System.out.println("amount: " + outstanding);
}
```

Candidate Extraction

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

Extracting the Amount Calculation

```
private double amountFor(Rental each) {
  double thisAmount = 0.0;
  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;
}
```

statement() after extraction

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

    thisAmount = amountFor(each);

    // 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" + thisAmount + "\n";
    totalAmount += thisAmount;

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

Change names of variables

```
private double amountFor(Rental rental) {
     double result = 0.0;
     switch (rental.getMovie().getPriceCode()) {
     case Movie.REGULAR:
        result += 2;
        if (rental.getDaysRented() > 2)
              result += (rental.getDaysRented() - 2) * 1.5;
        break;
     case Movie.NEW_RELEASE:
        result += rental.getDaysRented() * 3;
        break;
     case Movie.CHILDRENS:
        result += 1.5;
        if (rental.getDaysRented() > 3)
              result += (rental.getDaysRented() - 3) * 1.5;
        break;
     }
     return result;
  }
```

Is this important?

Is this method in the right place?

Review

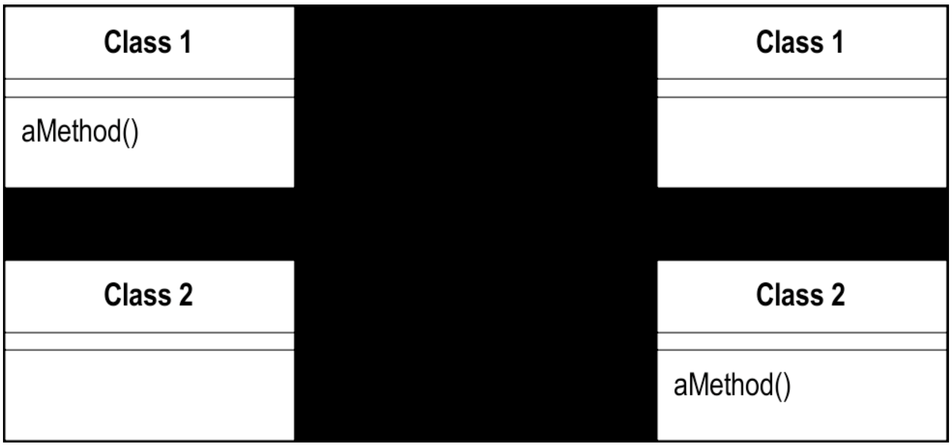
- **Express intent**
  - Extracted pricing functionality
  - Renamed variables

Move Method

Feature Envy

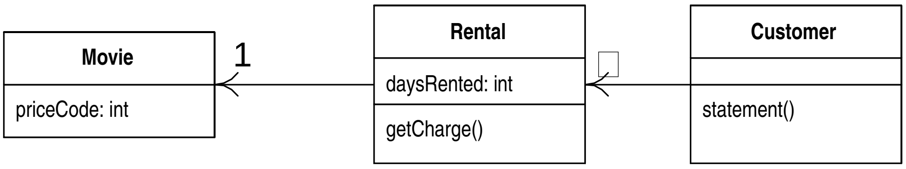
A method uses more features of another class than the class on which it is defined.

Create a new method with a similar body in the relevant class. Replace with simple delegation or remove the original method.



Moving amount() to Rental

```
public class Rental
  public double **getCharge**() {
    double result = 0.0;
    switch (getMovie().getPriceCode()) {
      case Movie.REGULAR:
        result += 2;
        if (getDaysRented() > 2)
          result += (getDaysRented() - 2) * 1.5;
        break;
      case Movie.NEW_RELEASE:
        result += getDaysRented() * 3;
        break;
      case Movie.CHILDRENS:
        result += 1.5;
        if (getDaysRented() > 3)
          result += (getDaysRented() - 3) * 1.5;
        break;
    }
    return result;
  }
  // etc…
}
```



statement() after move

```
class Customer {
  public String **statement**() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator rentals = rentalList.iterator();


    String result = "Rental Record for " + getName() + "\n";
    while (rentals.hasNext()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.next();


      thisAmount = each.amountFor();

      // 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“
                + thisAmount + "\n";
      totalAmount += thisAmount;
    }
  // etc…
```

Extract and move frequentRenterPoints

```
public String statement() {
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  Iterator rentals = rentalList.iterator();
  String result = "Rental Record for " + getName() + "\n";


  while (rentals.hasNext()) {
    double thisAmount = 0;
    Rental each = (Rental) rentals.next();


    thisAmount = each.amountFor();
    frequentRenterPoints += each.frequentRenterPoints();

    //show figures for this rental
    result += "\t" + each.getMovie().getTitle() + "\t"
              + thisAmount + "\n";
    totalAmount += thisAmount;
  }
  //add footer lines
  result += "Amount owed is " + totalAmount + "\n";
  result += "You earned " + frequentRenterPoints
            + " frequent renter points";
  return result;
}
```

ThoughtWorks
The art of heavy lifting.

Code Smells

- **What are the code smells in frequentRenterPoints()?**

```
int frequentRenterPoints() {
            int points = 0;
            // add frequent renter points
            points++;
            // add bonus for a two day new release rental
            if ((getMovie().getPriceCode() == Movie.NEW_RELEASE)
                  && getDaysRented() > 1) points++;
            return points;
      }
```

ThoughtWorks
The art of heavy lifting.

Consolidate conditional expression

```
class Rental {
  public int frequentRenterPoints() {
    if (qualifiesForBonusPoints())
       return 2;
    return 1;
  }

  private boolean qualifiesForBonusPoints() {
    return getMovie().getPriceCode() == Movie.NEW_RELEASE
           && getDaysRented() > 1;
  }
  // etc…
```

Review

» **Express intent**

- Moved amountFor and frequentRenterPoints to relevant class
- Added helper methods to describe conditions

Find related reporting behaviour

```
public String statement() {
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  Iterator rentals = rentalList.iterator();
  String result = "Rental Record for " + getName() + "\n";


  while (rentals.hasNext()) {
    double thisAmount = 0;
    Rental each = (Rental) rentals.next();


    thisAmount = each.amountFor();
    frequentRenterPoints += each.frequentRenterPoints();

    //show figures for this rental
    result += "\t" + each.getMovie().getTitle() + "\t"
          + thisAmount + "\n";
    totalAmount += thisAmount;
  }
  //add footer lines
  result += "Amount owed is " + totalAmount + "\n";
  result += "You earned " + frequentRenterPoints
          + " frequent renter points";
  return result;
}
```

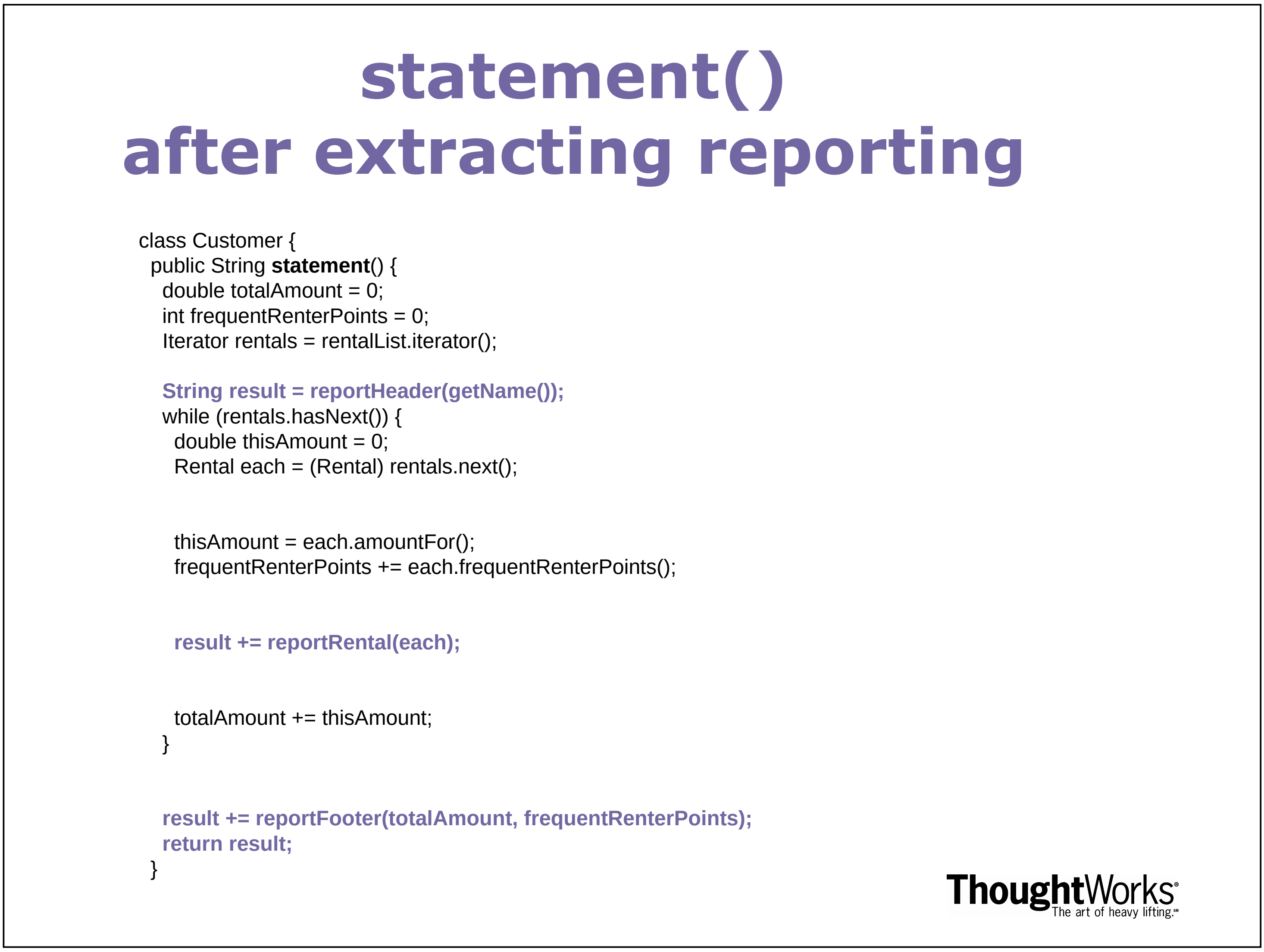
ThoughtWorks
The art of heavy lifting.℠

Extract reporting into methods

```
class Customer {
  // etc…
  private String reportHeader(String customerName) {
    return "Rental Record for " + customerName + "\n";
  }


  private String reportRental(Rental rental) {
    return "\t" + rental.getMovie().getTitle()
         + "\t"+ rental.amountFor() + "\n";
  }


  private String reportFooter(double totalAmount, int renterPoints) {
    return "Amount owed is " + totalAmount + "\n"
         + "You earned " + renterPoints
              + " frequent renter points\n";
 }
}
```

Preserve Whole Object

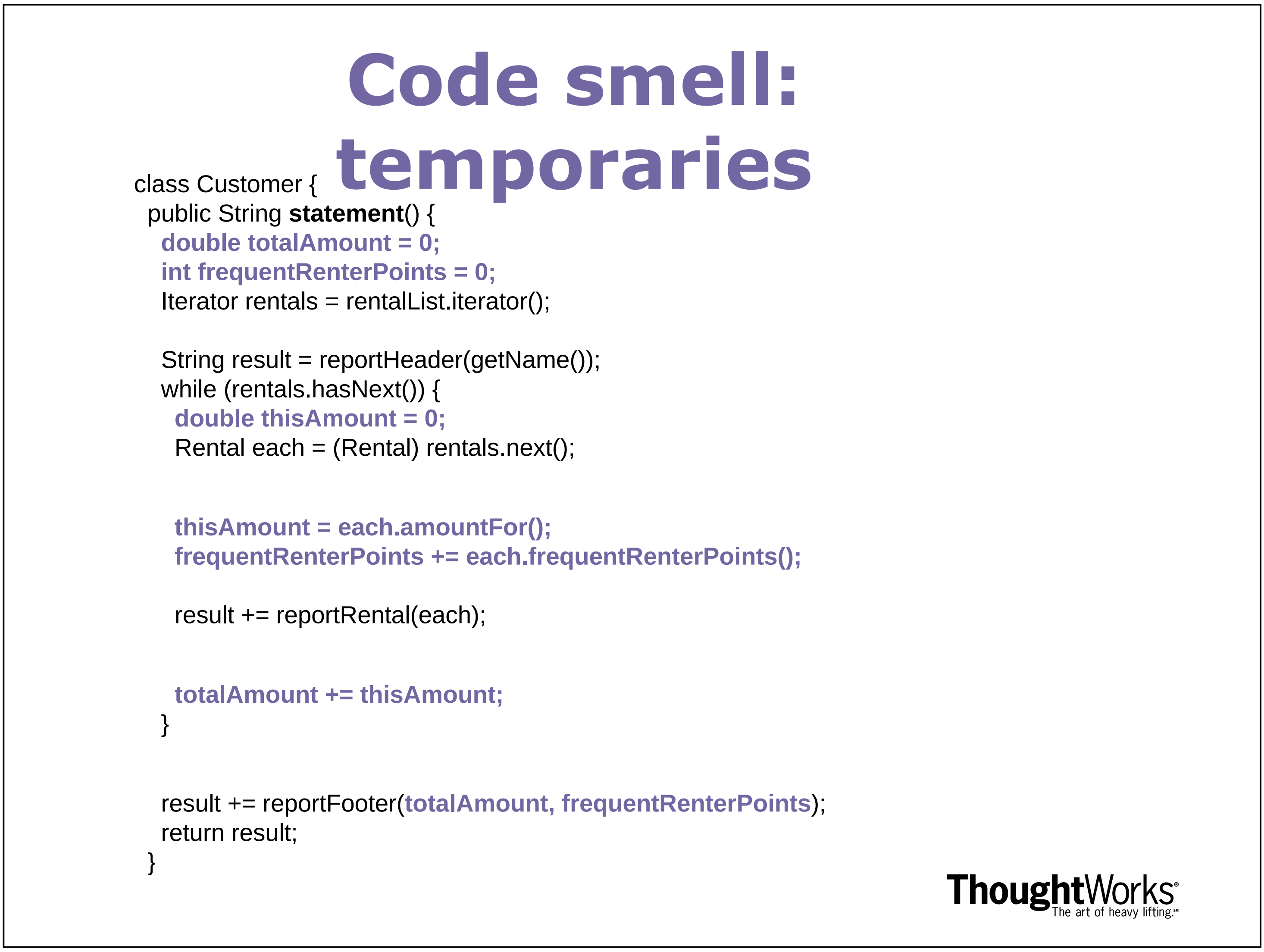
ThoughtWorks
The art of heavy lifting.℠



statement() after extracting reporting

```
class Customer {
  public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator rentals = rentalList.iterator();

    String result = reportHeader(getName());
    while (rentals.hasNext()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.next();


      thisAmount = each.amountFor();
      frequentRenterPoints += each.frequentRenterPoints();


      result += reportRental(each);


      totalAmount += thisAmount;
    }


    result += reportFooter(totalAmount, frequentRenterPoints);
    return result;
  }
```

ThoughtWorks
The art of heavy lifting.



Code smell: temporaries

```
class Customer {
  public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Iterator rentals = rentalList.iterator();

    String result = reportHeader(getName());
    while (rentals.hasNext()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.next();


      thisAmount = each.amountFor();
      frequentRenterPoints += each.frequentRenterPoints();

      result += reportRental(each);


      totalAmount += thisAmount;
    }


    result += reportFooter(totalAmount, frequentRenterPoints);
    return result;
  }
```

ThoughtWorks
The art of heavy lifting.

Split Loop

You have a loop that is doing two things

Duplicate the loop

```
void printValues(Person [] people) {
  double averageAge = 0;
  double totalSalary = 0;
  for (int i = 0; i < people.length; i++) {
    averageAge += people[i].age;
    totalSalary += people[i].salary;
  }
  averageAge = averageAge / people.length;
  System.out.println(averageAge);
  System.out.println(totalSalary);
}
```

```
void printValues(Person [] people) {
  double totalSalary = 0;
  for (int i = 0; i < people.length; i++) {
    totalSalary += people[i].salary;
  }

  double averageAge = 0;
  for (int i = 0; i < people.length; i++) {
    averageAge += people[i].age;
  }
  averageAge = averageAge / people.length;

  System.out.println(averageAge);
  System.out.println(totalSalary);
}
```

ThoughtWorks The art of heavy lifting.℠

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;
}
```

ThoughtWorks The art of heavy lifting.℠

Extract totals into methods

```
private double getTotalAmount() {
  double totalAmount = 0;

  Iterator rentals = rentalList.iterator();
  while (rentals.hasNext()) {
    totalAmount += ((Rental)rentals.next()).amountFor();
  }
  return totalAmount;
}

private int getTotalRenterPoints() {
  int renterPoints = 0;

  Iterator rentals = rentalList.iterator();
  while (rentals.hasNext()) {
    renterPoints += ((Rental)rentals.next()).frequentRenterPoints();
  }
  return renterPoints;
}
```

statement()
after extracting totals

```
class Customer {
  public String statement() {
    String result = reportHeader(getName());


    Iterator rentals = rentalList.iterator();
    while (rentals.hasNext()) {
      result += reportRental((Rental) rentals.next());
    }


    result += reportFooter(getTotalAmount(),
                           getTotalRenterPoints());
    return result;
  }
```

Review

- » **Express intent**
  - – Extracted reporting behaviour
  - – Extracted methods for totals
- » **Once and only once**
  - – Broke this, but sometimes you have to
    - – Language smell?

A word about performance

The best way to optimize performance is to first write a well factored program, then optimize it.

Most of a program's time is taken in a small part of the code

Profile a running program to find these "hotspots"

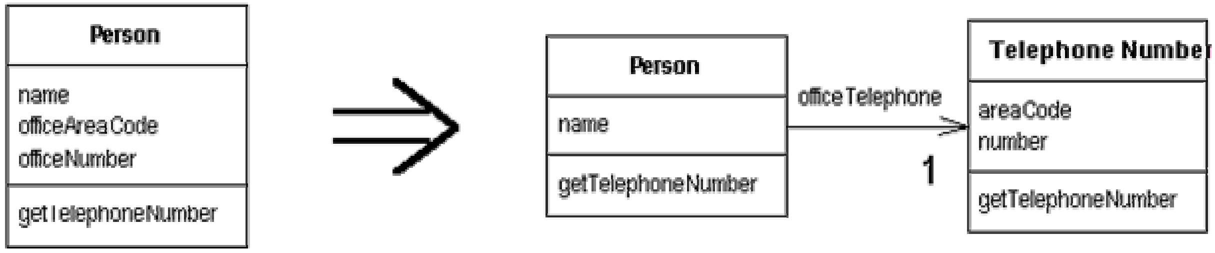
You won't be able to find them by eye

Optimize the hot spots, and measure the improvement

McConnell Steve, *Code Complete: A Practical Handbook of Software Construction*,
Microsoft Press, 1993

Extract Class

You have one class doing work that should be done by two.
Create a new class and move the relevant fields and methods from the old class into the new class.



Extract StatementReporter

```
public class StatementReporter {
  public String reportHeader(String customerName) {
    return "Rental Record for " + customerName + "\n";
  }

  public String reportRental(Rental rental) {
    return "\t" + rental.getMovie().getTitle() + "\t"
           + rental.amountFor() + "\n";
  }

  public String reportFooter(double totalAmount, int renterPoints) {
    return "Amount owed is " + totalAmount + "\n" + "You earned "
           + renterPoints + " frequent renter points\n";
  }
}
```

statement( ) with StatementReporter

```
public String statement() {
  StatementReporter reporter = new StatementReporter();
  String result = reporter.reportHeader(getName());

  Iterator rentals = rentalList.iterator();
  while (rentals.hasNext()) {
    result += reporter.reportRental((Rental) rentals.next());
  }


  result += reporter.reportFooter(getTotalAmount(), getTotalRenterPoints());
  return result;
}
```

Move string accumulation into StatementReporter

```
public class StatementReporter {
    private String contents = "";

    public void reportHeader(String customerName) {
        contents += "Rental Record for " + customerName + "\n";
    }
    public void reportRental(Rental rental) {
        contents += "\t" + rental.getMovie().getTitle()
                 + "\t"+ rental.amountFor() + "\n";
    }
    public void reportFooter(double totalAmount, int renterPoints) {
        contents += "Amount owed is " + totalAmount + "\n"
                 + "You earned " + renterPoints + " frequent renter points\n";
    }
    public String getContents() {
        return contents;
    }
}
```

statement() without strings

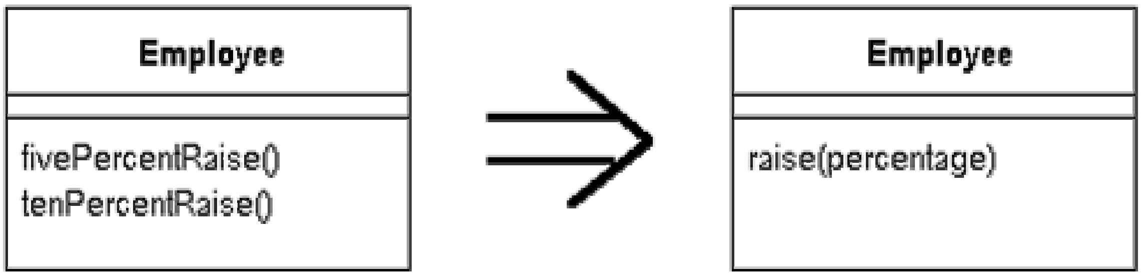
```
public String statement() {
  StatementReporter reporter = new StatementReporter();
  reporter.reportHeader(getName());

  Iterator rentals = rentalList.iterator();
  while (rentals.hasNext()) {
    reporter.reportRental((Rental) rentals.next());
  }

  reporter.reportFooter(getTotalAmount(), getTotalRenterPoints());
  return reporter.getContents();
}
```

Review

- **Express intent**
  - Separate report generation from statement code
- **Tell, don't ask**
  - Tell StatementReporter about rental details

Parameterise Method

Several methods do similar things but with different values contained in the method body.
Create one method that uses a parameter for the different values.



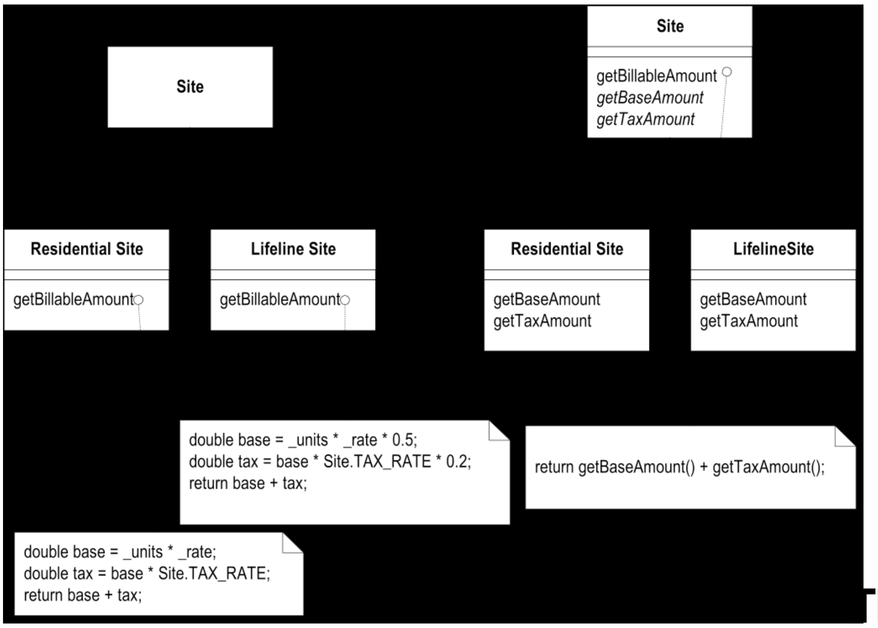
statement()
and htmlStatement()

```
public class Customer {
  public String statement() {
      return statement(new TextStatementReporter());
   }

   public String htmlStatement() {
      return statement(new HtmlStatementReporter());
   }


   public String statement(StatementReporter reporter) {
      reporter.reportHeader(getName());

      Iterator rentals = rentalList.iterator();
      while (rentals.hasNext()) {
         reporter.reportRental((Rental) rentals.next());
      }

      reporter.reportFooter(getTotalAmount(), getTotalRenterPoints());
      return reporter.getContents();
   }
```

Form Template Method

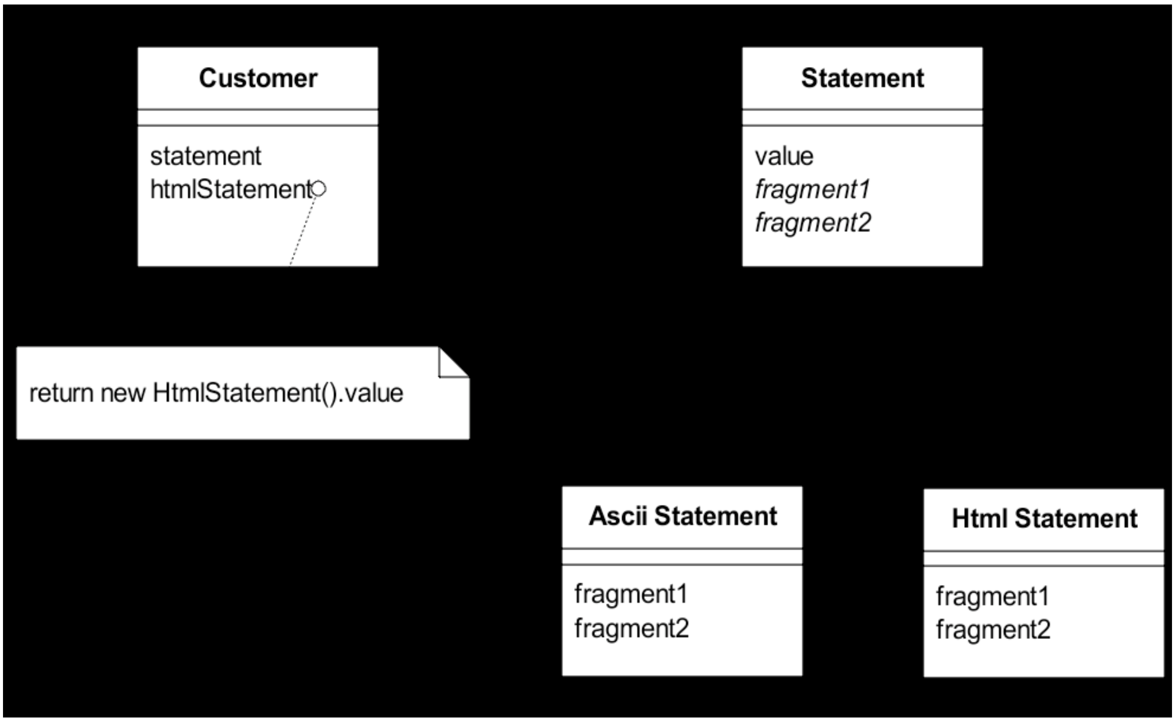
You have two methods in subclasses that carry out similar steps in the same order, yet the steps are different

Break each step into similar helper methods, so that the original methods become the same. Then pull up the original method into a super class.



ThoughtWorks
The art of heavy lifting.

Using a Template Method



ThoughtWorks
The art of heavy lifting.

Abstract StatementReporter

Would be better as a Collecting Object?

```
public abstract class StatementReporter {
   protected String contents = "";

   public String getContents() {
      return contents;
   }

   public abstract void reportHeader(String customerName);

   public abstract void reportRental(Rental rental);

   public abstract void reportFooter(double totalAmount, int renterPoints);
}
```

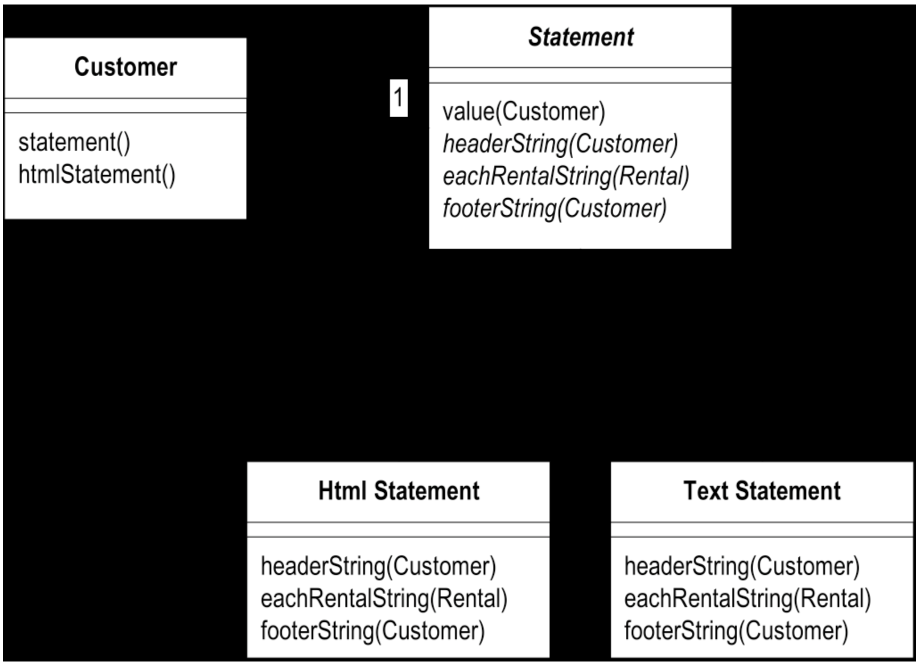
TextStatementReporter

```
public class TextStatementReporter extends StatementReporter {
   public void reportHeader(String customerName) {
      contents += "Rental Record for " + customerName + "\n";
   }
   public void reportRental(Rental rental) {
      contents += "\t" + rental.getMovie().getTitle()
            + "\t"+ rental.amountFor() + "\n";
   }
   public void reportFooter(double totalAmount, int renterPoints) {
      contents += "Amount owed is " + totalAmount + "\n"
            + "You earned " + renterPoints + " frequent renter points\n";
   }
}
```

HtmlStatementReporter

```
public class HtmlStatementReporter extends StatementReporter {
    public void reportHeader(String customerName) {
        contents += "<H1>Rentals for <EM>" + customerName + "</EM></H1><P>\n";
    }
    public void reportRental(Rental rental) {
        contents += rental.getMovie().getTitle() + ": "
                + rental.amountFor() + "<BR>\n";
    }
    public void reportFooter(double totalAmount, int renterPoints) {
        contents += "<P>You owe <EM>" + totalAmount + "</EM><P>\n"
                + "On this rental you earned <EM>"
                    + renterPoints + "</EM> frequent renter points<P>";
    }
}
```

Reporter Classes



Customer

statement()
htmlStatement()

1

Statement

value(Customer)
headerString(Customer)
eachRentalString(Rental)
footerString(Customer)

Html Statement

headerString(Customer)
eachRentalString(Rental)
footerString(Customer)

Text Statement

headerString(Customer)
eachRentalString(Rental)
footerString(Customer)

Review

- **Express intent**
  - Different classes for different formats
- **Tell, don't ask**
  - Pass formatter into statement method
- **Once and only once**
  - Template for common reporting behaviour

Changes so far

- **Functional**
  - Added html reporting
- **Structural**
  - Clarified report writing process
  - Encapsulated report rendering
  - Added extension point for new formats
  - Extracted totals for cost and points
- **Possible further steps**
  - Pass in a StringBuffer, rather than return a String
  - Extract RentalList class

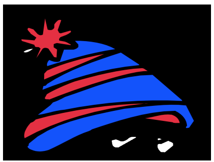
The Two Hats

Adding Functionality

» Add new capabilities to the system

» Adds new tests

» Get the test working



Refactoring

r Does not add any new features

r Does not add tests (but may change some)

r Restructure the code to remove duplication and redundancy

Swap frequently between the hats, but only wear one at a time

Reminder: Rental.amountFor()

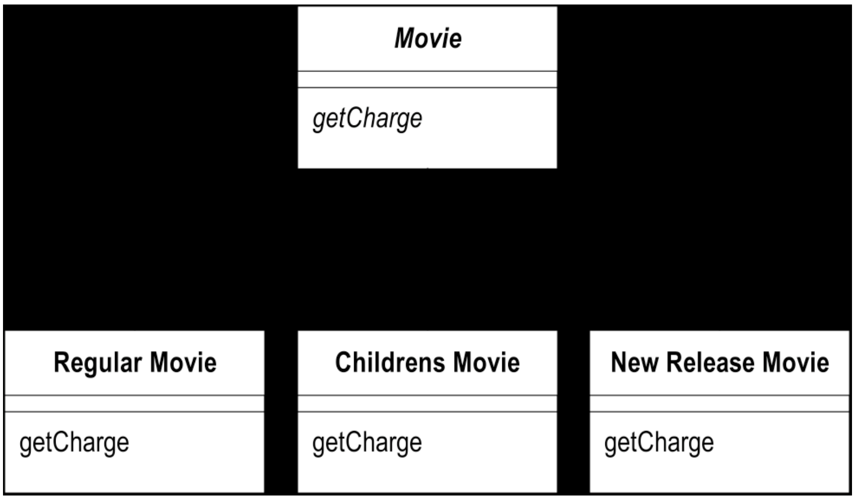
» **Expecting new pricing policy requirements**

```
class Rental            ...
 public double amountFor()  {
        double result = 0;

        switch (getMovie().getPriceCode()) {
        case Movie.REGULAR:
            result += 2;
            if (getDaysRented() > 2)
                    result += (getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            result += getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            result += 1.5;
            if (getDaysRented() > 3)
                    result += (getDaysRented() - 3) * 1.5;
            break;

        }
        return result;
    }
```

Code Smell: Magic Number

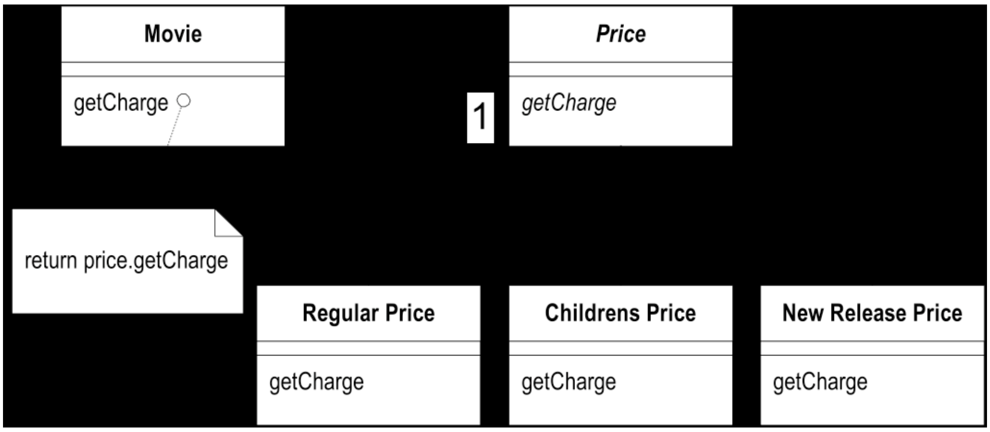
- A Magic Number is a constant that has some special 'magic' meaning
- `switch` and `if` statements
  - public static final int CHILDRENS = 2;
  - if (type == CHILDRENS) {…}
  - behaviour is scattered throughout the code

Consider inheritance



How’s this?

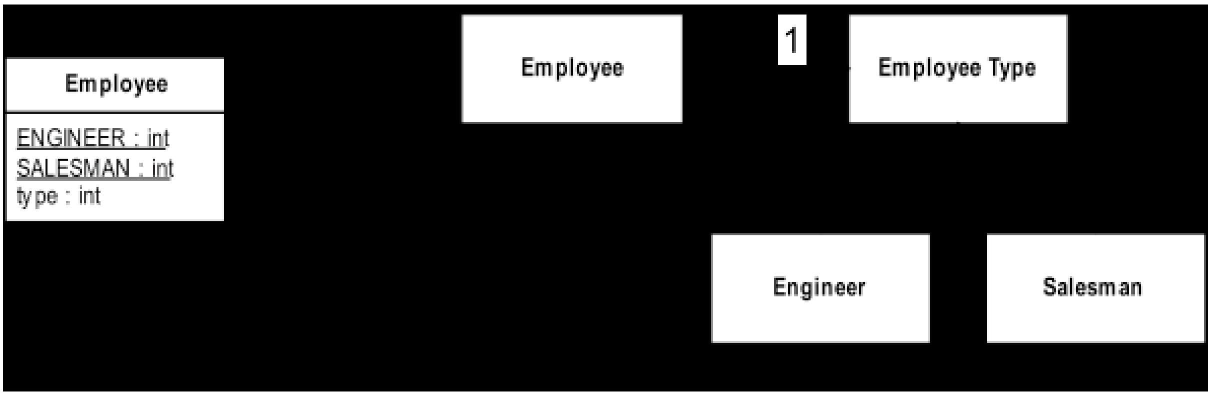
Using the State Pattern



Replace Type Code with State/Strategy

ou have a type code which affects the behavior of a class but you cann use subclassing.

Replace the type code with a state object.



First, move amountFor() to Movie

```
class Rental...
  public double amountFor()  {
    return movie.amountFor(getDaysRented());
  }

class Movie …
  public double amountFor(int daysRented)  {
    double result = 0;

    switch (priceCode) {
    case Movie.REGULAR:
      result += 2;
      if (daysRented > 2)
         result += (daysRented - 2) * 1.5;
      break;
    case Movie.NEW_RELEASE:
      result += daysRented * 3;
      break;
    case Movie.CHILDRENS:
      result += 1.5;
      if (daysRented > 3)
         result += (daysRented - 3) * 1.5;
      break;
    }
    return result;
  }
```

Do the same with frequentRenterPoints()

Push codes into Price object

```
public abstract class Price {
    public final int code;
    protected Price(int code) {
      this.code = code;
    }

    static public class Childrens extends Price {
         public Childrens() { super(Movie.CHILDRENS); }
    }


    static public class NewRelease extends Price {
         public NewRelease() { super(Movie.NEW_RELEASE); }
    }


    static public class Regular extends Price {
         public Regular() { super(Movie.REGULAR); }
  }
}
```

Convert Movie API to Price

```
public class Movie {
  public static final int CHILDRENS = 2;
  public static final int REGULAR = 0;
  public static final int NEW_RELEASE = 1;

  private String title;
  private Price price;

  public Movie(String title, Price price) {
    this.title = title;
    this.price = price;
  }


  public double amountFor(int daysRented)  {
    double result = 0;


    switch (price.code) {
    case Movie.REGULAR:
      result += 2;
      if (daysRented > 2)
        result += (daysRented - 2) * 1.5;
      break;
    // etc. . .
```

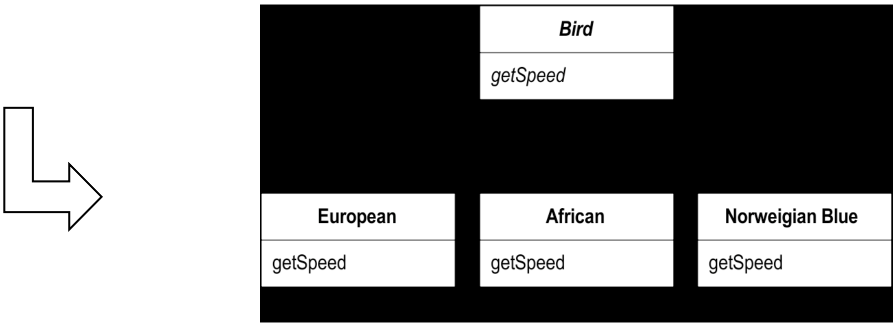
Replace Conditional With Polymorphism

You have a conditional that chooses different behavior depending on a value.

Move each branch of the conditional to an overriding method in a subclass.

Make the original method abstract

```
double getSpeed() throws SparrowException {
  switch (sparrowType) {
    case EUROPEAN:
      return getBaseSpeed();
    case AFRICAN:
      return getBaseSpeed() * getLoadFactor() * numberOfCoconuts;
    case NORWEGIAN_BLUE:
      return isNailed ? 0 : getBaseSpeed(voltage);
  }
  throw new SparrowException(sparrowType);
}
```



Move amountFor() to Price

```
class Movie…
    double amountFor(int daysRented) {
        return price.amountFor(daysRented);
    }

class Price…
    double amountFor(int daysRented) {
        double result = 0;
        switch (code) {
            case Movie.REGULAR:
                result += 2;
                if (daysRented > 2)
                    result += (daysRented - 2) * 1.5;
                break;
            case Movie.NEW_RELEASE:
                result += daysRented * 3;
                break;
            case Movie.CHILDRENS:
                result += 1.5;
                if (daysRented > 3)
                    result += (daysRented - 3) * 1.5;
                break;
        }
        return result;
    }
```

ThoughtWorks
The art of heavy lifting.℠

Override amountFor()

```
public class Childrens extends Price {
  public double amountFor(int daysRented) {
    return 1.5 + chargeAfterMinumumDays(1.5, 3, daysRented);
  }
  public int frequentRenterPoints(int daysRented) {
    return 1;
  }
}

public class NewRelease extends Price {
  public double amountFor(int daysRented) {
    return daysRented * 3;
  }
  public int frequentRenterPoints(int daysRented) {
    return (daysRented > 1) ? 2 : 1;
  }
}

// etc…

class Price {
  abstract double amountFor(int daysRented);
  abstract int frequentRenterPoints(int daysRented);
```

Do each leg, then make parent abstract

ThoughtWorks
The art of heavy lifting.℠

Review

- **Express intent**
  - Extracted Price classes to represent concept
- **Once, and only once**
  - Moved all pricing behaviour into Price objects
- **Tell, don’t ask**
  - Tell price object how many days

Changes so far (ii)

- **Smaller, focussed objects**
  - Movie (title, price)
  - Rental (movie, days rented)
  - Customer (name, list of rentals)
  - Price (charging and points policies)
  - StatementReporters
- **Extensible in one place**
  - By adding new reporting and pricing types
  - Got rid of magic numbers

In this example

- **We saw a poorly factored program improved**
  - Easier to add new services on customer
  - Easier to add new types of movie
- **No debugging during refactoring**
  - Appropriate steps reduce chance of bugs
  - Small steps make bugs easy to find
  - Tests verify behaviour
- **Illustrated several refactorings**
  - Extract method, Move method, Extract class
  - Split loop, Replace temp with query
  - Replace type code with state/strategy
  - Replace conditional with polymorphism
  - Parameterise method, Form template method

Definitions of Refactoring

- **Loose Usage**
  - Reorganize a program (or something)
- **As a noun**
  - a change made to the internal structure of some software to make it easier to understand and cheaper to modify, without changing the observable behavior of that software
- **As a verb**
  - the activity of restructuring software by applying a series of refactorings without changing the observable behavior of that software.

Where Refactoring Came From

- **Ward Cunningham and Kent Beck**
  - Smalltalk style
- **Ralph Johnson at University of Illinois at Urbana-Champaign**
- **Bill Opdyke's Thesis**

  ftp://st.cs.uiuc.edu/pub/papers/refactoring/opdyke-thesis.ps.Z
- **John Brant and Don Roberts: The Smalltalk Refactoring Browser**
  - Available in most Smalltalks

Refactoring Tools

- **Based on provable transformations**
  - use the parse tree of programs
  - can be proven that refactorings do not change semantics
- **Speeds up refactoring**
  - Extract method: select code, type in method name.
  - Big speed and productivity improvement
- **Widely available in modern Java IDEs**
  - Eclipse and IntelliJ IDEA have excellent support
- **Less common in other environments**
  - C#: ReSharper (by IntelliJ) is in early access release and is very good. Refactorings promised in VisualStudio

The Importance of Tests

- **Even with a tool, testing is important**
  - Not all refactorings can be proven
- **Write tests as you write the code**
  - Make the test self-checking
  - Test results are Pass/Fail
    - colloquially known as Green bar/Red bar
    - there is no 'maybe'!
- **Test with every compile**

www.junit.org
www.xprogramming.com/software

When should you refactor? Always!

- **Only refactor on a green bar – ie. when all the tests are passing**
  - Otherwise you don't know if you have broken the system
- **Motivations**
  - To clarify intent
  - To accommodate new functionality
  - To remove duplication
  - To improve testability

Refactoring v Redesign

- **You should not need permission to refactor**
  - it is a key part of modern software development practice
  - you don't need to tell your manager
- **Large scale refactoring/redesign decisions should be owned by the whole team**
  - redesign requires courage
  - discuss with the entire team
  - the team needs to have a shared ownership and a common understanding of what needs to be done

Problems with Refactoring

- **Database Migration**
  - Insulate your objects from the details of persistent database structure
  - Database refactoring is starting to become more common with the use of O/R mapping tools
- **Published Interfaces**
  - Distinguish between stable and unstable versions
  - Keep them decoupled from the internal domain model
- **Generated Code**
  - Tool-generated code is a barrier to refactoring
- **Without working tests**
  - Only with great care

Design Implications

- **Refactoring complements evolutionary design**
  - Consider primarily current needs
  - Refactor as new requirements appear
  - Implies common code ownership
  - Simplicity, simplicity, simplicity
- **Design as a process of *discovery*, not of *invention***
  - Let the code tell you what it needs
  - Avoid speculative design clutter
  - "Heisenberg Principle" says absolute design is impossible

`www.martinfowler.com/articles/designDead.html`

ThoughtWorks
The art of heavy lifting.℠

Team Issues

- **Encourage refactoring culture**
  - Nobody gets things right first time
  - Refactoring is forward progress
- **Provide sound testing base**
  - Tests are essential for refactoring
  - Build system and run tests continually
- **Shared Code Ownership**
  - You should be free to refactor any part of the system

ThoughtWorks
The art of heavy lifting.℠

Final Thoughts

- **Refactoring allows you to improve the design after the code is written**
  - Imagine applications that get better, rather than worse.
- **You don’t have to get the design perfect before you start**
- **Refactor Mercilessly**
  - Don’t put up with inappropriate designs
- **Refactoring is a fundamental part of every developer's toolkit**

Refactor towards simplicity

- **Simple design is hard to achieve the first time around**
- **The 'simplest thing' will change as requirements change and the system evolves**
- ***Occam's Razor:***
  - *Pluralitas non est ponenda sine neccesitate*
  - Do the simplest thing possible
  - XP: You Ain't Gonna Need It (YAGNI)

Refactor towards Patterns

- **Design Patterns and Refactoring work together**
  - The end point of most refactorings is a pattern
  - Patterns are part of a common developer language
- **Design Patterns: Elements of Reusable Object-Oriented Software**
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (aka The Gang of Four)
- **Refactoring to Patterns**
  - by Joshua Kerievsky

Refactor towards a rich Domain Model

- **Code smell: Anaemic Domain Model**
  - See Martin Fowler's article

  `www.martinfowler.com/bliki/AnemicDomainModel.html`
- **Responsibility-Driven Design**
  - *Object Design*, Wirfs-Brock and McKean

  `http://www.wirfs-brock.com/`
- **Domain Driven Design**
  - by Eric Evans

  `http://domaindrivendesign.org/book`

Some references

- ***Refactoring***
  - Martin Fowler, Addison-Wesley
- **www.refactoring.com**
- **www.industriallogic.com/xp/refactoring/**
- **c2.com/cgi/wiki?RefactorMercilessly**

ThoughtWorks®
The art of heavy lifting.℠