JavaOne℠

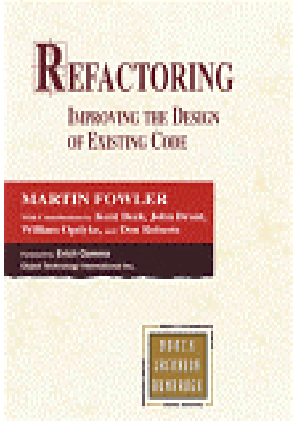
Sun's 2000 Worldwide Java Developer Conference℠

Refactoring: Improving the Design of Existing Code

Martin Fowler
Chief Scientist, ThoughtWorks
fowler@acm.org
www.martinfowler.com

What We Will Cover

- **An example of refactoring**
  - Blow by blow example of changes
  - Steps for illustrated refactorings
- **Background of refactoring**
  - Where it came from
  - Tools
  - Why and When



Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999

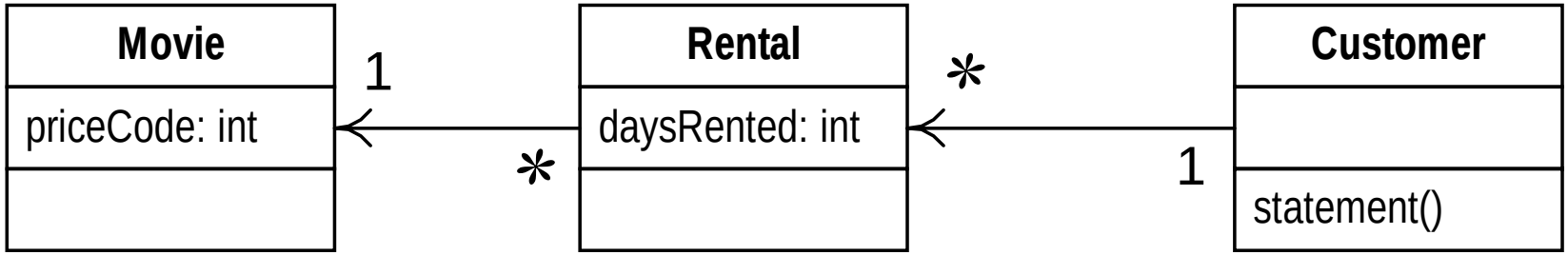
JavaOne

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
- ➨ In practice good tests are essential

Starting Class Diagram



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
```

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) {
    _title = title;
    _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) {
       _movie = movie;
       _daysRented = daysRented;
    }
    public int getDaysRented() {
       return _daysRented;
    }
    public Movie getMovie() {
       return _movie;
    }
}
```

Class Customer (Almost)

```
class Customer {
   private String _name;
   private Vector _rentals = new Vector();

   public Customer (String name)    {
        _name = name;
   };

   public void addRental(Rental arg) {
              _rentals.addElement(arg);
   }
   public String getName ()  {
        return _name;
   };

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

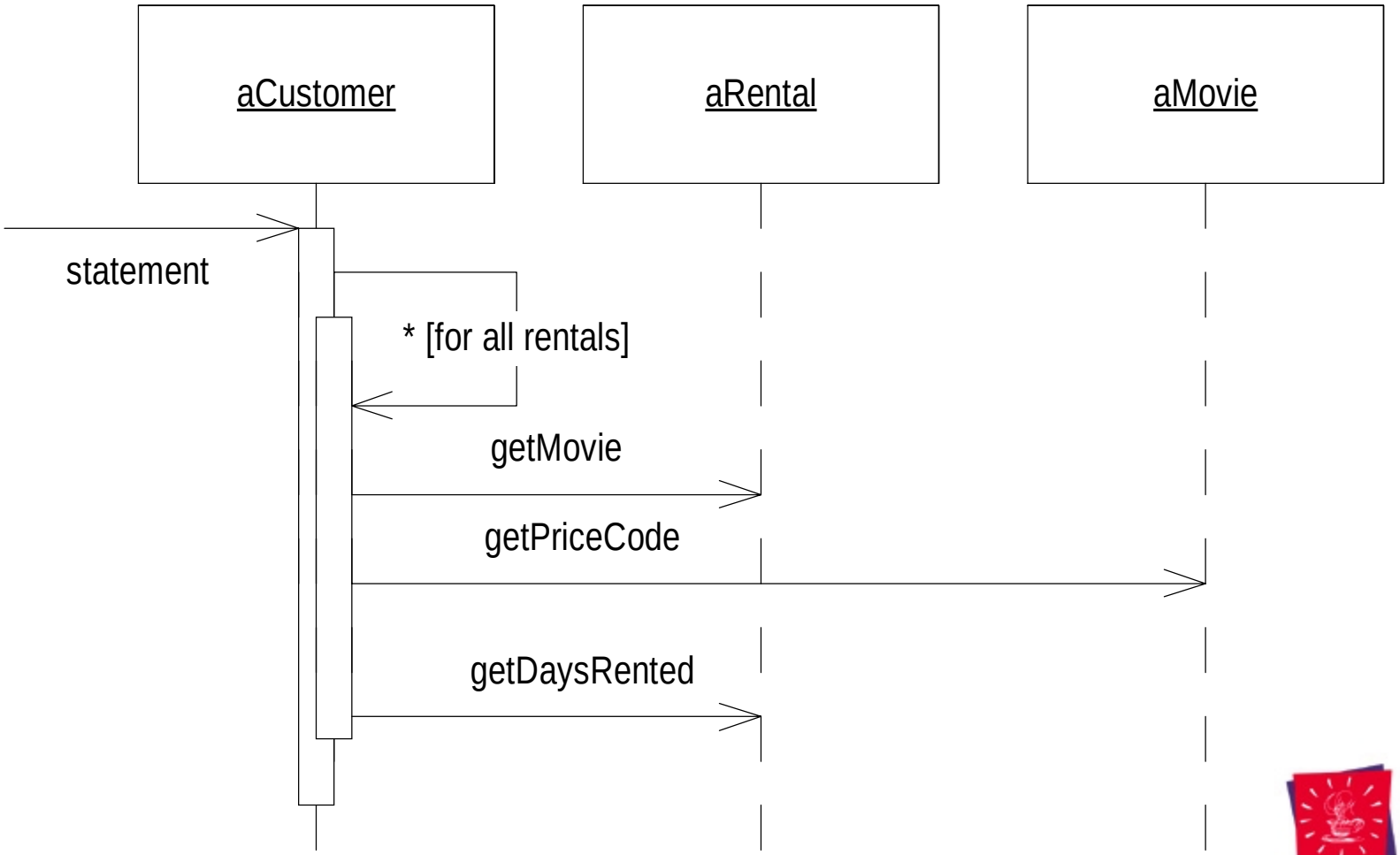
Customer.statement() Part 1

```
public String statement() {
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  Enumeration rentals = _rentals.elements();
  String result = "Rental Record for " + getName() + "\n";
  while (rentals.hasMoreElements()) {
    double thisAmount = 0;
    Rental each = (Rental) rentals.nextElement();

    //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() Part 2

```
    // 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;

}
```

Interactions For Statement



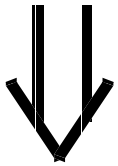
Requirements Changes

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

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();

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



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

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

Candidate Extraction

```
public String statement() {
   double totalAmount = 0;
   int frequentRenterPoints = 0;
   Enumeration rentals = _rentals.elements();
   String result = "Rental Record for " + getName() + "\n";
   while (rentals.hasMoreElements()) {
      double thisAmount = 0;
      Rental each = (Rental) rentals.nextElement();

      //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;

      } //[snip]
```

Steps for *Extract Method*

- Create method named after intention of code
- Copy extracted code
- Look for local variables and parameters
  - Turn into parameter
  - Turn into return value
  - Declare within method
- Compile
- Replace code fragment with call to new method
- Compile and test

Extracting the Amount Calculation

```
private int amountFor(Rental each) {
    int 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;
}
```

Statement() After Extraction

```
public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + getName() + "\n";
      while (rentals.hasMoreElements()) {
          double thisAmount = 0;
          Rental each = (Rental) rentals.nextElement();

          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" +
         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;

  }
}
```

Extracting the Amount Calculation (2)

```
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;
}
```

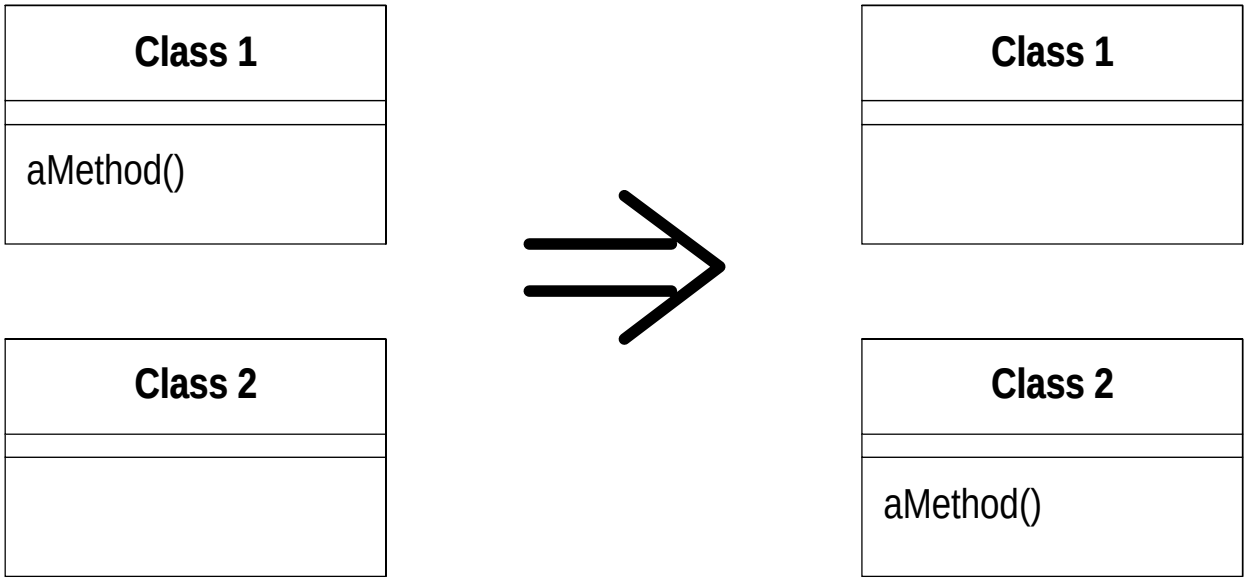
Change Names of Variables

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

Move Method

A method is, or will be, using or used by more features of another class than the class it is defined on.

Create a new method with a similar body in the class it uses most. Either turn the old method into a simple delegation, or remove it altogether.

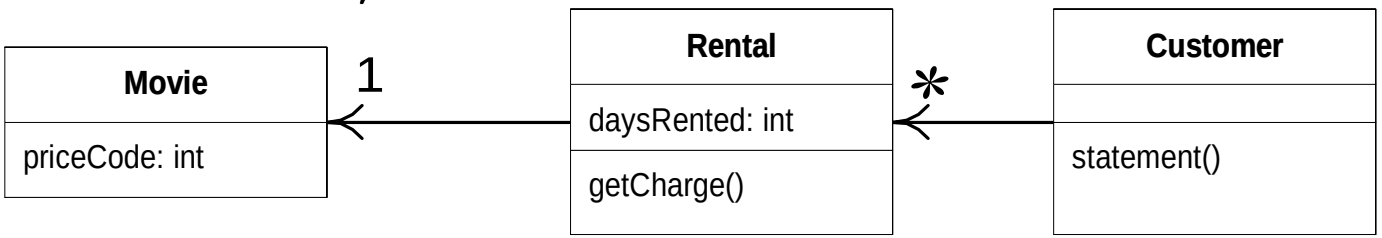


Steps for Move Method

- Declare method in target class
- Copy and fit code
- Set up a reference from the source object to the target
- Turn the original method into a delegating method
  - amountOf(Rental each) {return each.charge()}
  - Check for overriding methods
- Compile and test
- Find all users of the method
  - Adjust them to call method on target
- Remove original method
- Compile and test

Moving Amount() to Rental

```
class Rental                          ...
  double getCharge() {
      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;
  }
```



Altered Statement

```
class Customer...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            double thisAmount = 0;
            Rental each = (Rental) rentals.nextElement();

         thisAmount = each.getCharge();
            // 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;
    }
```

Problems With Temps

```
class Customer...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            double thisAmount = 0;
            Rental each = (Rental) rentals.nextElement();

         thisAmount = each.getCharge();
            // 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;

    }
```

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 "hot spots"**
  - 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

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

Steps for *Replace Temp With Query*

- Find temp with a single assignment
- Extract Right Hand Side of assignment
- Replace all references of temp with new method
- Remove declaration and assignment of temp
- Compile and test

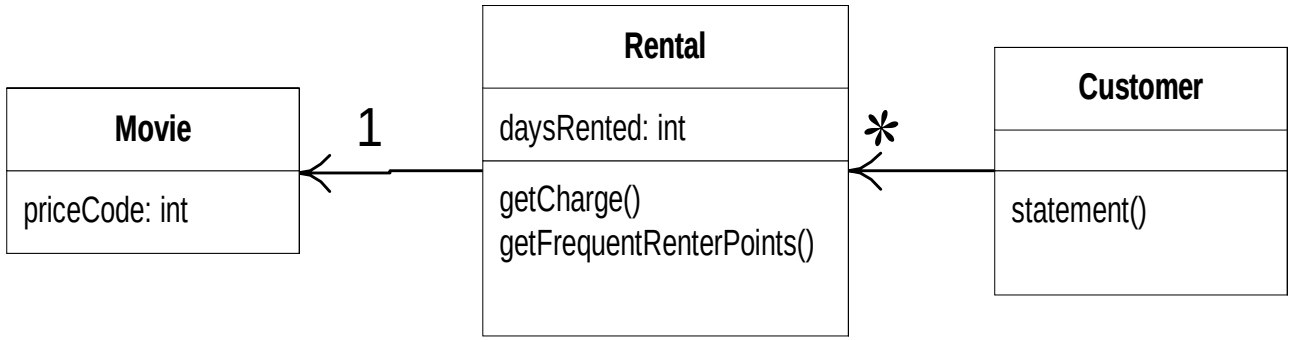
thisAmount Removed

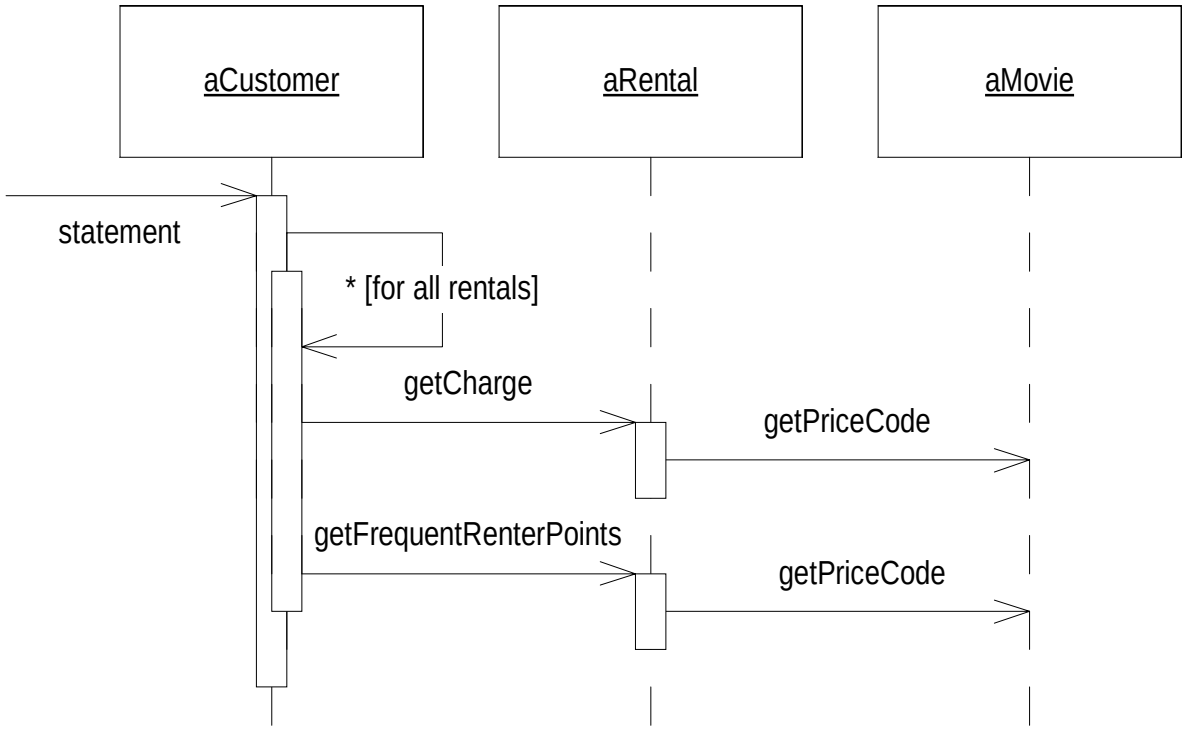
```
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();

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

Extract and Move frequentRenterPoints()

```
class Customer...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();
            frequentRenterPoints += each.getFrequentRenterPoints();

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

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

After Moving Charge and Frequent Renter Points





More Temps to Kill

```
class Customer...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();
            frequentRenterPoints += each.getFrequentRenterPoints();

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

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

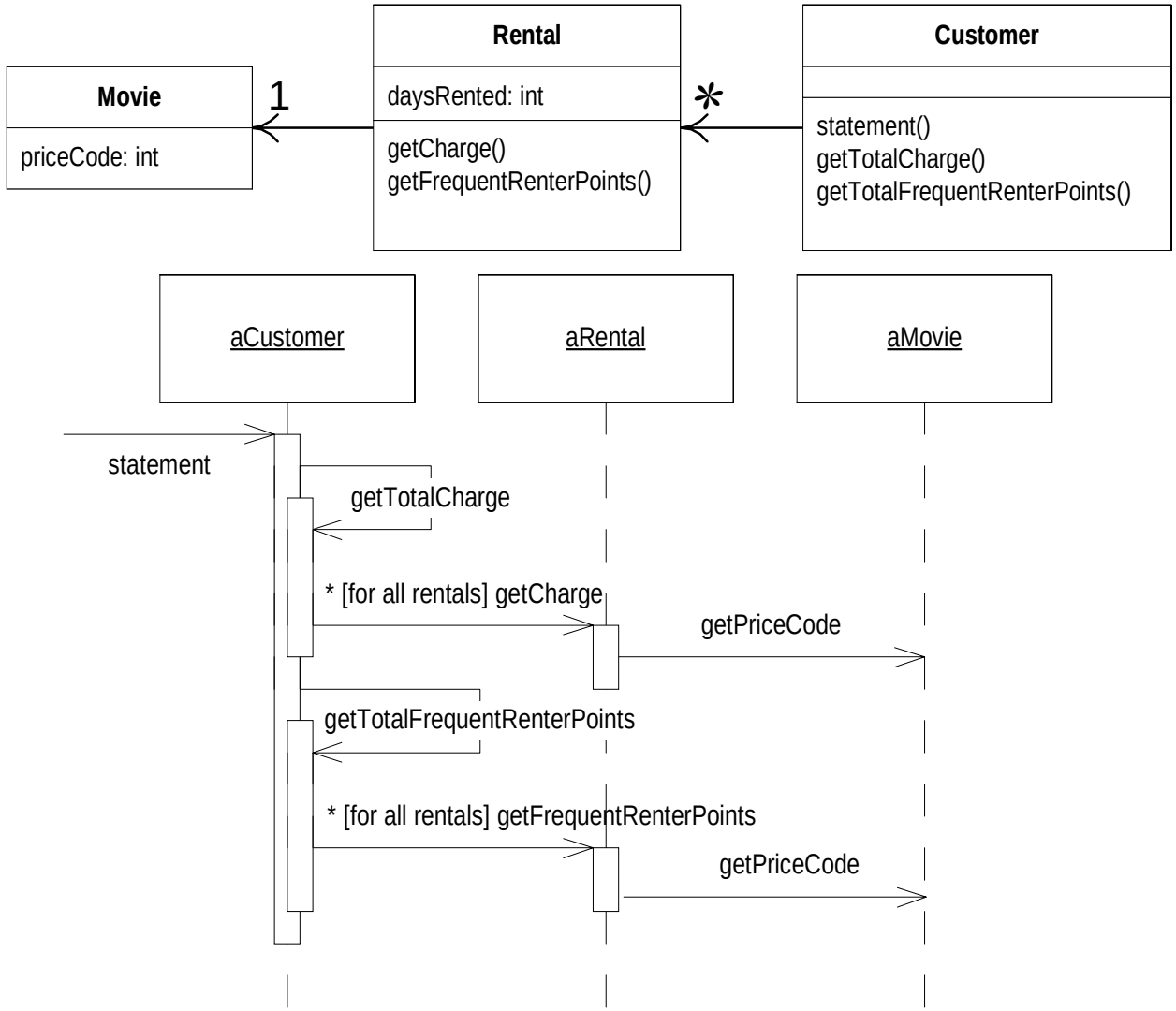
The New Methods

```
class Customer…

 private double getTotalCharge() {
        double result = 0;
        Enumeration rentals = _rentals.elements();
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();
            result += each.getCharge();
        }
        return result;
  }

 private int getTotalFrequentRenterPoints(){
        int result = 0;
        Enumeration rentals = _rentals.elements();
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();
            result += each.getFrequentRenterPoints();
        }
        return result;
  }
```

The Temps Removed

```
public String statement() {
    Enumeration rentals = _rentals.elements();
    String result = "Rental Record for " + getName() + "\n";
    while (rentals.hasMoreElements()) {
      Rental each = (Rental) rentals.nextElement();


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

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

After Replacing the Totals



htmlStatement()

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

The Current getCharge Method

```
class Rental                    ...
  double getCharge() {
      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;
  }
```

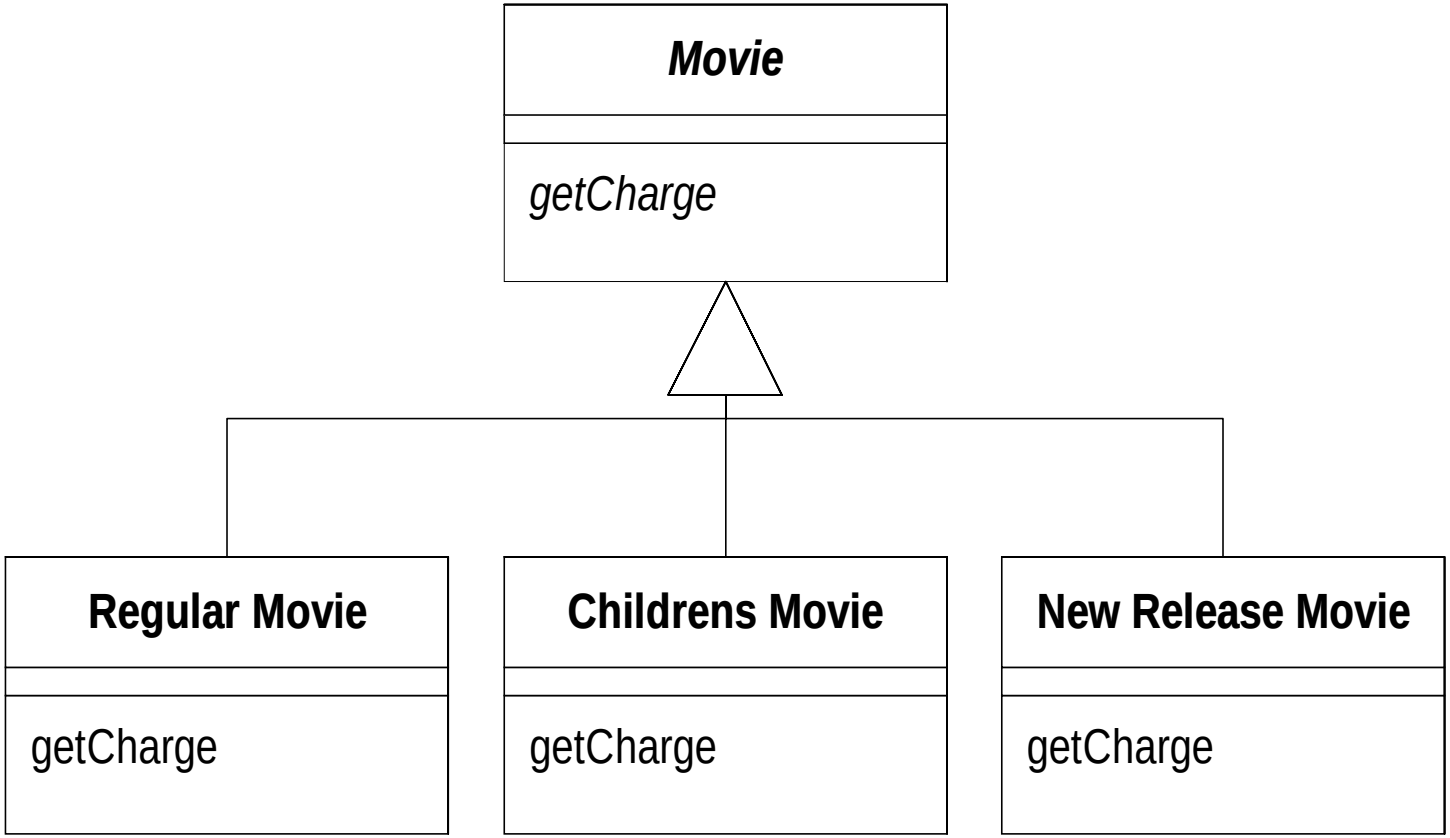
getCharge Moved to Movie

```
class Rental...
  double getCharge() {
    return _movie.getCharge(_daysRented);
}
```

```
class Movie …
  double getCharge(int daysRented) {
      double result = 0;
      switch (getPriceCode()) {
          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;
  }
```

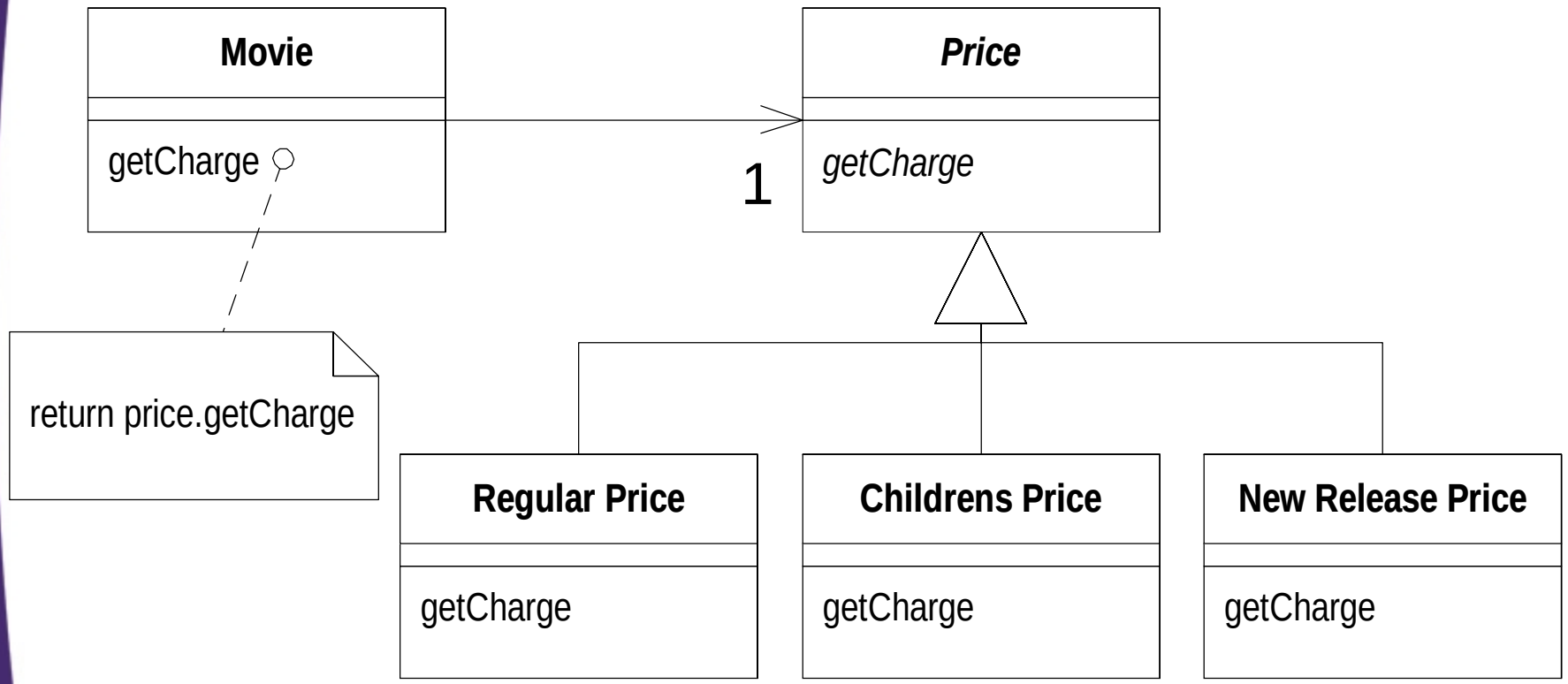
- Do the same with frequentRenterPoints()

Consider Inheritance



How's This?

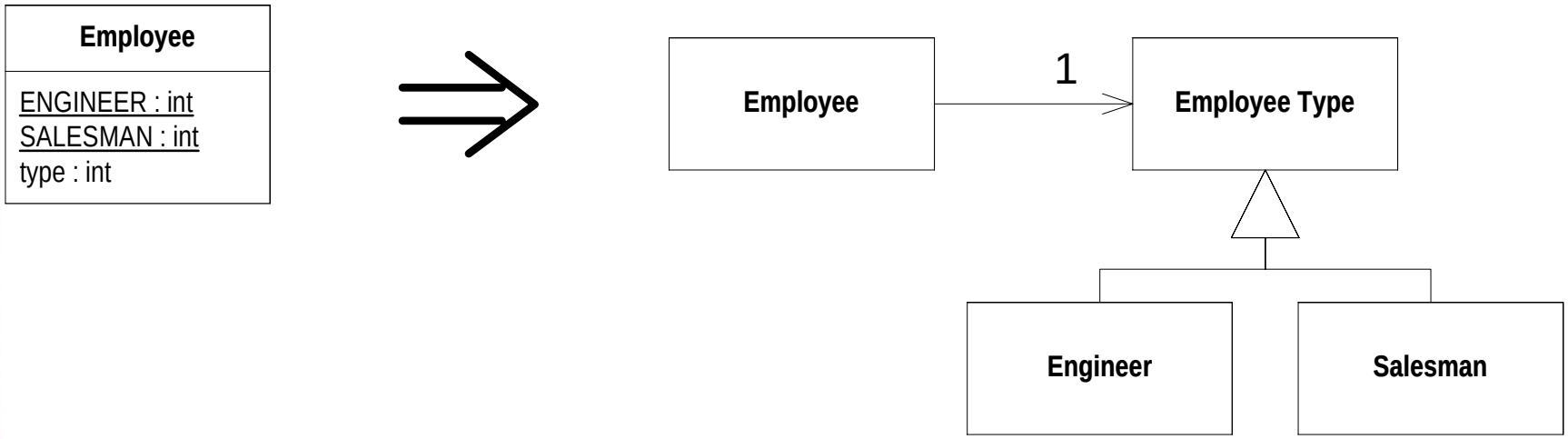
Using the State Pattern



Replace Type Code With State/Strategy

You have a type code which affects the behavior of a class but you cannot use subclassing.

Replace the type code with a state object.



Steps for *Replace Type Code With State/Strategy*

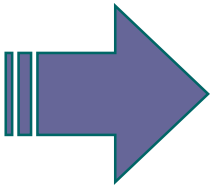
- Create a new state class for the type code
- Add subclasses of the state object, one for each type code
- Create an abstract query in the superclass to return the type code. Override in subclasses to return correct type code
- Compile
- Create field in old class for the state object
- Change the type code query to delegate to the state object
- Change the type code setting methods to assign an instance of the subclass
- Compile and test

Price Codes on the Price Hierarchy

```
abstract class Price {
  abstract int getPriceCode();
}
class ChildrensPrice extends Price {
  int getPriceCode() {
    return Movie.CHILDRENS;
  }
}
class NewReleasePrice extends Price {
  int getPriceCode() {
    return Movie.NEW_RELEASE;
  }
}
class RegularPrice extends Price {
  int getPriceCode() {
    return Movie.REGULAR;
  }
}
```

Change Accessors on Movie

```
public int
getPriceCode() {
  return _priceCode;
}
public setPriceCode
(int arg) {
  _priceCode = arg;
}
private int
_priceCode;
```



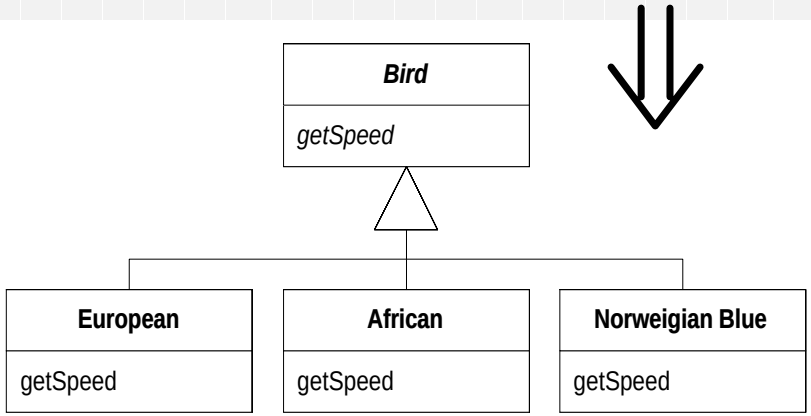
```
public int getPriceCode() {
  return _price.getPriceCode();
}
public void setPriceCode
  (int arg) {
  switch (arg) {
    case REGULAR:
      _price = new RegularPrice();
      break;
    case CHILDRENS:
      _price = new ChildrensPrice();
      break;
    case NEW_RELEASE:
      _price = new NewReleasePrice();
      break;
    default:
      throw new
IllegalArgumentException
      ("Incorrect Price Code");
  }
}
private Price _price;
```

Replace Conditional With Polymorphsim

You have a conditional that chooses different behavior depending on the type of an object

Move each leg of the conditional to an overriding method in a subclass. Make the original method abstract

```
double getSpeed() {
    switch (_type) {
        case EUROPEAN:
           return getBaseSpeed();
        case AFRICAN:
           return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
        case NORWEIGIAN_BLUE:
           return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("Should be unreachable");
}
```



Steps for *Replace Conditional With Polymorphism*

- Move switch to superclass of inheritance structure
- Copy one leg of case statement into subclass
- Compile and test
- Repeat for all other legs
- Replace case statement with abstract method

Move getCharge To Price

```
class Movie…
double getCharge(int daysRented) {
  return _price.getCharge(daysRented);
}

class Price…
double getCharge(int daysRented) {
    double result = 0;
    switch (getPriceCode()) {
        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;
}
```

Override getCharge in the Subclasses

```
Class RegularPrice…
    double getCharge(int daysRented){
        double result = 2;
        if (daysRented > 2)
            result += (daysRented - 2) * 1.5;
        return result;
    }
Class ChildrensPrice
    double getCharge(int daysRented){
        double result = 1.5;
        if (daysRented > 3)
            result += (daysRented - 3) * 1.5;
        return result;
    }

Class NewReleasePrice…
    double getCharge(int daysRented){
        return daysRented * 3;
}
```

Do each leg one at a time then...

```
Class Price…
    abstract double getCharge(int daysRented);
```

Similar Statement Methods

```
 public String statement() {
   Enumeration rentals = _rentals.elements();
   String result = "Rental Record for " + getName() + "\n";
   while (rentals.hasMoreElements()) {
      Rental each = (Rental) rentals.nextElement();
      result += "\t" + each.getMovie().getTitle()+ "\t" +
         String.valueOf(each.getCharge()) + "\n";
         }
   result += "Amount owed is " + String.valueOf(getTotalCharge()) + "\n";
   result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) +
           " frequent renter points";
   return result;
}

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

Form Template Method

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

Give each step into methods with the same signature, so that the original methods become the same. Then you can pull them up.

Steps for *Form Template Method*

- Take two methods with similar overall structure but varying pieces
  - Use subclasses of current class, or create a strategy and move the methods to the strategy
- At each point of variation extract methods from each source with the the same signature but different body
- Declare signature of extracted method in superclass and place varying bodies in subclasses
- When all points of variation have been removed, move one source method to superclass and remove the other

Create a Statement Strategy

```
class Customer …
public String statement() {
    return new TextStatement().value(this);
}
```

- Do the same with htmlStatement()

```
class TextStatement {
  public String value(Customer aCustomer) {
    Enumeration rentals = aCustomer.getRentals();
    String result = "Rental Record for " + aCustomer.getName() + "\n";
    while (rentals.hasMoreElements()) {
      Rental each = (Rental) rentals.nextElement();
      result += "\t" + each.getMovie().getTitle()+ "\t" +
          String.valueOf(each.getCharge()) + "\n";
       }
    result +=  "Amount owed is " +
      String.valueOf(aCustomer.getTotalCharge()) + "\n";
    result += "You earned " +
      String.valueOf(aCustomer.getTotalFrequentRenterPoints()) +
      " frequent renter points";
    return result;
   }
```

Extract Differences

```
class TextStatement...
  public String value(Customer aCustomer) {
    Enumeration rentals = aCustomer.getRentals();
    String result = headerString(aCustomer);
    while (rentals.hasMoreElements()) {
      Rental each = (Rental) rentals.nextElement();
      result += "\t" + each.getMovie().getTitle()+ "\t" +
        String.valueOf(each.getCharge()) + "\n";
    }
    result +=  "Amount owed is " +
      String.valueOf(aCustomer.getTotalCharge()) + "\n";
    result += "You earned " +
      String.valueOf(aCustomer.getTotalFrequentRenterPoints()) +
      " frequent renter points";
    return result;
  }
  String headerString(Customer aCustomer) {
    return "Rental Record for " + aCustomer.getName() + "\n";
  }

class HtmlStatement...
  String headerString(Customer aCustomer) {
       return "<H1>Rentals for <EM>" + aCustomer.getName() + "</EM></H1><P>\n";
  }
```

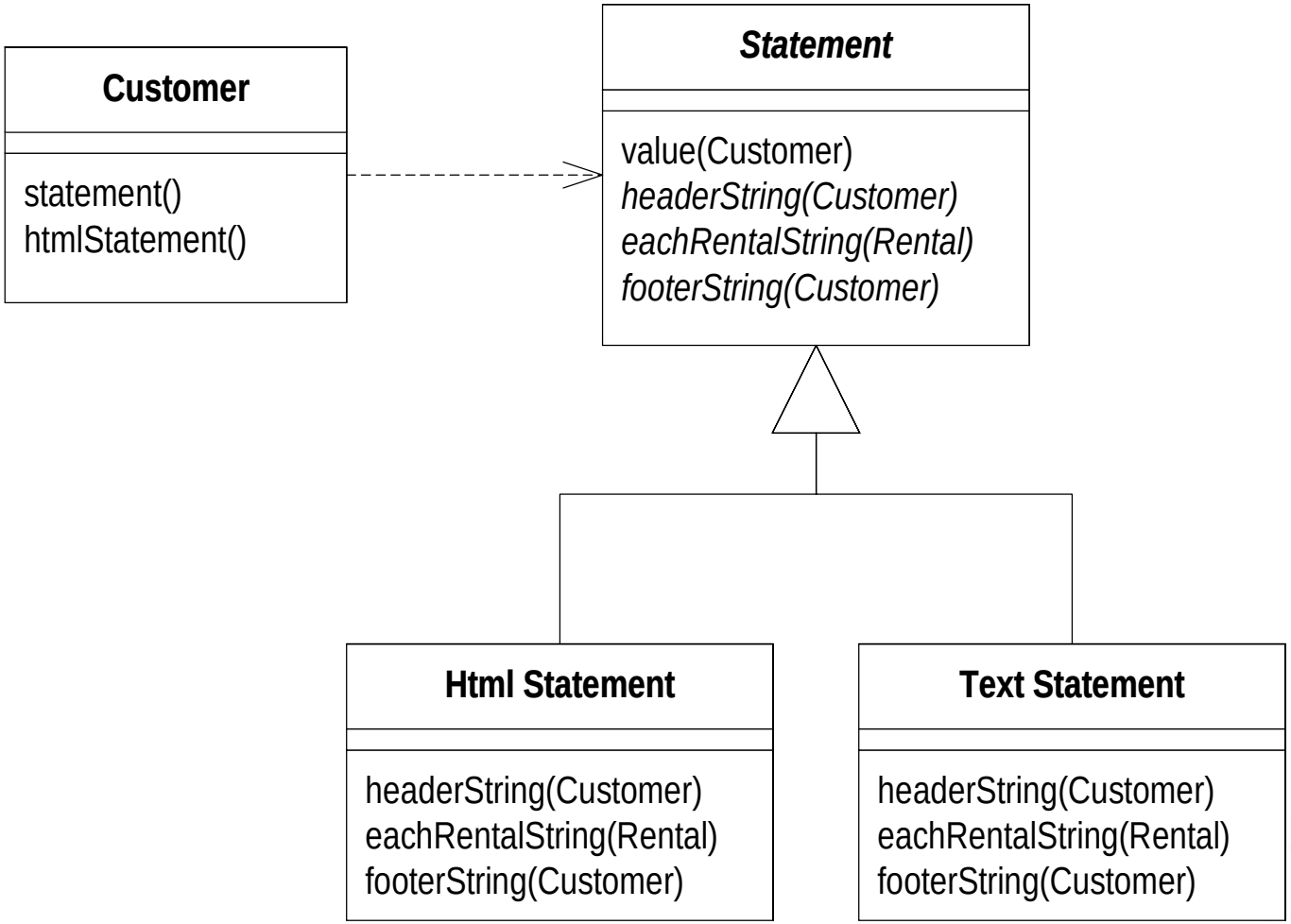
Continue Extracting

```
class TextStatement …
 public String value(Customer aCustomer) {
   Enumeration rentals = aCustomer.getRentals();
   String result = headerString(aCustomer);
   while (rentals.hasMoreElements()) {
     Rental each = (Rental) rentals.nextElement();
     result += eachRentalString(each);
   }
   result +=  footerString(aCustomer);
   return result;
 }

 String eachRentalString (Rental aRental) {
   return "\t" + aRental.getMovie().getTitle()+ "\t" +
     String.valueOf(aRental.getCharge()) + "\n";
   }

 String footerString (Customer aCustomer) {
   return "Amount owed is " +
     String.valueOf(aCustomer.getTotalCharge()) + "\n" +
     "You earned " +
     String.valueOf(aCustomer.getTotalFrequentRenterPoints()) +
     " frequent renter points";
 }
```

Pull Up the Value Method

```
class Statement...
public String value(Customer aCustomer) {
        Enumeration rentals = aCustomer.getRentals();
        String result = headerString(aCustomer);
        while (rentals.hasMoreElements()) {
            Rental each = (Rental) rentals.nextElement();
            result += eachRentalString(each);
        }
        result +=  footerString(aCustomer);
        return result;
    }

abstract String headerString(Customer aCustomer);
abstract String eachRentalString (Rental aRental);
abstract String footerString (Customer aCustomer);
```

State of Classes



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
- Illustrated several refactorings
  - Extract Method
  - Move Method
  - Replace Temp with Query
  - Replace Type Code with State/Strategy
  - Replace Switch with Polymorphism
  - 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 Refactoring Browser

Refactoring Tools

- Based on provable transformations
  - Build parse tree of programs
  - Mathematic proof that refactoring does not change semantics
  - Embed refactoring in tool
- Speeds up refactoring
  - Extract method: select code, type in method name
  - No need for tests (unless dynamic reflection)
  - Big speed improvement
- Not Science Fiction
  - Available for Smalltalk

  http://st-www.cs.uiuc.edu/~brant/RefactoringBrowser

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
  - Return “OK” if good, errors if not
- Run a suite with a single command
- Test with every compile

www.xprogramming.com/software

The Two Hats

Adding Function

- Add new capabilities to the system
- Adds new tests
- Get the test working

Refactoring

- Does not add any new features
- Does not add tests (but may change some)
- Restructure the code to remove redundancy

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

Why Refactor

- To improve the software design
  - Combat's "bit rot"
  - Makes the program easier to change
- To make the software easier to understand
  - Write for people, not the compiler
  - Understand unfamiliar code
- To help find bugs
  - Refactor while debugging to clarify the code

Refactoring helps you program faster!

When Should You Refactor?

- To add new functionality
  - Refactor existing code until you understand it
  - Refactor the design to make it easy to add
- To find bugs
  - Refactor to understand the code
- For code reviews
  - Rmmediate effect of code review
  - Allows for higher level suggestions

Don't set aside time for refactoring, include it in your normal activities

What Do You Tell Your Manager

- If the manager is *really* concerned about quality
  - Then stress the quality aspects
- Otherwise you need to develop as fast as possible
  - You're the professional, so you know to do what makes you go faster

Problems With Refactoring

- We don’t know what they are yet
- Database Migration
  - Insulate persistent database structure from your objects
  - With OO databases, migrate frequently
- Published Interfaces
  - Publish only when you need to
  - Don’t publish within a development team
- Without working tests
  - Don’t bother

Design Decisions

- In the moment
  - Consider current needs
  - Patch code when new needs appear
- Planned design
  - Consider current needs and possible future needs
  - Design to minimize change with future needs
  - Patch code if unforeseen need appears
- Evolutionary design
  - Consider current needs and possible future needs
  - Trade off cost of current flexibility versus cost of later refactoring
  - Refactor as changes appear

Extreme Programming

- Methodology developed by Kent Beck
- Designed to adapt to changes
- Key Practices
  - Iterative Development
  - Self Testing Code
  - Refactoring
  - Pair Programming
- Leverages refactoring to encourage evolutionary design

Beck, K. *Extreme Programming Explained,* Addison-Wesley

Team Techniques

- Encourage refactoring culture
  - Nobody gets things right first time
  - Nobody can write clear code without reviews
  - Refactoring is forward progress
- Provide sound testing base
  - Tests are essential for refactoring
  - Build system and run tests daily
- Pair Programming
  - Two programmers working together can be quicker than working separately
  - Refactor with the class writer and a class user

Creating Your Own Refactorings

- Consider a change to a program
- Should it change the external behavior of the system
- Break down the change into small steps
  - Look for points where you can compile and test
- Carry out the change, note what you do
  - If a problem occurs, consider how to eliminate it in future
- Carry it out again, follow and refine the notes
- After two or three times you have a workable refactoring

Final Thoughts

- The one benefit of objects is that they make it easier to change.
- Refactoring allows you to improve the design after the code is written
- Up front design is still important, but not so critical
- Refactoring is an immature subject: not much written and very few tools


JavaOne
SM
Sun's 2000 Worldwide Java Developer Conference