Refactoring

Lecture 7

Refactoring

- Martin Fowler (and Kent Beck, John Brant, William Opdyke, Don Roberts), Refactoring- Improving the Design of Existing Code, Addison Wesley, 1999.
- **Refactoring** (noun):

  a change made to the internal structure of software to make it

  easier to understand and

  cheaper to modify

  without changing its observable behavior.
- **Refactor** (verb):

  to restructure software by applying

  a series of refactorings.

Refactoring, applied

- Straight from the book:

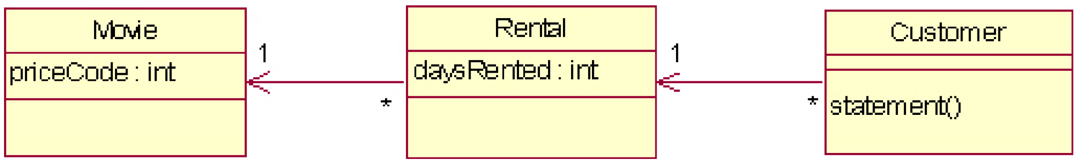
  "a program to calculate and print a statement of a customer's charges at a video store"

  ...price depends on how long the movie is rented and the category of the movie

  ...also compute frequent renter points

Refactoring: Movie

- Class diagram of the starting point classes.



Refactoring: Movie Class

```
public class Movie {
   public static final int CHILDREN=2;
   public static final int REGULARS=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;
   }
} // end of Movie
```

Refactoring: Rental Class

```
public 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;
   }
} // end of Rental
```

Refactoring: Customer Class

```
public 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;
   }
   . . .
```

Refactoring: Customer Class

```
public class Customer
...
   public String statement() {
     double totalAmount          = 0;
     int frequentRenterPoints = 0;
     Enumeration rentals         = _rental.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;
```

Refactoring: Customer Class

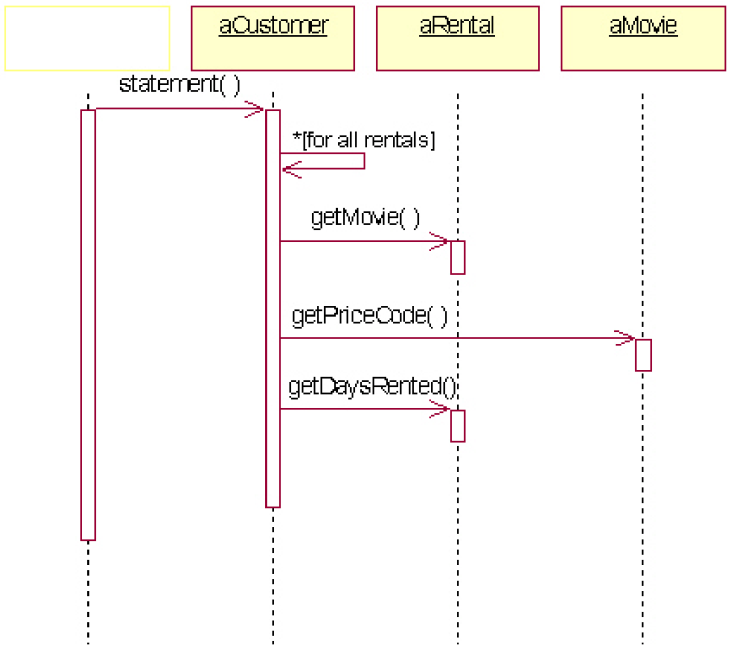
```
public class Customer
    public String statement()
    ...
            case Movie.NEW_RELEASE:
                   thisAmount += each.getDaysRented() * 3; break;
            case Movie.CHILDRENS:
                   thisAmount += 1.5;
                   if (each.getDaysRented() > 3)
                      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++;
```

Refactoring: Customer Class

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

Refactoring

- Interaction diagram for the statement method.



Refactoring: problem statement

Add a `htmlStatment` method which returns a customer statement string containing html tags.

...and there will be some changes to the way movies are classified

...affecting frequent renter points and charging.

Refactoring: step 1

- Write a test suite !
- Refactoring should not affect the outcome of tests. The test suite must exercise the published interface of the classes.
- Obviously, refactoring should not affect the published interface. So, avoid publishing interfaces too early.

Refactoring: step 2

- statement() is overly long, apply the Extract Method refactoring

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

Refactoring: step 2

```
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.elements();
        String result = "Rental Record for " + getName() +  "\n";
        while (rentals.hasMoreElements()) {
                double thisAmount = 0;
                Rental each = (Rental) rentals.nextElement();
                thisAmount = amountFor(each);
                frequentRenterPoints ++;
                if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE)&&
                    each.getDaysRented() > 1) frequentRenterPoints++;
                result += "\t" + each.getMovie().getTitle()+ "\t" +
                        String.valueOf(thisAmount) + "\n";
                totalAmount += thisAmount;
        }
        result += "Amount owed is "+Sting.valueOf(totalAmount) +
    "\n";                   result += "You earned
    "+Sting.valueOf(frequentRenterPoints)
                + "frequent renter points\n";
        return result;
```

Refactoring: step 2

```
public 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;
}
```

Refactoring: step 3

- **TEST**

Refactoring: step 4

- **oops, (double) -> (int) bug!**

```
public 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;
}
```

Refactoring: step 5

- Variable names not helpful

```
public 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;
}
```

Refactoring: step 5

```
public 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 ;
}
```

Refactoring: step 6

- Moving amount computation (does not use info from Customer only Rental)

```
class Customer ...
public 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;}
```

Refactoring: step 6

```
class Rental ...
public 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;}
```

Refactoring: step 6

```
class Customer ...
public double amountFor(Rental aRental) {
   return aRental.getCharge();
}
```

Refactoring: step 7

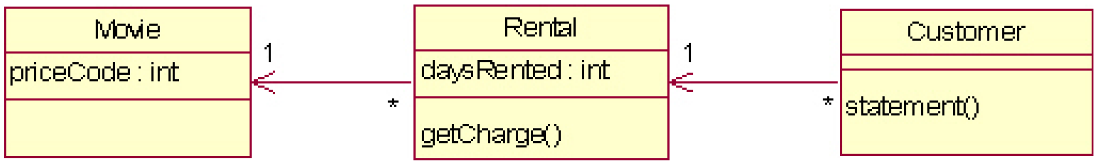
```
class Customer ...
public String statement() {
       double totalAmount = 0;
       int frequentRenterPoints = 0;
       Enumeration rentals = _rental.elements();
       String result = "Rental Record for " + getName() +  "\n";
       while (rentals.hasMoreElements()) {
              double thisAmount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

Refactoring: step 7

```
class Customer ...
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.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;
```

Refactoring

- State of classes after moving the charge method. amountFor has been deleted.



Refactoring: step 8

- Replace Temp with Query (thisAmount is redundant)

```
class Customer ...
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.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();
        }
```

Refactoring: step 9

- Extract Method (frequent renter computation)

```
class Customer ...
public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.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();
        }
```

Refactoring: step 9

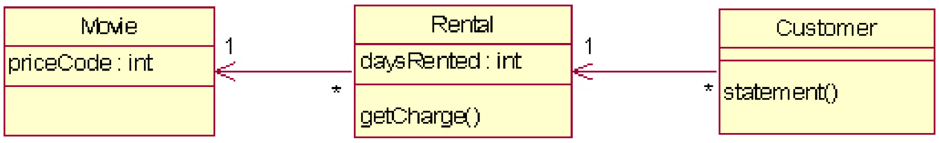
```
class Customer ...
  public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.elements();
        String result = "Rental Record for " + getName() +  "\n";
        while (rentals.hasMoreElements()) {
                Rental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

Refactoring: step 9

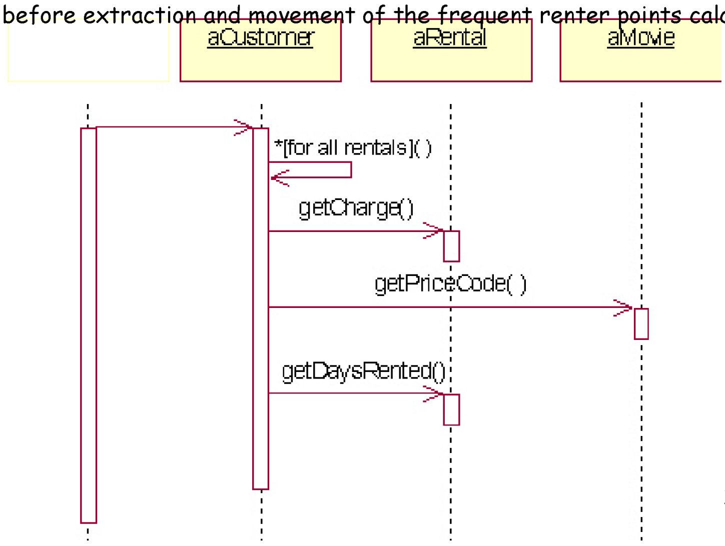
```
class Rental ...
public int getFrequentRenterPoints() {
        if ((getMovie().getPriceCode() == Movie.NEW_RELEASE)
            && getDaysRented() > 1)
                return 2;
        else
                return 1;
}
```

Refactoring

- Class diagram before extraction and movement of the frequent renter points calculation

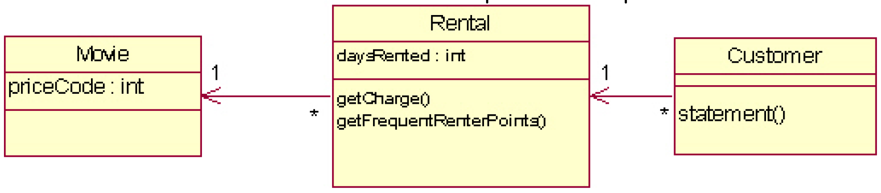


- Interaction diagram before extraction and movement of the frequent renter points calculation

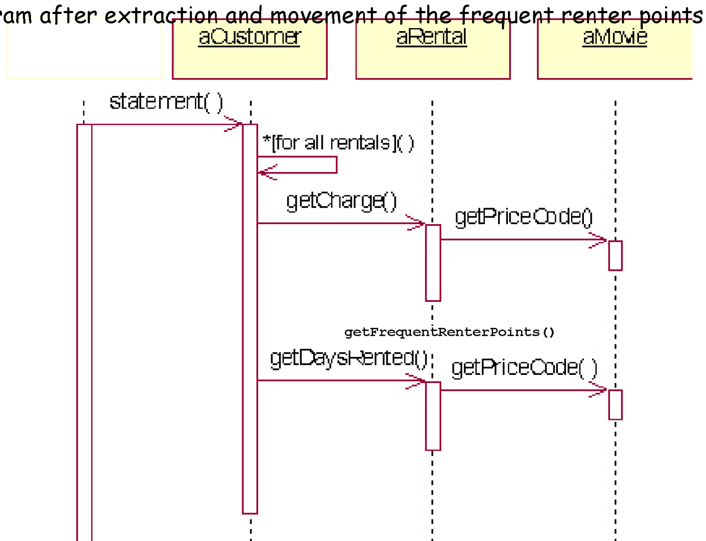


Refactoring

- Class diagram after extraction and movement of the frequent renter points calculation



- Interaction diagram after extraction and movement of the frequent renter points calculation



Refactoring: step 10

- Replace Temp with Query (the temporaries make the method complex and force code duplication)

```
class Customer ...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.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();
        }
```

Refactoring: step 10

```
class Customer ...
   public String statement() {
        int frequentRenterPoints = 0;
        Enumeration rentals = _rental.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";
        }
        // add footer lines
        result += "Amount owed is "+Sting.valueOf(getTotalCharge()) +
   "\n";        result += "You earned "+Sting.valueOf(frequentRenterPoints)
   +
                "frequent renter points\n";
        return result;
   }
```

Refactoring: step 10

```
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;
}
```

Refactoring: step 11

- Replace Temp with Query

```
class Customer ...
public String statement() {
       int frequentRenterPoints = 0;
       Enumeration rentals = _rental.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";
       }
       // add footer lines
       result += "Amount owed is "+Sting.valueOf(getTotalCharge()) +
   "\n";        result += "You earned "+Sting.valueOf(frequentRenterPoints)
   +
              "frequent renter points\n";
       return result;}
```

Refactoring: step 11

- Replace Temp with Query

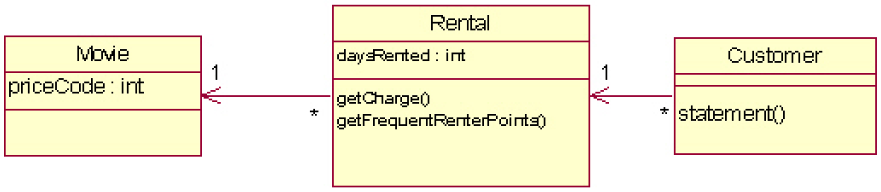
```
class Customer ...
public String statement() {
        Enumeration rentals = _rental.elements();
        String result = "Rental Record for " + getName() +  "\n";
        while (rentals.hasMoreElements()) {
                Rental each = (Rental) rentals.nextElement();
                //show figures for this rental
                result += "\t" + each.getMovie().getTitle()+ "\t" +
                        String.valueOf(each.getCharge()) + "\n";
        }
        // add footer lines
        result += "Amount owed is "+Sting.valueOf(getTotalCharge()) + "\n";
        result += "You earned "+Sting.valueOf(getFrequentRenterPoints())+
                "frequent renter points\n";
        return result;
    }
```

Refactoring: step 11

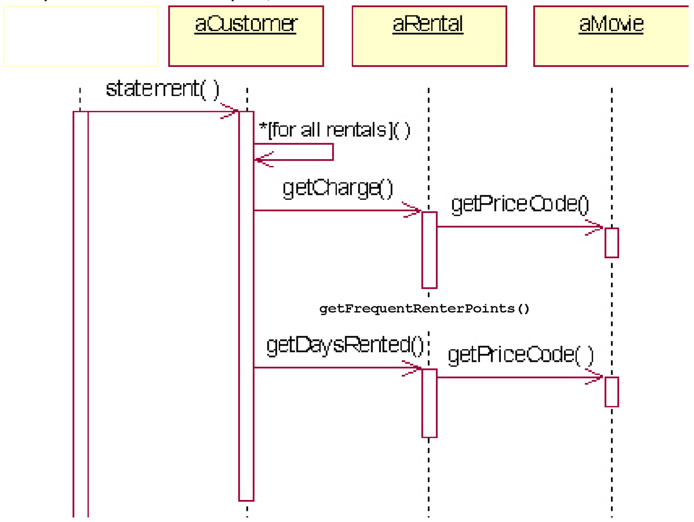
```
class Customer ...
   private double getFrequentRenterPoints() {
        double result = 0;
        Enumeration rentals = _rentals.elements();
        while (rentals.hasMoreElements()) {
                Rental each = (Rental) rentals.nextElement();
                result += each.getFrequentRenterPoints();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

Refactoring

- Class diagram before extraction of the totals

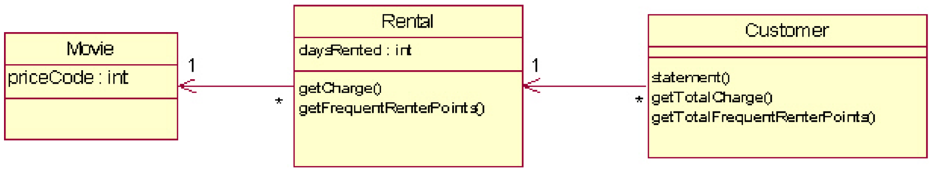


- Interaction diagram before extraction of the totals

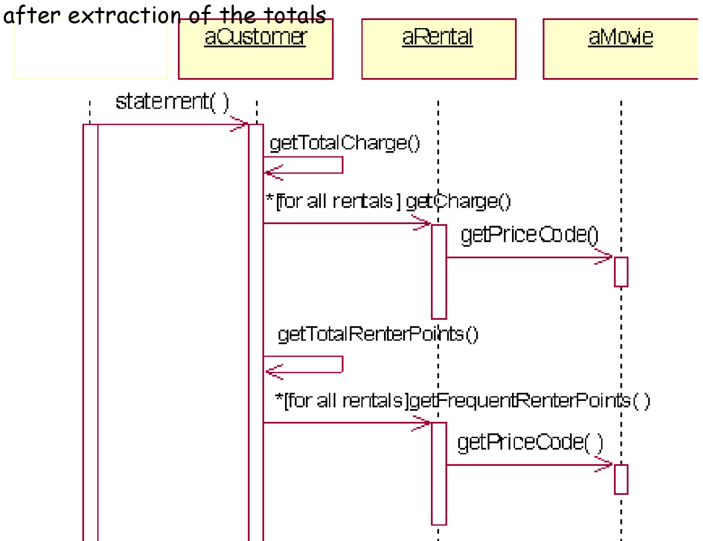


Refactoring

- Class diagram after extraction of the totals



- Interaction diagram after extraction of the totals



Refactoring

- Remarks
  - Most refactoring reduce code size, but this is not necessarily the case. The point is to make code easier to modify and more readable.
  - Performance gets a hit by running the same loop three times, or does it? Profile the program and find the answer.

Software extension

- The requested method can be added with minimal code duplication

```
class Customer ...
public String htmlStatement() {
        Enumeration rentals = _rental.elements();
        String result = "<H1>Rental Record for<EM> "+getName()+ "<EM></
   H1><P>\n";
        while (rentals.hasMoreElements()) {
                Rental each = (Rental) rentals.nextElement();
                //show figures for this rental
                result +=  each.getMovie().getTitle()+ ": " +
                        String.valueOf(each.getCharge()) + "<BR>\n";
        }
        // add footer lines
        result +="<P>Amount owed is<EM> "+Sting.valueOf(getTotalCharge())
                        + "</EM><P>\nYou earned <EM>";
        return result + Sting.valueOf(getFrequentRenterPoints())
                 +"</EM> frequent renter points<P>\n";
   }
```

New functionality

- Getting ready to change the classification of the movies in the store.
- Perhaps new classification, perhaps modification to existing.
- Charging and frequent renting will be affected.

Refactoring: step 12

- Replacing conditional logic on Price Code with polymorphism

Refactoring: step 12

- Move getCharge

```
class Rental ...
public 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;}
```

Refactoring: step 12

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

Refactoring: step 12

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

Refactoring: step 13

- Move getFrequentRenterPoints()

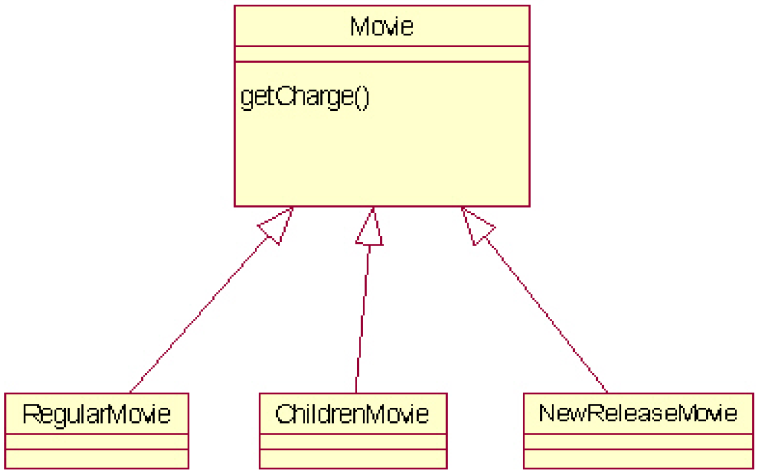
```
class Rental ...
public int getFrequentRenterPoints() {
        if ((getMovie().getPriceCode() == Movie.NEW_RELEASE)
         && getDaysRented() > 1)
                return 2;
        else
                return 1;
}
```

Refactoring: step 13

```
class Movie ...
public int getFrequentRenterPoints(int daysRented) {
       if ((getPriceCode() == Movie.NEW_RELEASE) && daysRented > 1)
              return 2;
       else
              return 1;
}



class Rental ...
public int getFrequentRenterPoints() {
   return
   _movie.getFrequentRenterPoints(_daysRented);
}
```

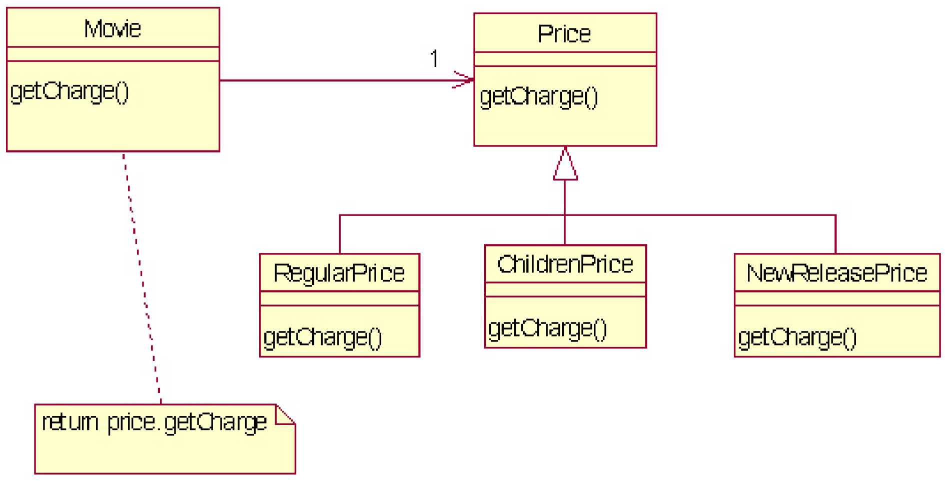
Refactoring

- Inheritance



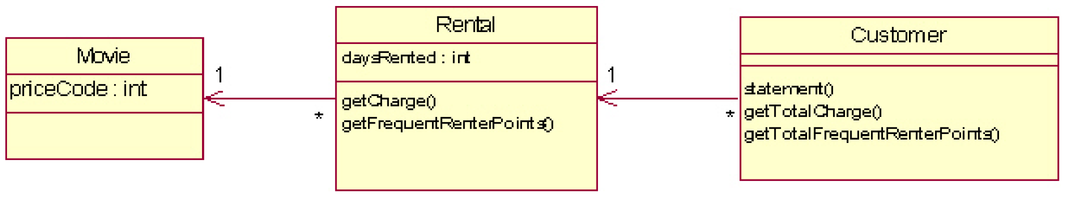
Refactoring

- Inheritance

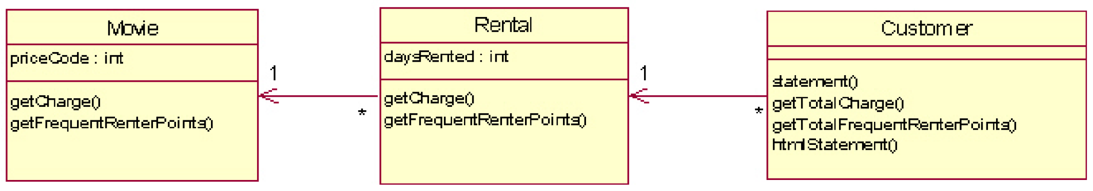


Refactoring

- Class diagram before moving methods to movie



- Class diagram after moving methods to movie



Refactoring: step 14

- Replace Type Code with State/Strategy

```
class Movie ...
   public Movie(String name, int priceCode) {
        _name = name;
        _priceCode = priceCode;
   }
```

Refactoring: step 14

```
class Movie ...
   public Movie(String name, int priceCode) {
        _name = name;
        setPriceCode(priceCode);
   }
```

Refactoring: step 14

```
abstract class Price {
   abstract int getPriceCode();
}

class ChildrenPrice extends Price {
   int getPriceCode(){
        return MOVIE.CHILDREN;
   }
}
class NewReleasePrice extends Price {
   int getPriceCode(){
        return MOVIE.NEW_RELEASE;
   }
}
class RegularPrice extends Price {
   int getPriceCode(){
        return MOVIE.REGULAR;
   }
}
```

Refactoring: step 15

```
class Movie ...

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

Refactoring: step 15

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

Refactoring: step 16

- Move Method

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

Refactoring: step 16

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

Refactoring: step 16

- Replace Conditional with Polymorphism

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

Refactoring: step 16

```
class RegularPrice ...
double getCharge(int daysRented) {
   double result = 2;
   if (getDaysRented() > 2) result +=(getDaysRented()-2) * 1.5;
   return result;
}
class NewReleasePrice ...
double getCharge(int daysRented) {
   return daysRented * 3;
}
class ChildrenPrice ...
double getCharge(int daysRented) {
   double result = 1.5;
   if (getDaysRented() > 3) result +=(getDaysRented()-3) * 1.5;
   return result ;
}
class Price...
   abstract double getCharge(int daysRented);
```

Refactoring: step 17

- Replace Conditional with Polymorphism

```
class Rental ...
 int getFrequentRenterPoints(int daysRented) {
        if ((getPriceCode() == Movie.NEW_RELEASE) && daysRented > 1)
                return 2;
        else
                return 1;
}
```

Refactoring: step 17

```
class Movie  ...
 int getFrequentRenterPoints(int daysRented) {
   return _price.getFrequentRenterPoints(daysRented);
}

class Price...
    int getFrequentRenterPoints(int daysRented) {
        return 1;
   }
class NewReleasePrice..
 int getFrequentRenterPoints(int daysRented) {
        return (daysRented > 1) ? 2:1;
}
```

Refactoring Principles

- Why do we refactor?
  - To improve the design of software
  - To make software easier to understand
  - To help you find bugs
  - To make you program faster
- When should we refactor?
  1. Refactor when you add functionality
  2. Refactor when you need to fix a bug
  3. Refactor as you do code reviews
  - Refactor when the code starts to smell.
- What about performance?
  - Worry about performance only when you have identified a performance problem

Bad Smells in Code

If it stinks, change it.

--Grandma Beck on child rearing

Duplicated Code (stench 10)

If the same code structure is repeated

- Extract Method - gather duplicated code
- Pull Up Field - move to a common parent
- Form Template Method - gather similar parts, leaving holes
- Substitute Algorithm - choose the clearer algorithm
- Extract class - for unrelated classes, create a new class with functionality

Bad Smells in Code

Long Method (stench 7)

If the body of a method is over a page (choose your page size)

- Extract Method - extract related behavior
- Replace Temp with Query - remove temporaries when they obscure meaning
- Introduce Parameter Object - slim down parameter lists by making them into objects
- Replace Method with Method Object - still too many parameters
- Decompose Conditionals - conditional and loops can be moved to their own methods

Bad Smells in Code

Large Class (stench 7)

If a class has either too many variables or too many methods

- Extract Class - to bundle variables/methods

Bad Smells in Code

Long Parameter List (stench 6)

A method does not need many parameter, only enough to be able to retrieve what it needs

- Replace Parameter with Method - turn a parameter into a message
- Introduce Parameter Object - turn several parameters into an object

Bad Smells in Code

Divergent Change (stench 5)

If you find yourself repeatedly changing the same class then there is probably something wrong with it

- Extract Class - group functionality commonly changed into a class

Bad Smells in Code

Shotgun Surgery (stench 5)

If you find yourself making a lot of small changes for each desired change

- Move Method/Field - pull all the changes into a single class
- Inline Class - group a bunch of behaviors together

Bad Smells in Code

Feature Envy (stench 6)

If a method seems more interested in a class other than the class it actually is in

- Move Method - move the method to the desired class
- Extract Method - if only part of the method shows the symptoms

Bad Smells in Code

Data Clumps (stench 4)

Data items that are frequently together in method signatures and classes belong to a class of their own

- Extract Class - turn related fields into a class
- Introduce Parameter Object - for method signatures

Bad Smells in Code

Primitive Obsession (stench 3)

Primitive types inhibit change

- Replace Data Value with Object - on individual data values
- Move Method/Field - pull all the changes into a single class
- Introduce Parameter Object - for signatures
- Replace Array with Object - to get rid of arrays

Bad Smells in Code

Switch Statements (stench 5)

Switch statements lead to duplication and inhibit change

- Extract method - to remove the switch
- Move method - to get the method where polymorphism can apply
- Replace Type Code with State/Strategy - set up inheritance
- Replace Conditional with Polymorphism - get rid of the switch

Bad Smells in Code

Parallel Inheritance Hierarchies (stench 6)

If when ever you make a subclass in one corner of the hierarchy, you must create another subclass in another corner

- Move Method/Field - get one hierarchy to refer to the other

Bad Smells in Code

Lazy Class (stench 4)

If a class (e.g. after refactoring) does not do much, eliminate it.

- Collapse Hierarchy- for subclasses
- Inline Class - remove a single class

Bad Smells in Code

Speculative Generality (stench 4)

If a class has features that are only used in test cases, remove them.

- Collapse Hierarchy- for useless abstract classes
- Inline Class - for useless delegation
- Rename Method - methods with odd abstract names should be brought down to earth

Bad Smells in Code

Temporary Field (stench 3)

If a class has fields that are only set in special cases, extract them

- Extract Class- for the special fields

Bad Smells in Code

Message Chains (stench 3)

Long chains of messages to get to a value are brittle as any change in the intermittent structure will break the code

- Hide Delegate - remove one link in a chain
- Extract Method - change the behavior to avoid chains

Bad Smells in Code

Middle Man (stench 3)

An intermediary object is used too often to get at encapsulated values

- Remove Middle Man - to talk directly to the target
- Replace Delegation with Inheritance - turns the middle man into a subclass of the real object

Bad Smells in Code

Inappropriate Intimacy (stench 5)

Classes are too intimate and spend too much time delving in each other's private parts

- Move Method/Field - to separate pieces in order to reduce intimacy
- Extract Class - make a common class of shared behavior/data
- Replace Inheritance with Delegation - when a subclass is getting too cozy

Bad Smells in Code

Comments (stench 2)

Comments are often a sign of unclear code... consider refactoring