Refactoring: Improving the Design of Existing Code

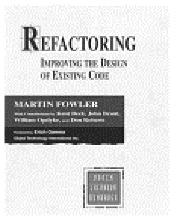
Martin Fowler

fowler@acm.org

http://ourworld.compuserve.com/homepages/Martin_Fowler

What we will cover

- ❑ A simple example of refactoring
  - ➢ Blow by blow example of changes
  - ➢ Steps for illustrated refactorings
- ❑ Background of refactoring
  - ➢ Where it came from
  - ➢ Tools
  - ➢ Why and When
- ❑ Unit testing with JUnit
  - ➢ Rhythm of development
- ❑ Bad Smells and their cures



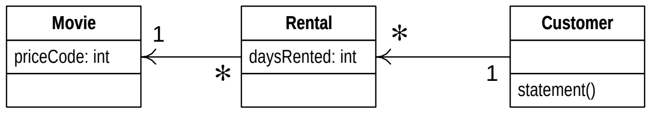
Fowler, *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999

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



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

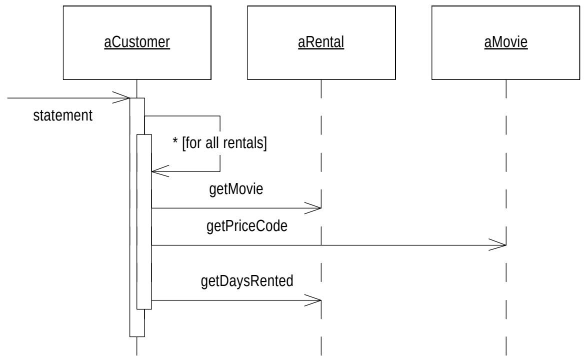
```
        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
           each.getDaysRented() > 1) frequentRenterPoints ++;

        //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



Sample Output

```
Rental Record for Dinsdale Pirhana
  Monty Python and the Holy Grail   3.5
  Ran2
  Star Trek 27       6
  Star Wars 3.2      3
  Wallace and Gromit 6
Amount owed is 20.5
You earned 6 frequent renter points
```

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;
}
```

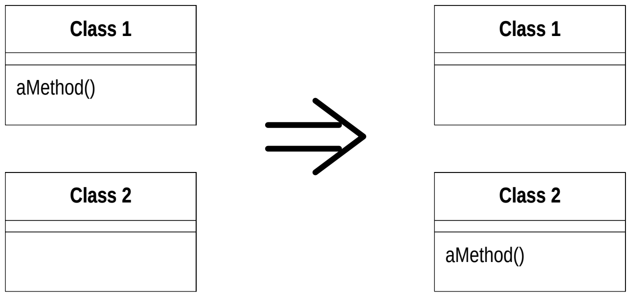
Is this important?

Is this method in the right place?

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.

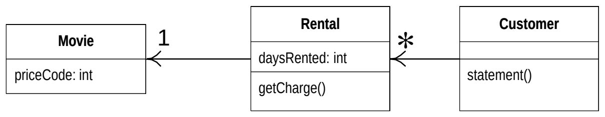


Steps for *Move method*

- ❑ 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.

```
        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;
    }
```

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;

    }
}
```

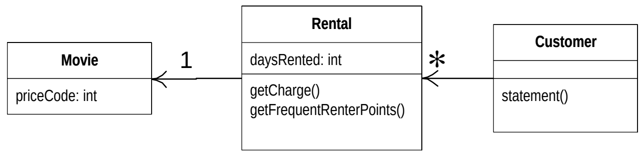
Extract and move frequentRenterPoints()

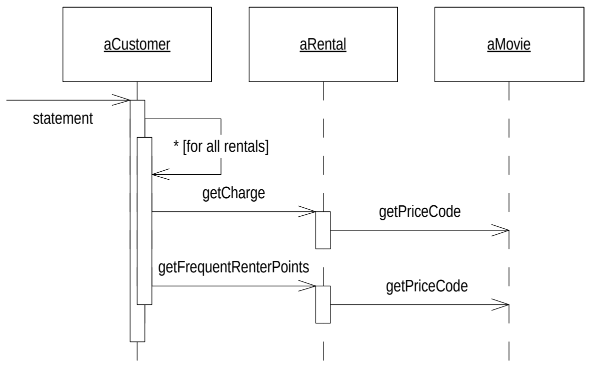
```
class Customer...
    public String statement() {
        double totalAmount = 0;
        int frequentRenterPoints = 0;
        Enumeration rentals = _rentals.elements();
        String result = "Rental Record for " + getName() + "\n";
        while (rentals.hasMoreElements()) {
            Rental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

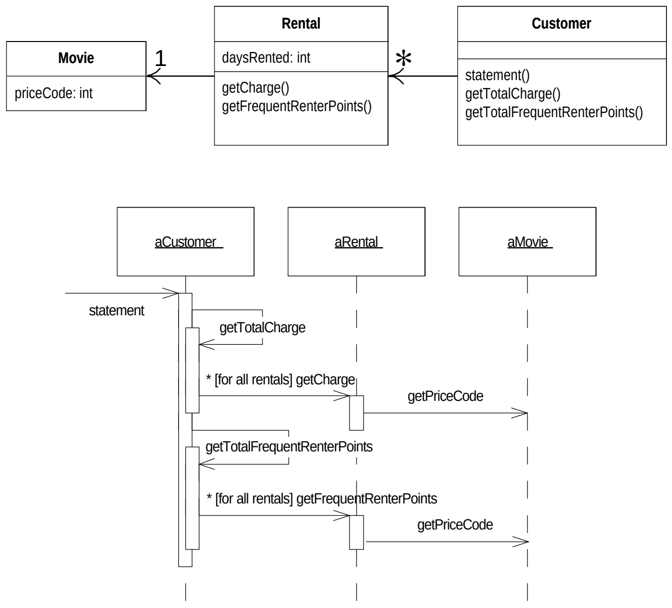
The temps removed

```
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;
   }
```

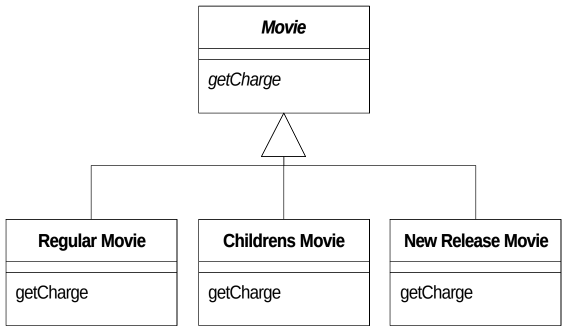
getCharge moved to Movie

```
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;
   }
```

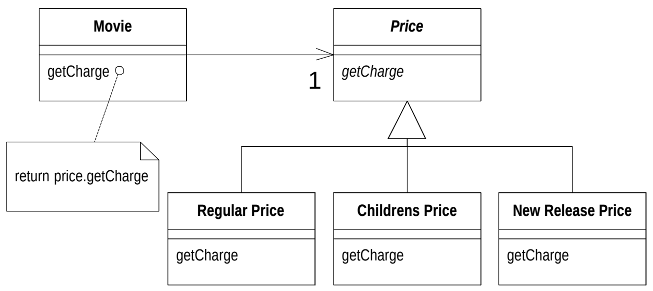
- **Do the same with frequentRenterPoints()**

Consider inheritance



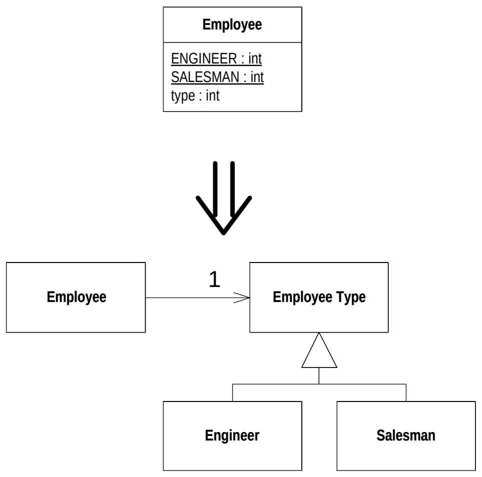
How's this?

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

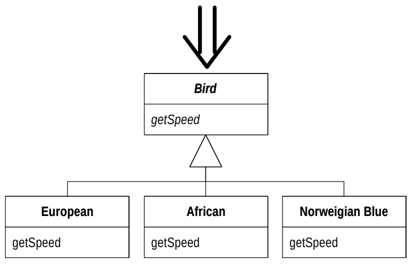
```
public int getPriceCode() {
   return _priceCode;
}
public setPriceCode (int arg) {
   _priceCode = arg;
}
private int _priceCode;
```

```
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);
```

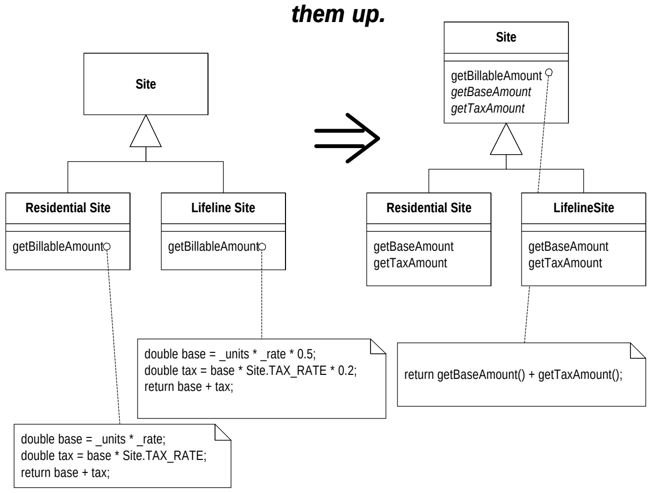
Similar Statement Methods

```
 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.



```
double base = _units * _rate * 0.5;
double tax = base * Site.TAX_RATE * 0.2;
return base + tax;
```

```
return getBaseAmount() + getTaxAmount();
```

```
double base = _units * _rate;
double tax = base * Site.TAX_RATE;
return base + tax;
```

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);
}

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;
    }
```

❑ **Do the same with htmlStatement()**

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";
   }
```

❑ **Do the same with htmlStatement**

```
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";
 }
```

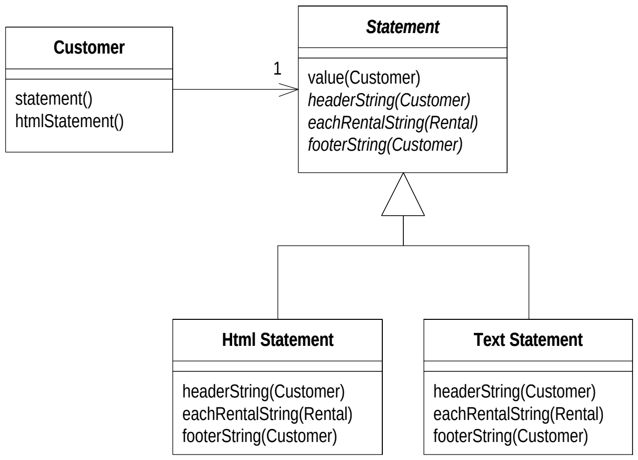
❑ **Do the same with htmlStatement**

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

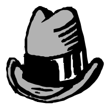
The Importance of Tests

- ❑ **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**

➧ ftp://www.armaties.com/D/home/armaties/ftp/TestingFramework/

➧ http://ourworld.compuserve.com/homepages/Martin_Fowler

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?

- ❑ The Rule of Three
- ❑ 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
  - ➢ immediate 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

Dont!

- ❑ 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
- ❑ Up front 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

XP ©

- ❑ **Methodology developed by Kent Beck**
- ❑ **Designed to adapt to changes**
- ❑ **Key Practices**
  - ➢ **Iterative Development**
  - ➢ **Self Testing Code**
  - ➢ **Refactoring**
  - ➢ **Pair Programming**
- ❑ **Moves away from up-front design**

http://www.armaties.com/extreme.htm

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

Self Testing Code

Build and run tests as you build production code

- ❑ For each piece of new function
  - ➢ Write the test
  - ➢ Write the production code
  - ➢ Run your test suite
  - ➢ If it works you’re done
- ❑ Developers
  - ➢ Do this with every small bit of function you add
- ❑ QA or Test Group
  - ➢ Do this with each increment

The JUnit Framework

- ❑ Simple, but effective framework for collecting and running unit tests in Java
- ❑ Written by Erich Gamma and Kent Beck
  - ➢ based on Kent's framework for Smalltalk
- ❑ Easily define tests
- ❑ Easily group tests into suites
- ❑ Easily run suites and monitor results

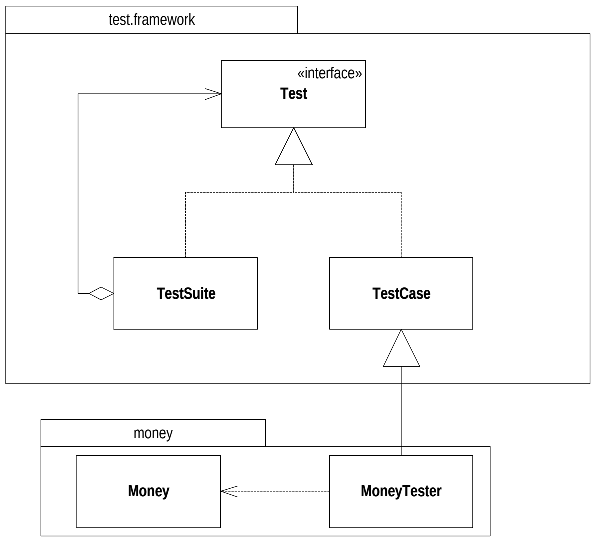
ftp://www.armaties.com/D/home/armaties/ftp/TestingFramework/JUnit/

An Example Coding Session

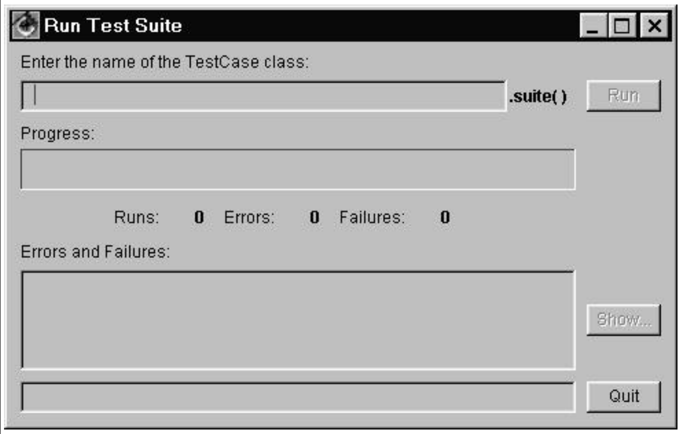
- ❑ Build a Money class
  - ➢ combines amount and currency
  - ➢ provides arithmetic operations
  - ➢ use of *Quantity* pattern
- ❑ Build a MoneyTester class

Fowler, Martin. *Analysis Patterns: Reusable Object Models*, Addison Wesley 1997

Fitting into the Framework



The Junit GUI



Enter the name of the TestCase class:

Progress:

Runs: 0 Errors: 0 Failures: 0

Errors and Failures:

- There is also a text console interface
- Invoke with
  - `java test.textui.TestRunner MoneyTester`

Creating MoneyTester

```
import test.framework.*;

public class MoneyTester extends TestCase{

  public MoneyTester(String name) {
    super(name);
  }
  public static Test suite() {
    return new TestSuite(MoneyTester.class);
  }
}
```

The First Test

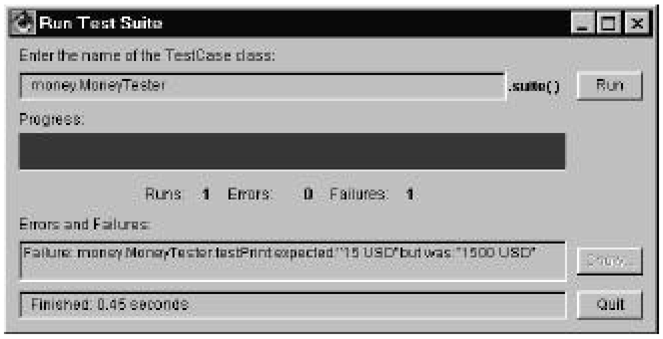
MoneyTester

```
public void testPrint() {
  Money d15 = new Money(15, "USD");
  assertEquals("15 USD", d15.toString());
}
```

```
public class Money {
  private long _amountInPennies;
  private String _currencyCode;

  public Money(double amount, String currencyCode) {
    _amountInPennies = Math.round (amount * 100);
    _currencyCode = currencyCode;
  }
  public String toString() {
    return ("" + _amountInPennies + " " + _currencyCode);
  }
}
```

First Test Result


Run Test Suite

Enter the name of the TestCase class:

money.MoneyTester .suite()

Progress:

Runs: 1 Errors: 0 Failures: 1

Errors and Failures:

```
Failure: money.MoneyTester.testPrint expected:"15 USD" but was:"1500 USD"
```

Finished: 0.45 seconds

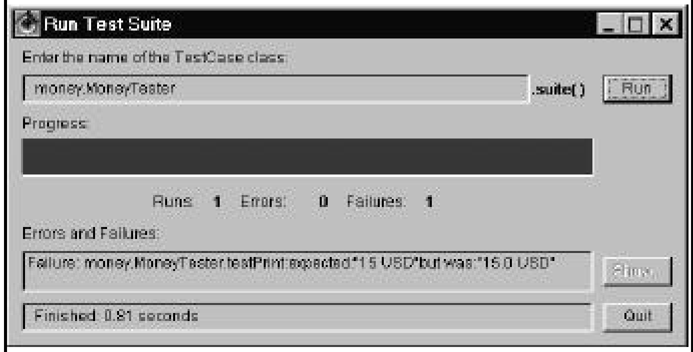
Fixing the First Test

```
public void testPrint() {
  Money d15 = new Money(15, "USD");
  assertEquals("15 USD", d15.toString());
}
```

```
public class Money {
  private long _amountInPennies;
  private String _currencyCode;

  public Money(double amount, String currencyCode) {
    _amountInPennies = Math.round (amount * 100);
    _currencyCode = currencyCode;
  }
  public String toString() {
    return ("" + getAmount() + " " + _currencyCode);
  }
  private double getAmount() {
    return (double) _amountInPennies / 100;
  }
}
```

After the fix



money.MoneyTester

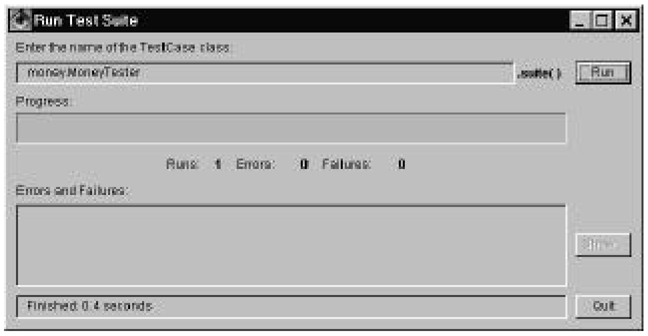
Runs: 1 Errors: 0 Failures: 1

```
Failure: money.MoneyTester.testPrint:expected:"15 USD"but was:"15.0 USD"
```

Finished: 0.81 seconds

Changing the Test

```
public void testPrint() {
  Money d15 = new Money(15, "USD");
  assertEquals("15.0 USD", d15.toString());
}
```



money.MoneyTester

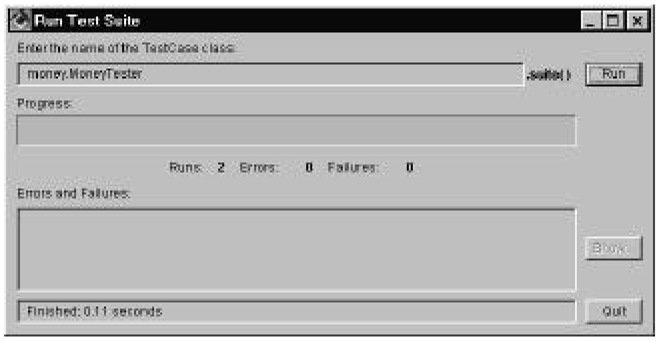
Runs: 1 Errors: 0 Failures: 0

Finished: 0.4 seconds

- ❑ **Don’t consider fancy formatting yet**

Checking Rounding

```
public void testRound() {
  Money dRounded = new Money (1.2350, "USD");
  assertEquals ("1.24 USD", dRounded.toString());
}
```



money.MoneyTester

Runs: 2 Errors: 0 Failures: 0

Finished: 0.11 seconds

Adding Addition

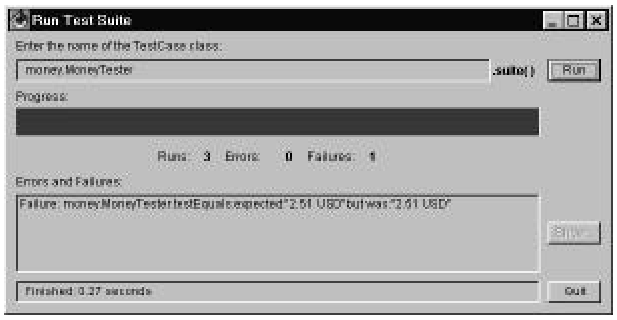
- ❑ **Add two monies together in the same currency**

```
public void testAddition() {
  Money d15 = new Money (15, "USD");
  Money d2_51 = new Money (2.51, "USD");
  assertEquals (new Money (17.51, "USD"),
                d15.plus(d2_51));
}
```

Need to add equals

- ❑ **Money is a value, so needs a special equals (and hash)**

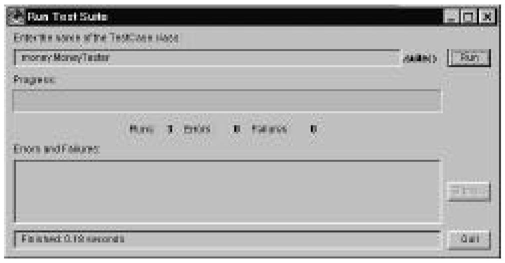
```
public void testEquals() {
  Money d2_51a = new Money (2.51, "USD");
  Money d2_51b = new Money (2.51, "USD");
  assertEquals (d2_51a, d2_51b);
}
```



```
Runs: 3   Errors: 0   Failures: 1
Failure: money.MoneyTester.testEquals:expected:"2.51 USD" but was:"2.51 USD"
Finished: 0.27 seconds
```

The Equals Method

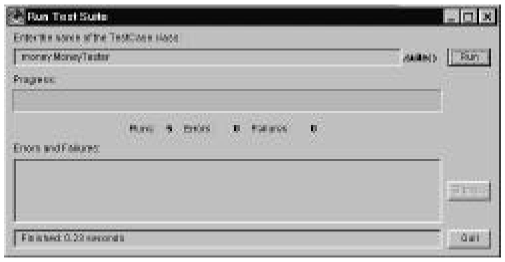
```
Money
public boolean equals (Object arg) {
  if (! (arg instanceof Money)) return false;
  Money moneyArg = (Money) arg;
  return (_amountInPennies == moneyArg._amountInPennies &&
       _currencyCode.equals(moneyArg._currencyCode));
}
```



```
Runs: 3   Errors: 0   Failures: 0
Finished: 0.19 seconds
```

Additional Tests

```
public void testCloseNumbersNotEqual() {
  Money d2_51a = new Money (2.515, "USD");
  Money d2_51b = new Money (2.5149, "USD");
  assert(! d2_51a.equals(d2_51b));
}
public void testDifferentCurrencyNotEqual() {
  Money d2_51a = new Money (2.51, "USD");
  Money d2_51b = new Money (2.51, "DEM");
  assert(! d2_51a.equals(d2_51b));
}
```

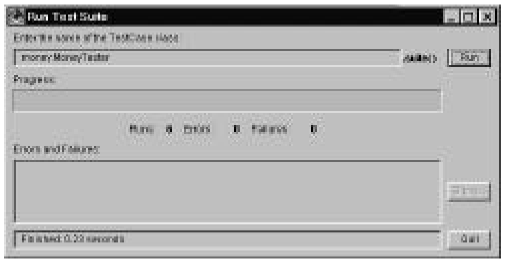


```
money.MoneyTester
Runs: 5  Errors: 0  Failures: 0
Finished: 0.22 seconds
```

Testing HashCode

MoneyTester
```
public void testHash() {
  Money d2_51a = new Money (2.51, "USD");
  Money d2_51b = new Money (2.51, "USD");
  assertEquals (d2_51a.hashCode(), d2_51b.hashCode());
}
```

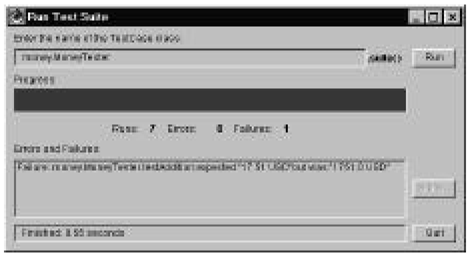
Money
```
public int hashCode() {
  return _currencyCode.hashCode() ^
        (int) _amountInPennies;
}
```



```
money.MoneyTester
Runs: 6  Errors: 0  Failures: 0
Finished: 0.22 seconds
```

The addition method

```
public Money plus (Money arg) {
  return new Money (
    _amountInPennies + arg._amountInPennies,
    _currencyCode);
}
```

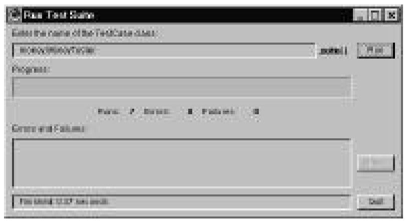


```
money.MoneyTester
Runs: 7  Errors: 0  Failures: 1
Failure: money.MoneyTester.testAddition expected:"17.51 USD" but was:"1751.0 USD"
Finished: 8.55 seconds
```

Addition with a marked constructor

```
public Money plus (Money arg) {
  return new Money (
    _amountInPennies + arg._amountInPennies,
    _currencyCode,
    false);
}

private Money (long amountInPennies, String currencyCode,
              boolean privacyMarker)
{
  _amountInPennies = amountInPennies;
  _currencyCode = currencyCode;
}
```



```
money.MoneyTester
Runs: 7  Errors: 0  Failures: 0
Finished: [illegible] seconds
```

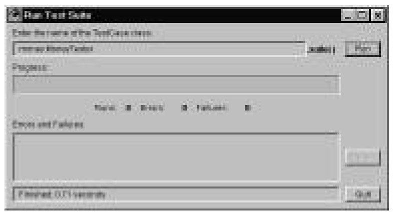
Adding different currencies

- ❑ **For this application, we will treat this as an error**
  - ➢ **An alternative is the MoneyBag pattern**

```
public void testAdditionOfDifferentCurrencies() {
  Money d15 = new Money (15, "USD");
  Money m2_51 = new Money (2.51, "DEM");
  try {
    d15.plus(m2_51);
    assert (false);
  } catch (IllegalArgumentException e) {}
}
```

The new plus method

```
public Money plus (Money arg) {
  if (! _currencyCode.equals(arg._currencyCode))
    throw new IllegalArgumentException
        ("Cannot add different currencies");
  return new Money (
    _amountInPennies + arg._amountInPennies,
    _currencyCode, false);
}
```



Finished: 0.71 seconds

Duplication of test setup code

```
public void testAdditionOfDifferentCurrencies() {
  Money d15 = new Money (15, "USD");
  Money m2_51 = new Money (2.51, "DEM");
  try {
    d15.plus(m2_51);
    assert (false);
  } catch (IllegalArgumentException e) {}
}
public void testAddition() {
  Money d15 = new Money (15, "USD");
  Money d2_51 = new Money (2.51, "USD");
  assertEquals (new Money (17.51, "USD"), d15.plus(d2_51));
}
public void testDifferentCurrencyNotEqual() {
  Money d2_51a = new Money (2.51, "USD");
  Money d2_51b = new Money (2.51, "DEM");
  assert(! d2_51a.equals(d2_51b));
}
```

Create a test fixture

```
public class MoneyTester extends TestCase{
  private Money d15;
  private Money d2_51;
  private Money m2_51;


public void setUp() {
  d15 = new Money (15, "USD");
  d2_51 = new Money (2.51, "USD");
  m2_51 = new Money (2.51, "DEM");
}

public void testDifferentCurrencyNotEqual() {
  assert(! d2_51.equals(m2_51));
}
```

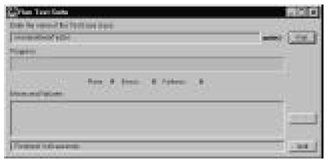
Adding Subtraction

MoneyTester

```
public void testSubtraction() {
  assertEquals (new Money (12.49, "USD"),
d15.minus(d2_51));
}
```

Money

```
public Money minus (Money arg) {
  if (! _currencyCode.equals(arg._currencyCode))
    throw new IllegalArgumentException ("Cannot add
different currencies");
  return new Money (_amountInPennies -
arg._amountInPennies, _currencyCode, false);
}
```



Duplicate Code

```
public Money minus (Money arg) {
  if (! _currencyCode.equals(arg._currencyCode))
    throw new IllegalArgumentException ("Cannot add
different currencies");
  return new Money (_amountInPennies -
arg._amountInPennies, _currencyCode, false);
}
public Money plus (Money arg) {
  if (! _currencyCode.equals(arg._currencyCode))
    throw new IllegalArgumentException ("Cannot add
different currencies");
  return new Money (_amountInPennies +
arg._amountInPennies, _currencyCode, false);
}
```

- **Kill such snakes immediately**

Extract Methods

```
public Money minus (Money arg) {
  assertSameCurrency(arg);
  return new Money (_amountInPennies -
arg._amountInPennies, _currencyCode, false);
}

public Money plus (Money arg) {
  assertSameCurrency(arg);
  return new Money (_amountInPennies +
arg._amountInPennies, _currencyCode, false);
}


public void assertSameCurrency (Money arg) {
  if (! _currencyCode.equals(arg._currencyCode))
    throw new IllegalArgumentException ("Currencies must
be the same");
}
```

- ❑ **Make it work, make it right**

Next Step: Local Printing

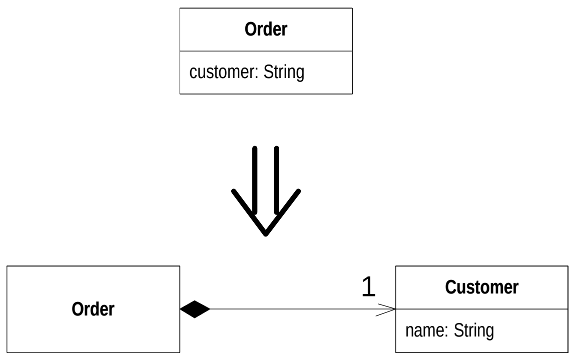
- ❑ **Leave other arithmetic and sort operations to the reader**
- ❑ **Provide a localString method that formats the currency in the native locale of the currency**

```
public void xtestLocalPrinting() {
  //assertEquals("$15.00", d15.localString());
  //assertEquals("2,51 DM", m2_51.localString());
}
```

- ❑ **We need a currency class**
  - ➢ **refactor the money class to use a currency:**
- ❑ **Define the test, but don’t run it yet**

Replace Data Value with Object

You have a data item that needs additional data or behavior
Turn the data item into an object



Replace Data Value with Object

Money

```
public Money(double amount, String currencyCode) {
  _amountInPennies = Math.round (amount * 100);
  _currency = new Currency(currencyCode);
}
public boolean equals (Object arg) {
  if (! (arg instanceof Money)) return false;
  Money moneyArg = (Money) arg;
  return (_amountInPennies == moneyArg._amountInPennies &&

_currency.getCode().equals(moneyArg._currency.getCode()));
}
private long _amountInPennies;
private Currency _currency;
```

Currency

```
public Currency(String code) {
  _code = code;
}
public String getCode() {
  return _code;
}
private String _code;
```

Code in the wrong place

Money

```
public boolean equals (Object arg) {
  if (! (arg instanceof Money)) return false;
  Money moneyArg = (Money) arg;
  return (_amountInPennies == moneyArg._amountInPennies &&

_currency.getCode().equals(moneyArg._currency.getCode()));
}
```

Move Method

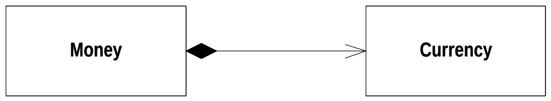
Money

```
public boolean equals (Object arg) {
  if (! (arg instanceof Money)) return false;
  Money moneyArg = (Money) arg;
  return (_amountInPennies == moneyArg._amountInPennies &&
      _currency.equals(moneyArg._currency));
}
```

Currency

```
public boolean equals(Object arg) {
  if (! (arg instanceof Currency)) return false;
  Currency currencyArg = (Currency) arg;
  return (_code.equals(currencyArg._code));
}
```

Currency is a value



Money

```
public Money(double amount, String currencyCode) {
        _amountInPennies = Math.round (amount * 100);
        _currency = new Currency(currencyCode);
}
```

Replace Constructor with Factory Method

You want to do more than simple construction when you create an object

Replace the constructor with a factory method

```
Employee (int type) {
    _type = type;
}
```

⇓

```
static Employee create(int type) {
    return new Employee(type);
}
```

Replacing the Constructor

```
class Currency...
public static Currency create (String code) {
  return new Currency (code);
}
private Currency(String code) {
  _code = code;
}

class Money...
public Money(double amount, String currencyCode) {
  _amountInPennies = Math.round (amount * 100);
  _currency = Currency.create(currencyCode);
}
```

Replace value object with reference object

```
class Currency...
private String _code;
private static Dictionary _instances = new Hashtable();

public static void loadInstances() {
    _instances.put("USD", new Currency("USD"));
    _instances.put("GBP", new Currency("GBP"));
    _instances.put("DEM", new Currency("DEM"));
}

public static Currency create (String code) {
    Currency result = (Currency) _instances.get(code);
    if (result == null)
      throw new IllegalArgumentException
        ("There is no currency with code: " + code);
    return result;
}
class MoneyTester…
public void setUp() {
  Currency.loadInstances();
  d15 = new Money (15, "USD");
  d2_51 = new Money (2.51, "USD");
  m2_51 = new Money (2.51, "DEM");
}
```

Add the locale to currency

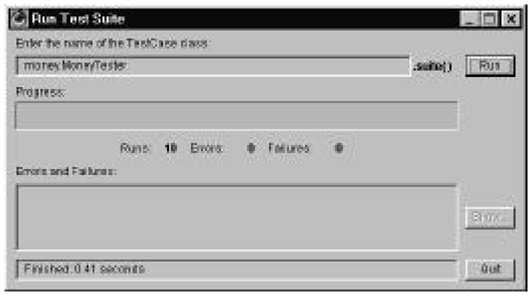
```
class Currency...
private Currency(String code, Locale locale) {
  _code = code;
  _locale = locale;
}
public static void loadInstances() {
  _instances.put("USD", new Currency("USD", Locale.US));
  _instances.put("GBP", new Currency("GBP", Locale.UK));
  _instances.put("DEM", new Currency("DEM",
Locale.GERMANY));
}

private Locale _locale;
```

Add methods for printing

```
class Money...
public String localString() {
  return _currency.getFormat().format(getAmount());
}
class Currency...
public NumberFormat getFormat() {
  return NumberFormat.getCurrencyInstance(_locale);
}
```

- ❑ **Enable the test**



money.MoneyTester

Runs: 10 Errors: 0 Failures: 0

Finished: 0.41 seconds

The Rhythm of Development

- ❑ Define a test
- ❑ Refactor to make it easy to add the function
- ❑ Add functionality
- ❑ Enable the test
- ❑ Refactor to remove any bad smells
- ❑ Integrate

Daily Build

- ❑ Build system every day
  - ➢ compile, link, and unit tests at 100%
  - ➢ Anyone who breaks build must fix it immediately
- ❑ Developers should check in daily
  - ➢ If more than 2 days - raise flag
  - ➢ break down coding effort for intermediate build
  - ➢ developers do personal build before checking in
- ❑ Assign a build token

Code Smells

Martin Fowler

fowler@acm.org

http://ourworld.compuserve.com/homepages/Martin_Fowler

Bad Smells in Code

"If it stinks, change it."
— Grandma Beck, discussing child raising philosophy

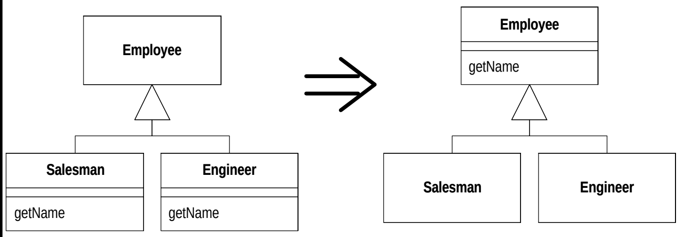
- ❑ How do we know when to refactor
- ❑ No hard and fast rules
- ❑ Bad Smells are things to look for
  - ➢ suggest certain refactorings

Duplicated Code

- ❑ **Same expression in two methods of the same class**
  - ➢ **Use *Extract Method***
- ❑ **Same expression in sibling subclasses**
  - ➢ ***Extract Method* and *Pull Up Method***
- ❑ **Similar code in sibling subclasses**
  - ➢ **Use *Form Template Method***
- ❑ **Same code in unrelated classes**
  - ➢ **Decide where it should really be and use *Move Method* to get it there.**
  - ➢ **May be a signal for *Extract Class***

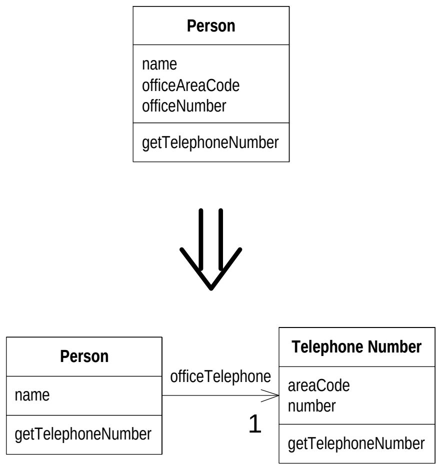
Pull Up Method

You have methods with identical results on subclasses
Move them to the superclass



Extract Class

You have a class that is doing the 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.



Long Method

- ❑ **Use *Extract Method* on logical chunks**
  - ➢ **For conditions: *Decompose Conditional***
- ❑ **Lots of temps make extraction difficult**
  - ➢ **Use *Replace Temp with Query***
  - ➢ **For parameters use *Introduce Parameter Object* and *Preserve Whole Object***
  - ➢ **As a last resort use *Replace Method with Method Object***

Decompose Conditional

You have a complicated conditional (if-then-else) statement
Extract methods from the condition, then part, and else parts.

```
if (date.before (SUMMER_START) || date.after(SUMMER_END))
    charge = quantity * _winterRate + _winterServiceCharge;
else charge = quantity * _summerRate;
```

⇓

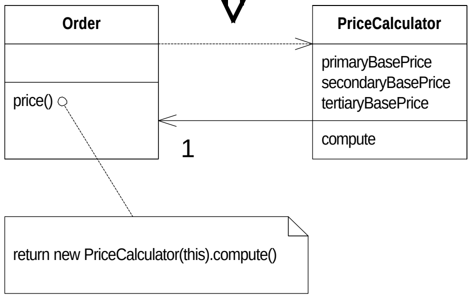
```
if (notSummer(date))
    charge = winterCharge(quantity);
else charge = summerCharge (quantity);
```

Replace Method with Method Object

You have a long method that uses local variables in such a way that you cannot apply Extract Method
Turn the method into its own object.

```
class Order...
  double price() {
      double primaryBasePrice;
      double secondaryBasePrice;
      double tertiaryBasePrice;
      // long computation;
      ...
  }
```

⇓



Preserve Whole Object

You are getting several values from an object and passing these values as parameters in a method call
Send the whole object instead

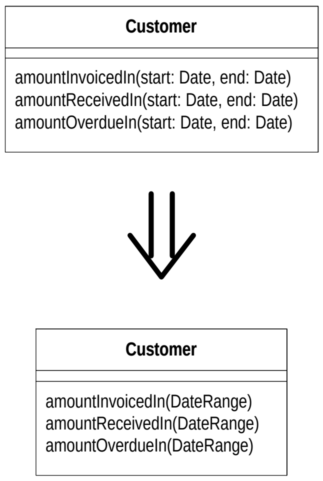
```
        int low = daysTempRange().getLow();
        int high = daysTempRange().getHigh();
        withinPlan = plan.withinRange(low, high);
```

$\Downarrow$

```
        withinPlan = plan.withinRange(daysTempRange());
```

Introduce Parameter Object

You have a group of parameters that naturally go together
Replace them with an object

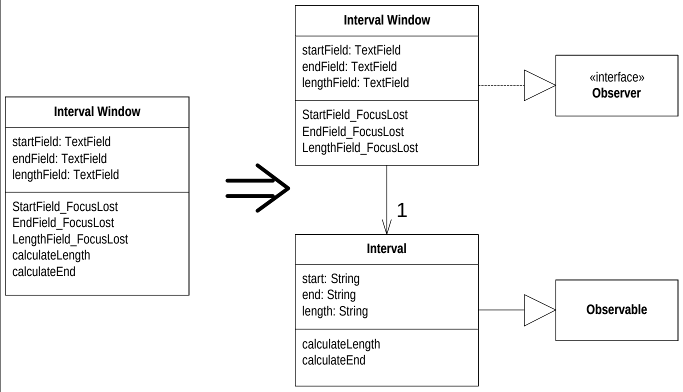

Customer
amountInvoicedIn(start: Date, end: Date)
amountReceivedIn(start: Date, end: Date)
amountOverdueIn(start: Date, end: Date)
Customer
amountInvoicedIn(DateRange)
amountReceivedIn(DateRange)
amountOverdueIn(DateRange)

Large Class

- ❑ Find a bunch of data and methods that go together
  - ➢ Use *Extract Class* or *Extract Subclass*
- ❑ Examine how clients use the class
  - ➢ separate different kinds of uses with *Extract Interface*
- ❑ Complex GUI Classes
  - ➢ Use *Extract Class* to create domain objects.
  - ➢ Use *Duplicate Observed Data* where data needs to be in both places

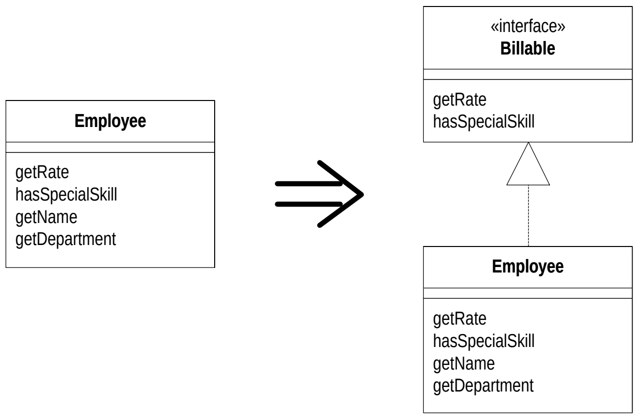
Duplicate Observed Data

You have domain data available only in a gui control and domain methods need access.
Copy the data to a domain object. Set up an observer to synchronize the two pieces of data.



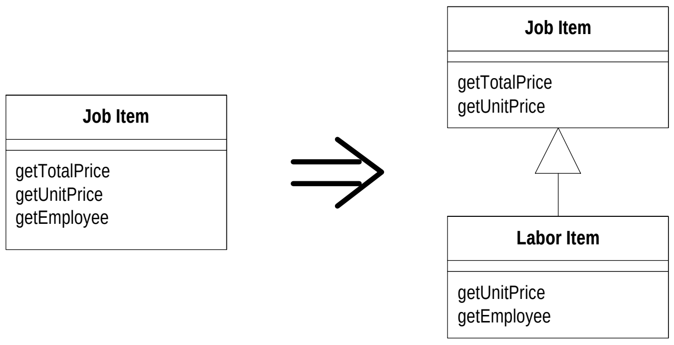
Extract Interface

Several clients use the same subset of a class's interface or two classes have part of their interfaces in common
Extract that subset into an interface



Extract Subclass

A class has features that are only used by some instances
Create a subclass for that subset of features



Long Parameter Lists

- ❑ **Parameters that seem to go together**
  - ➢ ***Preserve Whole Object***
  - ➢ ***Introduce Parameter Object***
- ❑ **The invoked method can find one parameter itself**
  - ➢ **Use *Replace Parameter With Method***

Replace Parameter With Method

An object invokes a method, then passes the result as a parameter for a method. The receiver could also invoke this method.
Remove the parameter and let the receiver invoke the method

```
int basePrice = _quantity * _itemPrice;
discountLevel = getDiscountLevel();
double finalPrice = discountedPrice (basePrice, discountLevel);
```

$\Downarrow$

```
int basePrice = _quantity * _itemPrice;
double finalPrice = discountedPrice (basePrice);
```

Divergent Change

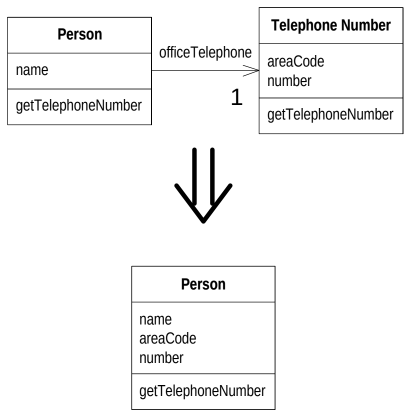
- ❑ Change one class in different ways for different reasons
- ❑ Usually only apparent during evolution of a system
  - ➢ Use ***Extract Class*** to factor out each style of change

Shotgun Surgery

- ❑ A Common kind of change affects several classes
- ❑ Need to bring the changes together to make change easier
  - ➢ ***Move Method*** and ***Move Field*** to bring common elements together
  - ➢ ***Inline Class*** to remove unnecessary separations

Inline Class

A class isn't doing very much
Move all its features into another class and delete it.



Feature Envy

- **A method uses more features from another class than it does its own class**
  - **Use *Move Method* to move it to where it wants to be**
  - **If only part of a method is jealous use *Extract Method* and *Move Method***
- **Many patterns deliberately break this rule**
  - **To avoid smells of Divergent Change or Shotgun Surgery**

Data Clumps

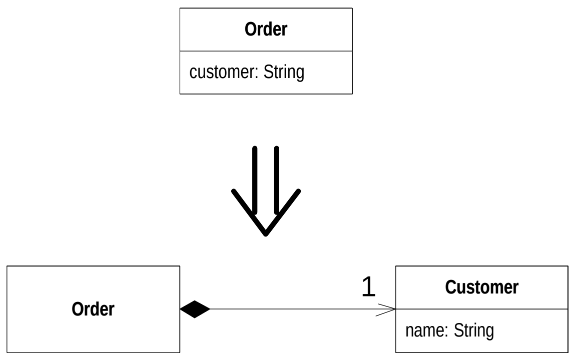
- ❑ Data Items that tend to hang around together
  - ➢ Groups of fields in several classes
  - ➢ Groups of parameters in several method calls
  - ➢ eg: startDate and endDate
- ❑ Start with classes
  - ➢ Use *Extract Class* to group fields into an object
- ❑ Continue with method calls
  - ➢ *Preserve Whole Object*
  - ➢ *Introduce Parameter Object*
- ❑ A test: if you delete on data item, do the others make sense?
- ❑ Now look for methods on other classes that have Feature Envy for the new classes

Primitive Obsession

- ❑ Objects blur the line between primitive data types and records
- ❑ Objects are almost always more useful
  - ➢ *Replace Data Value with Object*
  - ➢ *Replace Type Code with Class*
  - ➢ *Replace Type Code with Subclasses*
  - ➢ *Replace Type Code with State/Strategy*
  - ➢ *Replace Array with Object*
- ❑ Look for Data Clumps

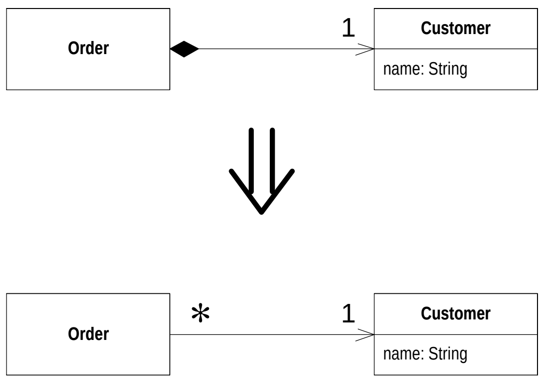
Replace Data Value with Object

You have a data item that needs additional data or behavior
Turn the data item into an object



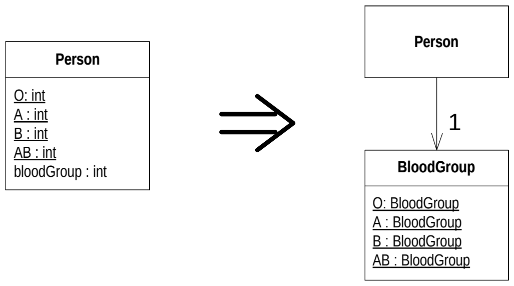
Change Value to Reference

You have a class with many equal instances that you want to replace with a single object
Turn the object into a reference object



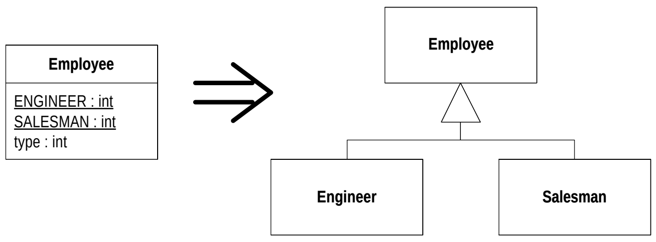
Replace Type Code with Class

A class has a numeric type code that does not affect its behavior
Replace the number with a new class



Replace Type Code with Subclasses

You have an immutable type code which affects the behavior of a class
Replace the type code with subclasses



Replace Array with Object

You have an array where certain elements mean different things
Replace the array with an object, with a field for each element

```
String[] row = new String[3];
row [0] = "Liverpool";
row [1] = "15";
```

⇓

```
Performance row = new Performance();
row.setName("Liverpool");
row.setWins("15");
```

Switch Statements

- ❑ **Usually leads to duplicated conditionals**
  - ➢ **particularly when used with a type code**
- ❑ **Set up structure and use polymorphism**
  - ➢ ***Extract Method*** **to remove the conditional**
  - ➢ ***Move Method*** **to put it in the right place**
  - ➢ ***Replace Type Code with Subclasses***
  - ➢ ***Replace Type Code with State/Strategy***
  - ➢ ***Replace Conditional with Polymorphism***
- ❑ **Conditional behavior on a parameter**
  - ➢ **Consider *Replace Parameter with Explicit Methods***
  - ➢ **But try to remove the parameter**
- ❑ **For null tests**
  - ➢ ***Introduce Null Object***

Replace Parameter with Explicit Methods

You have a method with a runs different code depending on the values of an enumerated parameter
Create a separate method for each value of the parameter

```
void setValue (String name, int value) {
    if (name.equals("height"))
        _height = value;
    if (name.equals("width"))
        _width = value;
    Assert.shouldNeverReachHere();
}
```

⇓

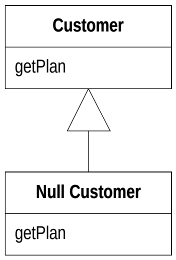
```
void setHeight(int arg) {
    _height = arg;
}
void setWidth (int arg) {
    _width = arg;
}
```

Introduce Null Object

You have a method with a runs different code depending on the values of an enumerated parameter
Create a separate method for each value of the parameter

```
if (customer == null) plan = BillingPlan.basic();
else plan = customer.getPlan();
```

⇓



Parallel Inheritance Hierarcies

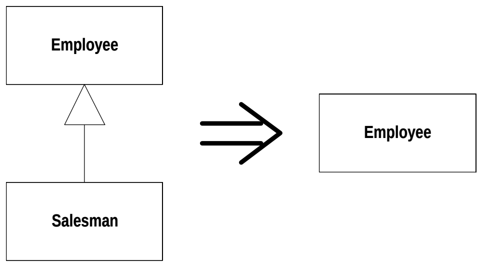
- ❑ Coupling between separate hierarcies
- ❑ Often signalled by common prefixes or suffixes
- ❑ Make one hierarchy refer to the other
- ❑ Move features to the latter
- ❑ Remove the former

Lazy Class

- ❑ A class that does not do enough
- ❑ Remove it
  - ➢ Collapse Hierarchy
  - ➢ Inline Class

Collapse Hierarchy

A superclass and subclass are not very different
Merge them together



Speculative Generality

- **Unused features that are there because you are “sure” you’ll need them**
- **Unused features make the program hard to understand, and are usually wrong**
- **You can always add them later**
- **So remove them**
  - **Lazy Abstract Classes: *Collapse Hierarchy***
  - **Unnecessary delegation: *Inline Class***
  - **Unused parameters: *Remove Parameter***
  - **Odd abstract method names: *Rename Method***

Temporary Field

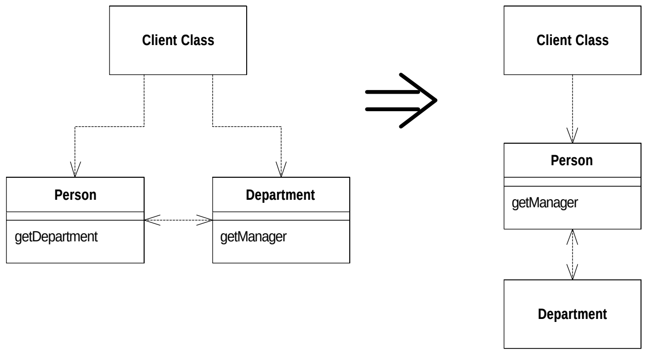
- A field that's only used for a short part of a class's life
- If there's more than one: separate them into their own class
  - *Extract Class*
  - Avoid conditionals with *Introduce Null Object*

Message Chain

- getObject().getAnotherObject().getYetAnotherObject().getYetAnotherAnotherObject().somethingMeaningful()
- Couples host to a whole data structure
- Hide the structure
  - *Hide Delegate*
  - But may result in Middle Men
- See what the final code is doing
  - Use *Extract Method* on the code that uses it
  - Use *Move Method* to move it down the chain

Hide Delegate

A client is calling a delegate class of an object
Create methods on the server to hide the delegate

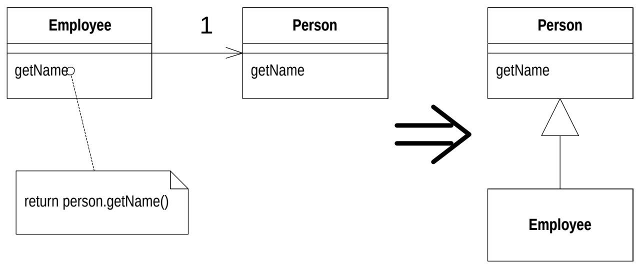


Middle Man

- ❑ **Chasing around a lot of empty delegation**
- ❑ **Talk to the real object**
  - ➢ ***Remove Middle Man***
  - ➢ **But beware of Message Chains**
  - ➢ **If several methods use the same delegation: *Inline Method***
  - ➢ ***Replace Delegation with Inheritance***

Replace Delegation with Inheritance

You're using delegation and are often writing many simple delegations for whole interface
Make the delegating class a subclass of the delegate

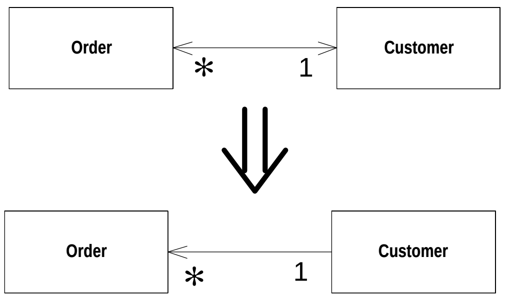


Inappropriate Intimacy

- ❑ **Classes should not know too much about each other**
- ❑ **Break up classes to reduce needed links**
  - ➢ **Use *Move Method* and *Move Field* to separate pieces**
  - ➢ ***Change Bidirectional Association to Unidirectional***
  - ➢ ***Extract Class* to combine common interests**
  - ➢ ***Hide Delegate* to let another class mediate.**
- ❑ **Inheritance often increases coupling**
  - ➢ ***Replace Inheritance with Delegation***

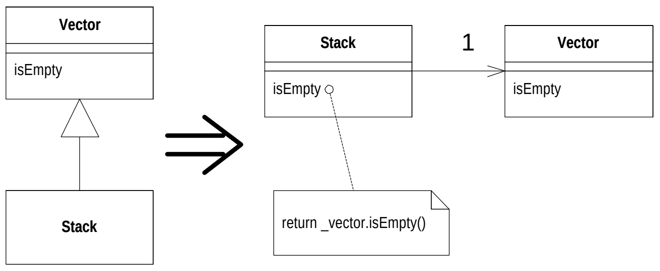
Change Bidirectional Association to Unidirectional

You have a two way association but one class no longer needs features from the other.
Drop the unneeded end of the association



Replace Inheritance with Delegation

A subclasses only uses part of a superclasses interface, or does not want to inherit its data.
Create a field for the superclass, adjust methods to delegate to the superclass, remove the subclassing

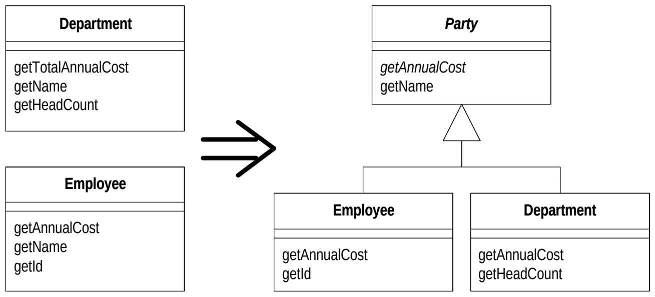


Alternative classes with different interfaces

- ❑ Try to ensure classes use different implementations with the same interface
  - ➢ *Rename Method* to get names the same
  - ➢ *Move Method* if one class does not do enough
  - ➢ *Extract Superclass* to factor out commonalities
  - ➢ *Extract Interface* if you can't superclass

Extract Superclass

You have two classes with similar features
Create a superclass and move the common features to the superclass



Incomplete Library Class

- ❑ **Cannot change library classes**
- ❑ **So usual tactics don't work**
  - ➢ ***Introduce Foreign Method***
  - ➢ ***Introduce Local Extension***

Introduce Foreign Method

A server class you are using needs an additional method, but you can't modify the class.
Create a method in the client class with an instance of the server class as its first argument

```
Date newStart = new Date (previousEnd.getYear(),
                  previousEnd.getMonth(), previousEnd.getDate() + 1);
```

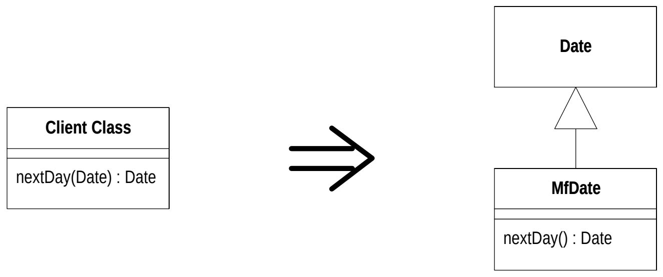
$\Downarrow$

```
Date newStart = nextDay(previousEnd);

private static Date nextDay(Date arg) {
    return new Date (arg.getYear(),arg.getMonth(), arg.getDate() + 1);
}
```

Introduce Local Extension

A server class you are using needs several additional methods, but you can't modify the class.
Create a new class which contains these extra methods. Make this extension class a subclass or a wrapper of the original



Data Class

- ❑ **A class that is just getters and setters**
- ❑ **May have public data**
  - ➢ ***Encapsulate Field***
  - ➢ ***Encapsulate Collection***
  - ➢ ***Remove Setting Method***
- ❑ **Look for methods that use the accessors**
  - ➢ **Use *Extract Method* and *Move Method* to move behavior into the data class**
  - ➢ **Look to *Hide Method* on the accessors**

Encapsulate Field

There is a public field
Make it private and provide accessors

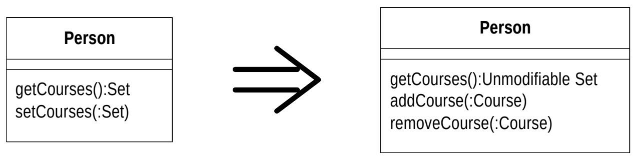
```
public String _name
```

⇓

```
private String _name;
public String getName() {return _name;}
public void setName(String arg) {_name = arg;}
```

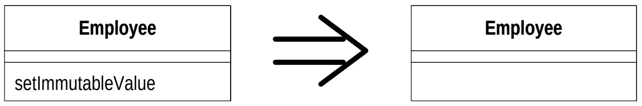
Encapsulate Collection

A method returns a collection
Make it return a read only view and provide add/remove methods



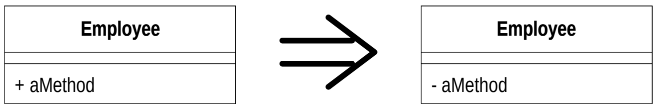
Remove Setting Method

A field should be set at creation time and never altered
Remove any Setting Method for that field



Hide Method

A Method is not used by any other class
Make the Method private



Refused Bequest

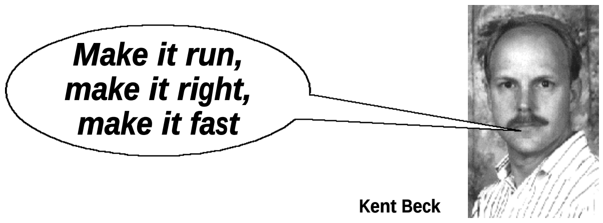
- ❑ Only using some of the features of the parent
- ❑ A sign of an incorrect hierarchy
- ❑ Create a new sibling class
  - ➢ *Push Down Method*
  - ➢ *Push Down Field*
- ❑ Doesn't want to support parent interface
  - ➢ *Replace Inheritance with Delegation*

Comments

- ❑ Not a bad smell: but is a deodorant
- ❑ Look for the smell that the comment is trying to mask
- ❑ Remove the smell, see if you still need the comment

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



Kent Beck