



En la clase pasada...

Alejandra Garrido – Objetos 2  
Facultad de Informática - UNLP

# Nos ponemos ágiles

- <http://agilemanifesto.org/>
- Dos prácticas ágiles esenciales:
  - Refactoring
  - Test first

# [ Create CLEAN Code ]

## ■ CLEAN:

- Cohesive,
- Loosely coupled,
- Encapsulated,
- Assertive,
- Non-redundant.
  - Esta semana en LinkedIn group “Software Refactoring”: “Technical Excellence” by David Bernstein, autor de “Beyond Legacy Code”

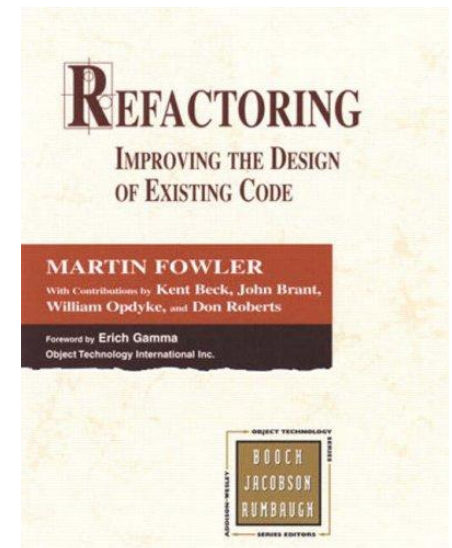


# Refactoring de código

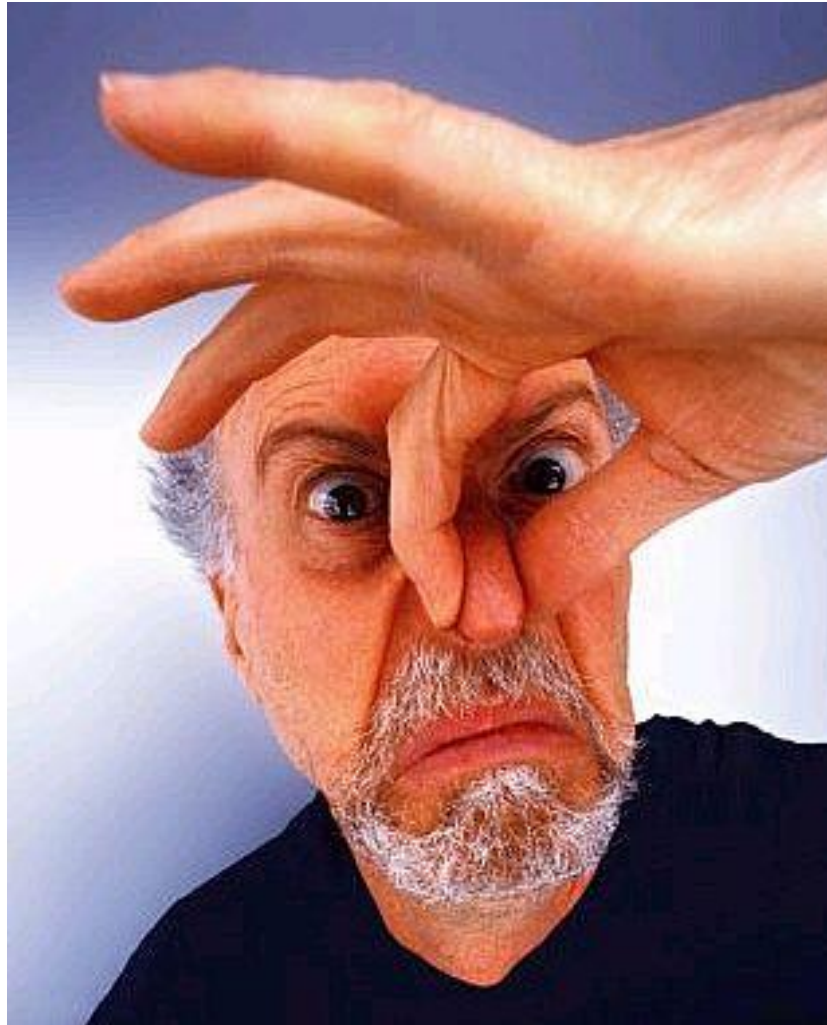
¿Cómo llegamos a  
tener CLEAN code?

# [Refactoring]

- Es el proceso a través del cual se cambia un sistema de software mejorando la organización, legibilidad, adaptabilidad y mantenibilidad del código luego que ha sido escrito....
  - que **NO altera** el comportamiento externo del sistema,
  - que *mejora* su estructura interna



# [ BAD SMELLS!! (in code) ]



# [ Algunos bad smells ]

- Duplicate Code
- Large Class
- Long Method
- Data Class
- Feature Envy
- Long Parameter List
- Switch Statements

# [ Code smell: Código duplicado ]

- El mismo código, o código muy similar, aparece en muchos lugares.
- Problemas:
  - Un bug fix en un clone no es fácilmente propagado a los demás clones
  - Hace el código más largo de lo que necesita ser
- Ejemplo (VisualWorks):  
BitEditor>>blackenPointAt: y >>whitenPointAt:

# [Code smell: Método largo]

- Un método tiene muchas líneas de código
- ¿Cuánto es muchas LOCs?
  - Más de 20? 30?
  - También depende del lenguaje
- Problemas:
  - Cuanto más largo es un método, más difícil es entenderlo, cambiarlo y reusarlo
- Ejemplo (Pharo):  
Date class>>readFrom:

# [ Code smell: Envidia de atributo ]

- Un método en una clase usa principalmente los datos y métodos de otra clase para realizar su trabajo (se muestra “envidiosa” de las capacidades de otra clase)
- Problema:
  - Indica un problema de diseño
  - Idealmente se prefiere que los datos y las acciones sobre los datos vivan en la misma clase
  - “Feature Envy” indica que el método fue ubicado en la clase incorrecta
- Ejemplo:
  - ClubSquash>>deleteRankingOf:at:

# [Code smell: Clase grande]

- Una clase intenta hacer demasiado trabajo
- Tiene muchas variables de instancia
- Tiene muchos métodos
- Problema:
  - Indica un problema de diseño.
  - Algunos métodos puede pertenecer a otra clase
  - Generalmente tiene código duplicado
- Ejemplos (Pharo): BlockNode, CharacterScanner, Paragraph

# [ Code smell: Condicionales ]

- Cuando sentencias condicionales contienen lógica para diferentes tipos de objetos
- Cuando todos los objetos son instancias de la misma clase, eso indica que se necesitan crear subclases.
- Problema: la misma estructura condicional aparece en muchos lugares
- Ejemplos (Pharo):
  - `Date class>>readFrom:`
  - `AbstractNautilusUI>>dragPassengersFor:inMorph:`

# [ Code smell: Clase de datos ]

- Una clase que solo tiene variables y getters/setters para esas variables
- Actúa únicamente como contenedor de datos
- Problemas:
  - En general sucede que otras clases tienen métodos con “envidia de atributo”
  - Esto indica que esos métodos deberían estar en la “data class”
  - Suele indicar que el diseño es procedural
- Ejemplos:
  - Jugador, Partido

# [Veamos un ejemplo....]

- Calcular e imprimir el extracto (statement) de alquileres que un cliente ha hecho en un video club con los costos de cada alquiler.
- El metodo recibe los videos alquilados y por cuanto tiempo los alquilo.
- Calcula los costos dependiendo de los días alquilados y del tipo de película
- Hay 3 tipos de películas: *común*, *infantil* y *novedad*
- También se calculan los puntos ganados por alquiler, y los puntos varían dependiendo si la película es novedad o no.

# [ El resumen impreso ... ]

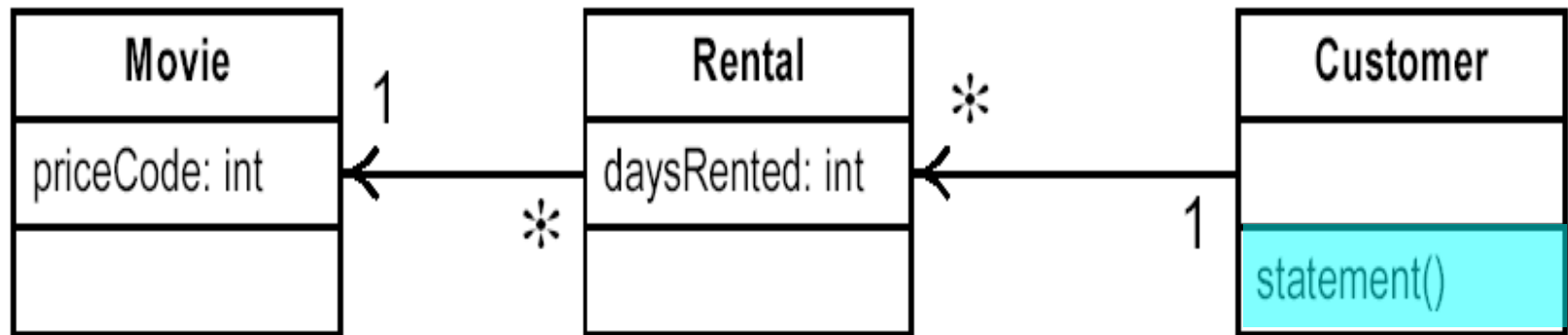
## Rental Record for Alejandra Garrido

The Revenant	46
The Martian	35
Up	20
El Padrino 3	20
Iron Man 3	25

Amount owed is \$146

You earned 6 frequent renter points

# [ El diseño ]



# [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 (1)

```
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() //próxima transparencia
```

# class Customer (2)

```
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;  
        }  
        //continua en la próxima transparencia
```

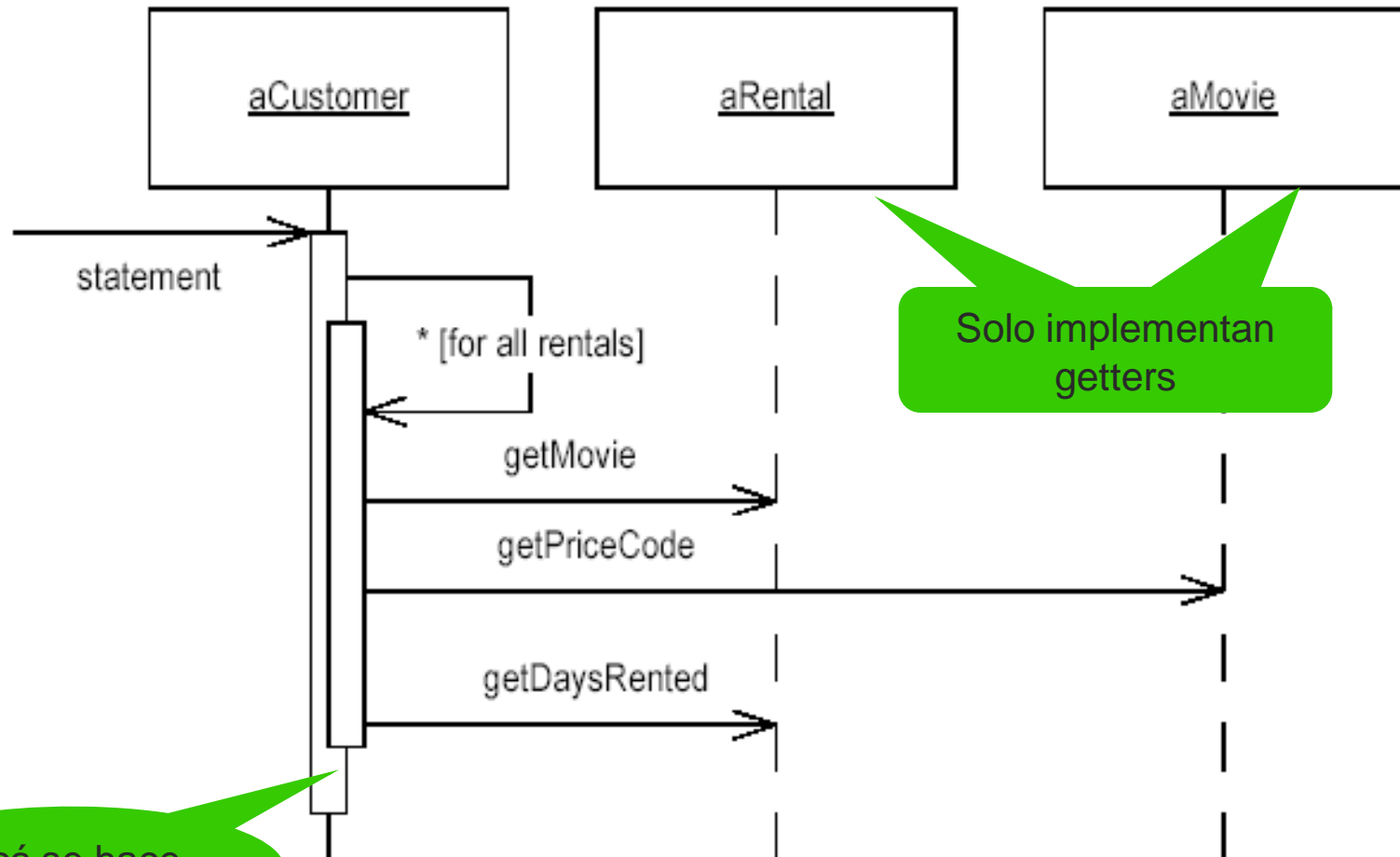


# [ class Customer (3)

```
// 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;
}
```

# [ Diagrama de Secuencia ]



Solo implementan  
getters

Acá se hace  
todo

# [Cambios pedidos ...]

- Producir una versión HTML del comprobante
- La manera de calcular el costo también cambiará
- La clasificación de película pronto va a cambiar, aunque no saben todavía como

# [ Ejemplo del video club ]

- Cuáles son los malos olores?
- Por dónde empezamos?

# [ Cálculo del costo del alquiler ]

```
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;
        }

        result += "\t" + each.getMovie().getTitle() + ": " + thisAmount + " "
            + (each.getDaysRented() > 1 ? "days" : "day") + ", ";
        totalAmount += thisAmount;
        frequentRenterPoints += 1;
    }
    result += "\n" + "Total Amount Due: " + totalAmount + "\n";
    result += "Frequent Renter Points: " + frequentRenterPoints;

    return result;
} // [snip]
```

# [ Extract Method ]

- Motivación :
  - Métodos largos
  - Métodos muy comentados
  - Incrementar reuso
  - Incrementar legibilidad

# [Extract Method]

- Mecánica:
  - Crear un nuevo método cuyo nombre explique su propósito
  - Copiar el código a extraer al nuevo método
  - Revisar las variables locales del original
  - Si alguna se usa sólo en el código extraído, mover su declaración
  - Revisar si alguna variable local es modificada por el código extraído. Si es solo una, tratar como query y asignar. Si hay más de una no se puede extraer.
  - Pasar como parámetro las variables que el método nuevo lee.
  - Compilar
  - Reemplazar código en método original por llamada
  - Compilar

# [ Cálculo del costo del alquiler ]

```
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;
        }

        result += "\t" + each.getMovie().getTitle() + ": " + thisAmount + " "
            + (each.getDaysRented() > 1 ? "days" : "day") + ", ";
        totalAmount += thisAmount;
        frequentRenterPoints++;
    }

    //print out total amount and frequent renter points
    result += "\n" + "Total Amount: " + totalAmount + "\n"
        + "Frequent Renter Points: " + frequentRenterPoints;

    return result;
} // [snip]
```

# [ Extract method: Extrayendo (1) ]

```
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;  
}
```

# [ Extract method: después ]

```
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;  
}
```

# [ A tener en cuenta... ]

- Testear siempre después de hacer un cambio
  - Sí se cometió un error es más fácil corregirlo

```
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;  
    }
```



# [ Nombres de variables

```
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;  
}
```

# [Con los nombres de variables cambiados]

```
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;  
}
```

# [Vale la pena?

- Todo buen código debería comunicar con claridad lo que hace
- Nombres de variables adecuados aumentan la claridad
- Sólo los buenos programadores escriben código legible por otras personas

# [ En la clase Customer ... ]

Todo este  
código es propio  
de `aRental`

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

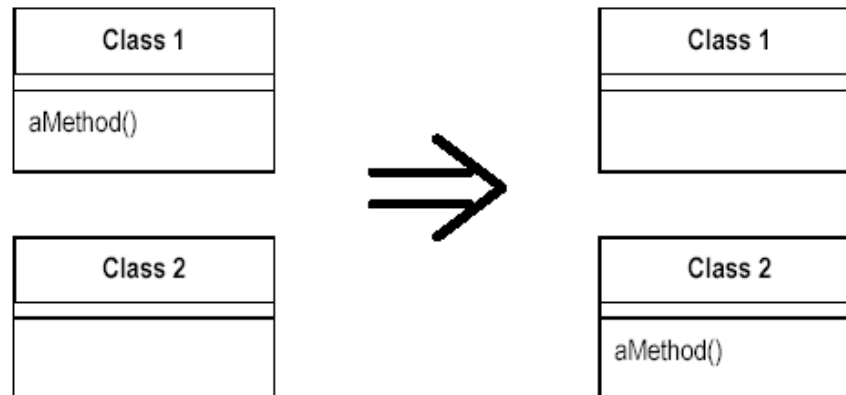
# [ Move Method ]

- Motivación:

- Un método esta usando o usará muchos servicios que están definidos en una clase diferente a la suya (Feature envy)

- Solucion:

- Mover el método a la clase donde están los servicios que usa.
- Convertir el método original en un simple delegación o eliminarlo

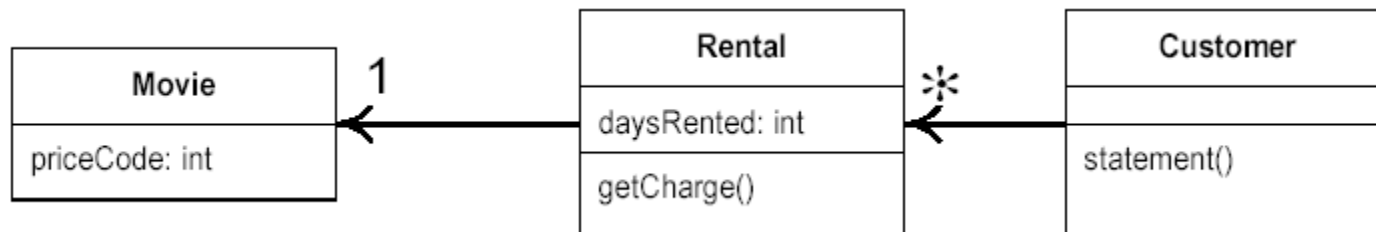


# [ Move Method: Mecánica ]

- Revisar otros atributos y métodos en la clase original (puede que también haya que moverlos)
- Chequear subclases y superclases de la clase original por si hay otras declaraciones del método (puede que no se pueda mover)
- Declarar el método en la clase destino
- Copiar y ajustar el código (ajustando las referencias desde el objeto origen al destino); chequear manejo de excepciones
- Convertir el método original en una delegación
- Compilar y testear
- Decidir si eliminar el método original → eliminar las referencias
- Compilar y testear

# Moviendo el cálculo del costo de alquiler a la clase **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;
    }
```



# [Y en Customer ...]

```
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;
    }
}
```



# Redundancia de variables temporales

```
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;
    }
}
```

# [ Replace Temp with Query ]

## ■ Motivación:

- Las temporales, al ser locales, fomentan métodos largos
- Para poder usar una expresión desde otros métodos
- Antes de un Extract Method, para evitar parámetros innecesarios

## ■ Solución:

- Extraer la expresión en un método
- Remplazar TODAS las referencias a la var. temporal por la expresión
- El nuevo método luego puede ser usado en otros métodos

# [Replace Temp With Query]

## ■ Mecánica:

- Encontrar las vars. temporales con una sola asignación (si no, Split Temporary Variable)
- Extraer el lado derecho de la asignación (tener cuidado con los efectos colaterales; si no, Separate Query From Modifier)
- Reemplazar todas las referencias de la var. temporal por el nuevo método
- Eliminar la declaración de la var. temporal y las asignaciones
- Compilar y testear

# Replace Temp With Query:

## reemplazando *thisAmount*

```
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;
}
}
```

# [ Sobre la performance ]

- La mejor manera de optimizar un programa, primero es escribir un programa bien factorizado y luego optimizarlo, previo profiling ...



# Extract & Move:

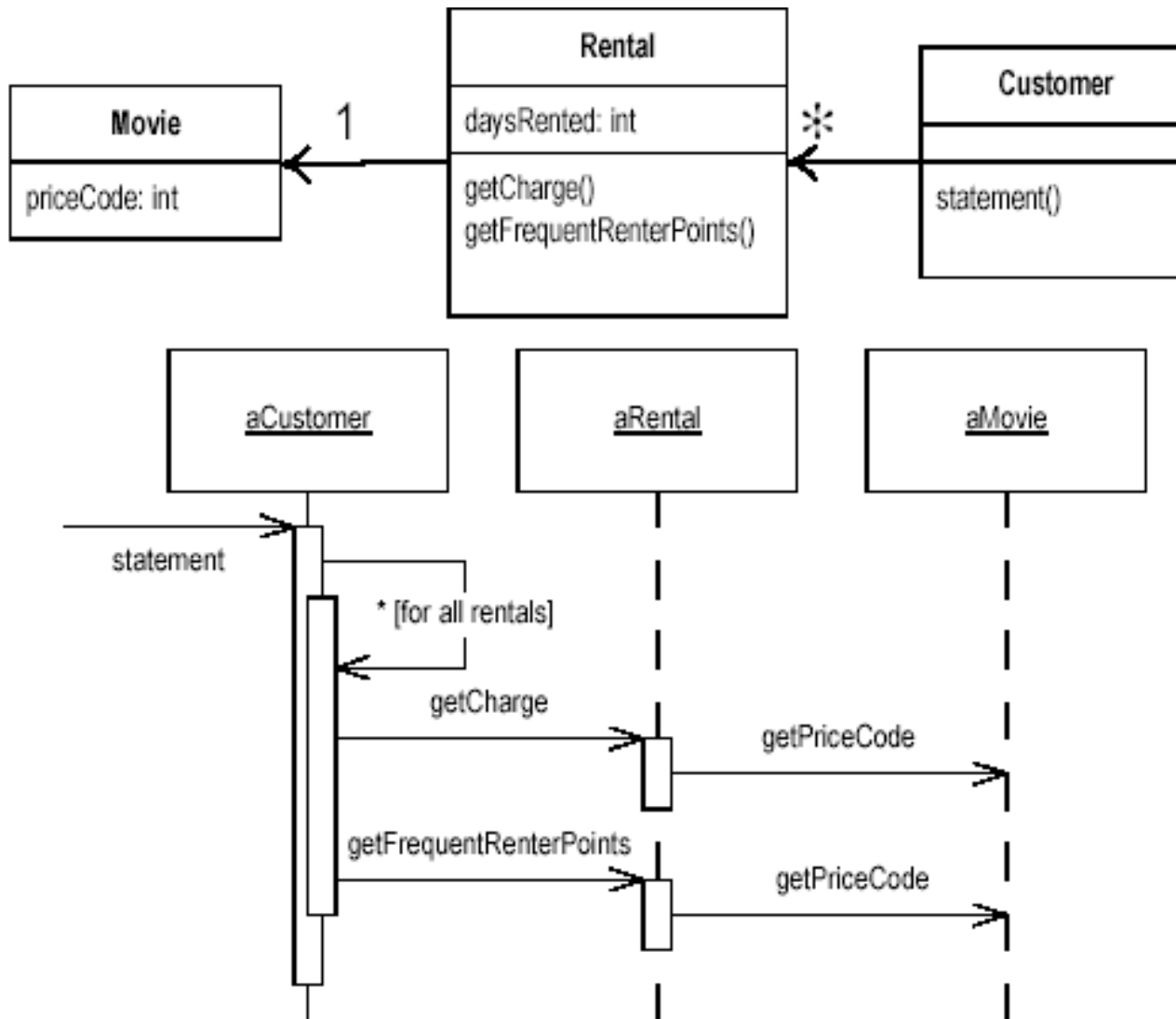
## frequentRenterPoint()

```
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;
    }
}
```

# [ Despues de Move & Replace ]



# [ Mas temporales para eliminar ]

```
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;
    }
}
```



# El nuevo método: **getTotalCharge()**

```
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;  
}
```



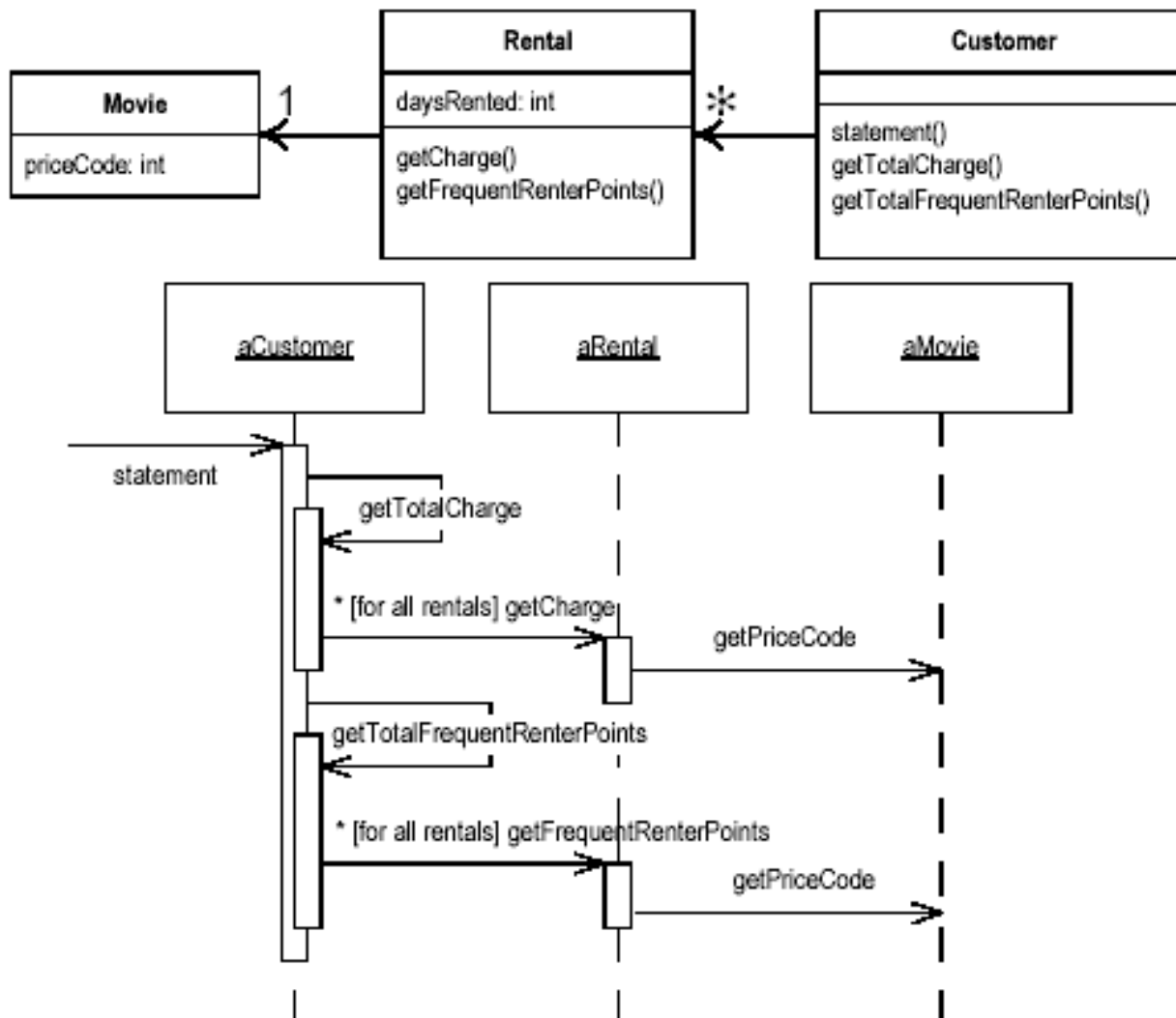
# [ Sin Vars. Temporales ]

```
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;
}
```

# [ Nuevo diseño ]



# [ Html statement ]

```
public String htmlStatement() {
    Enumeration rentals = _rentals.elements();
    String result = "<H1>Rentals for <EM>" + getName() + "</EM></H1><P>";
    while (rentals.hasMoreElements()) {
        Rental each = (Rental) rentals.nextElement();

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

    //add footer lines
    result += "<P>You owe " + String.valueOf(getTotalCharge()) + "\n";
    result += "You earned " +
        String.valueOf(getTotalFrequentRenterPoints()) +
            "frequent renter points<P>";
    return result;
}
```

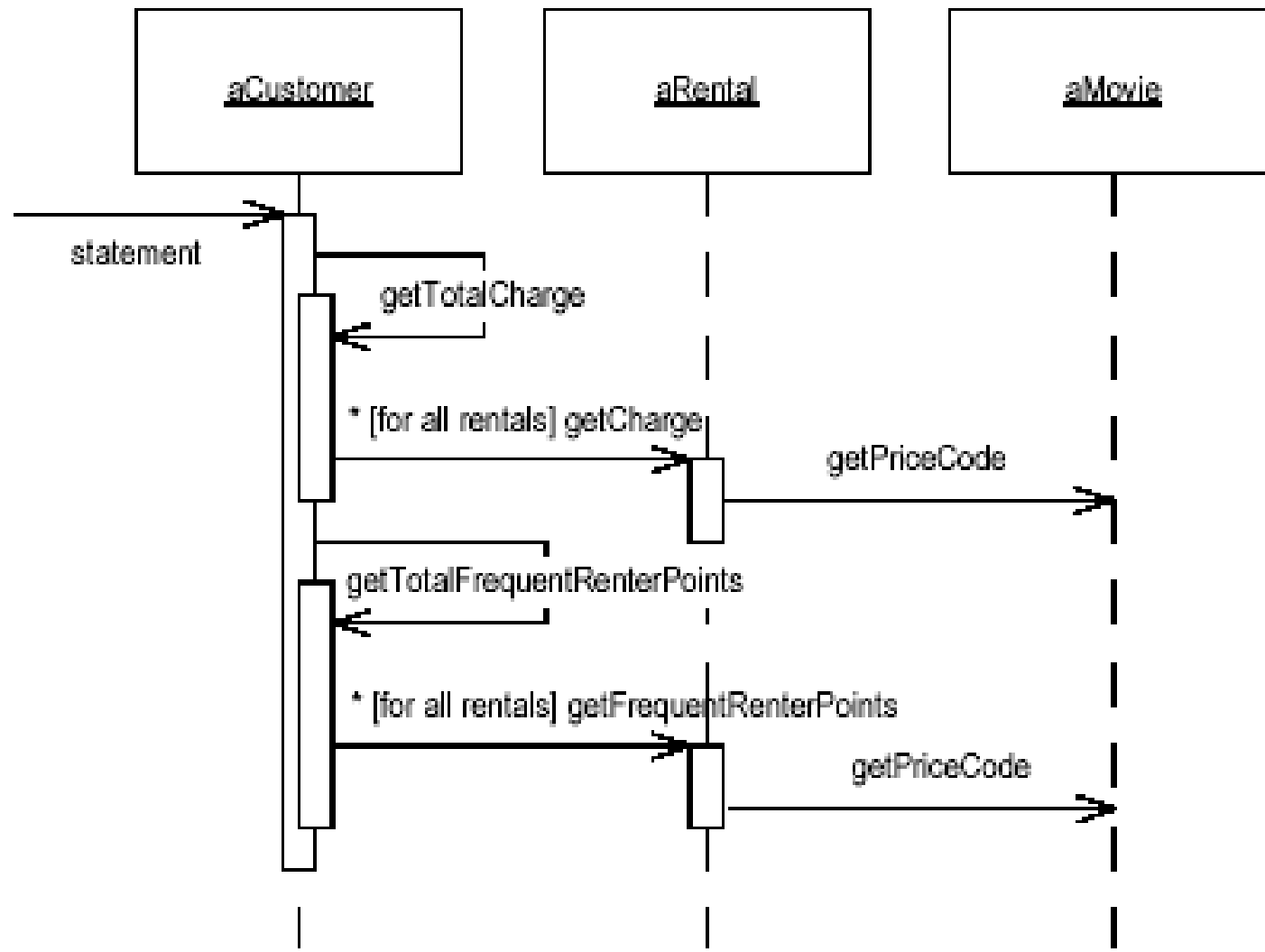
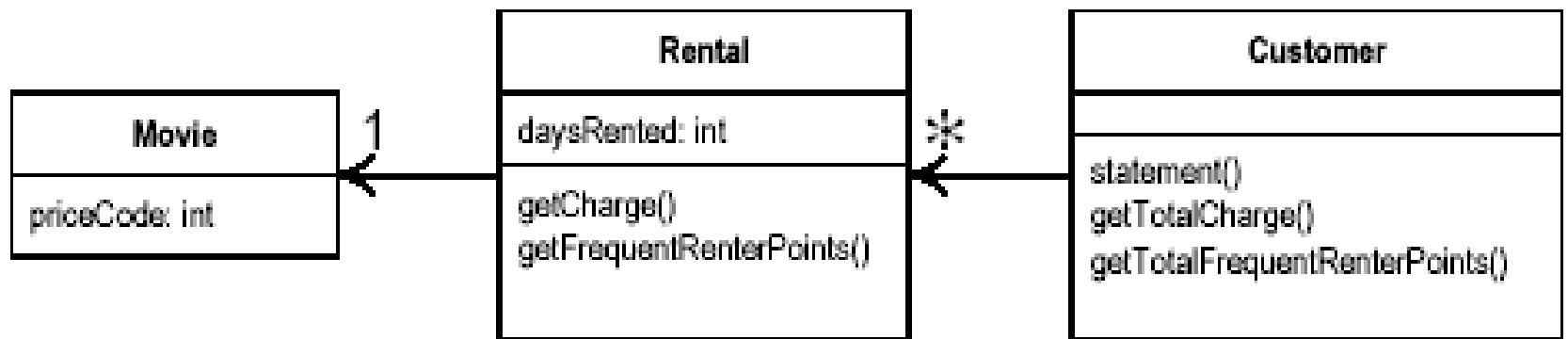
# [ Volviendo a la clase Rental ]

```
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;
    }
```

- ¿Qué problema seguimos teniendo? ¿Cómo se resuelve?

# [ Seguimos teniendo el switch ]

- ¿Cómo eliminar el switch?
- → Replace Conditional with Polymorphism
- ¿Tiene sentido hacer subclases de Rental? ¿Corresponde a Rental este cálculo?



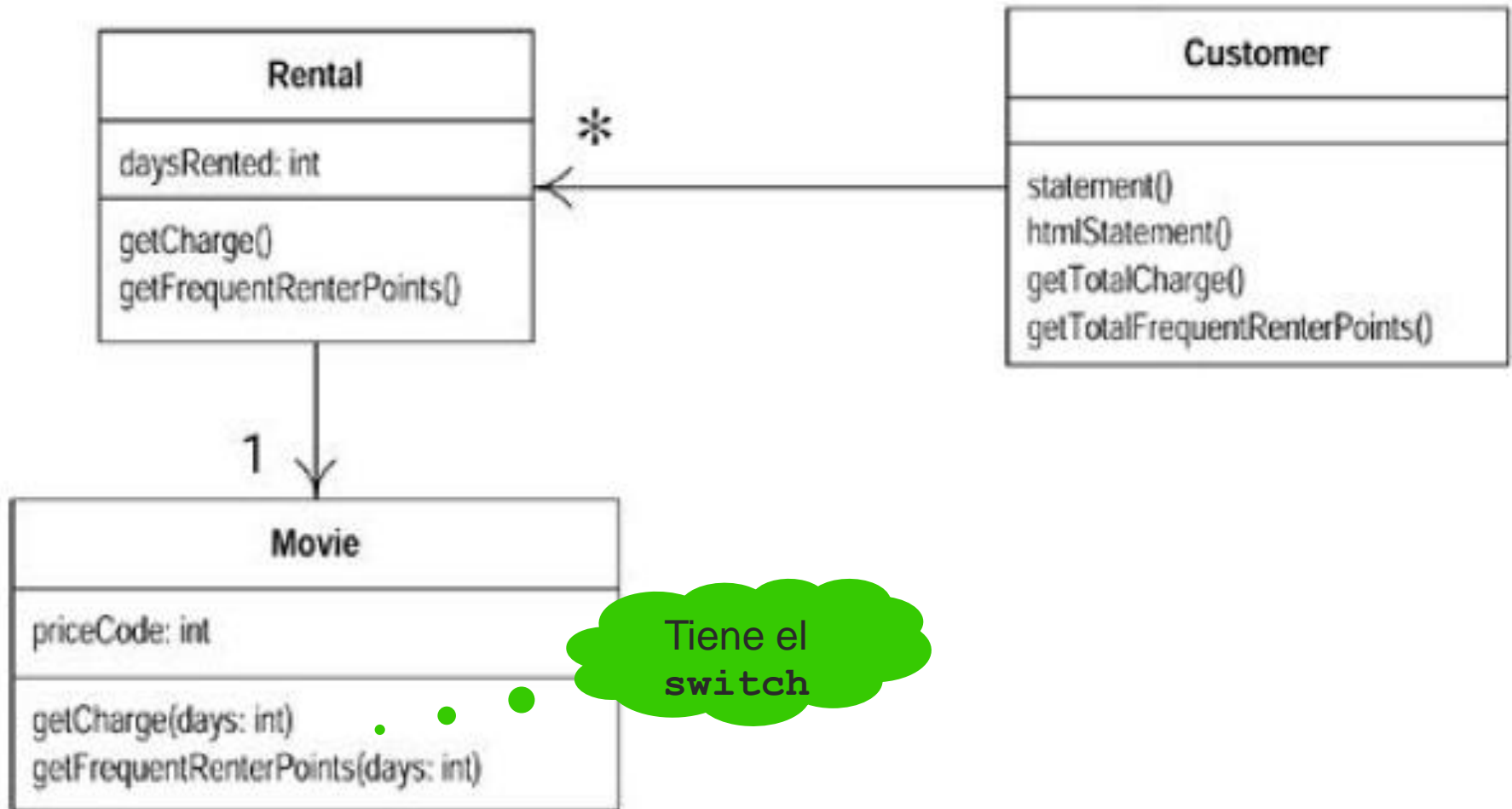


# Moviendo getCharge() a la clase Movie

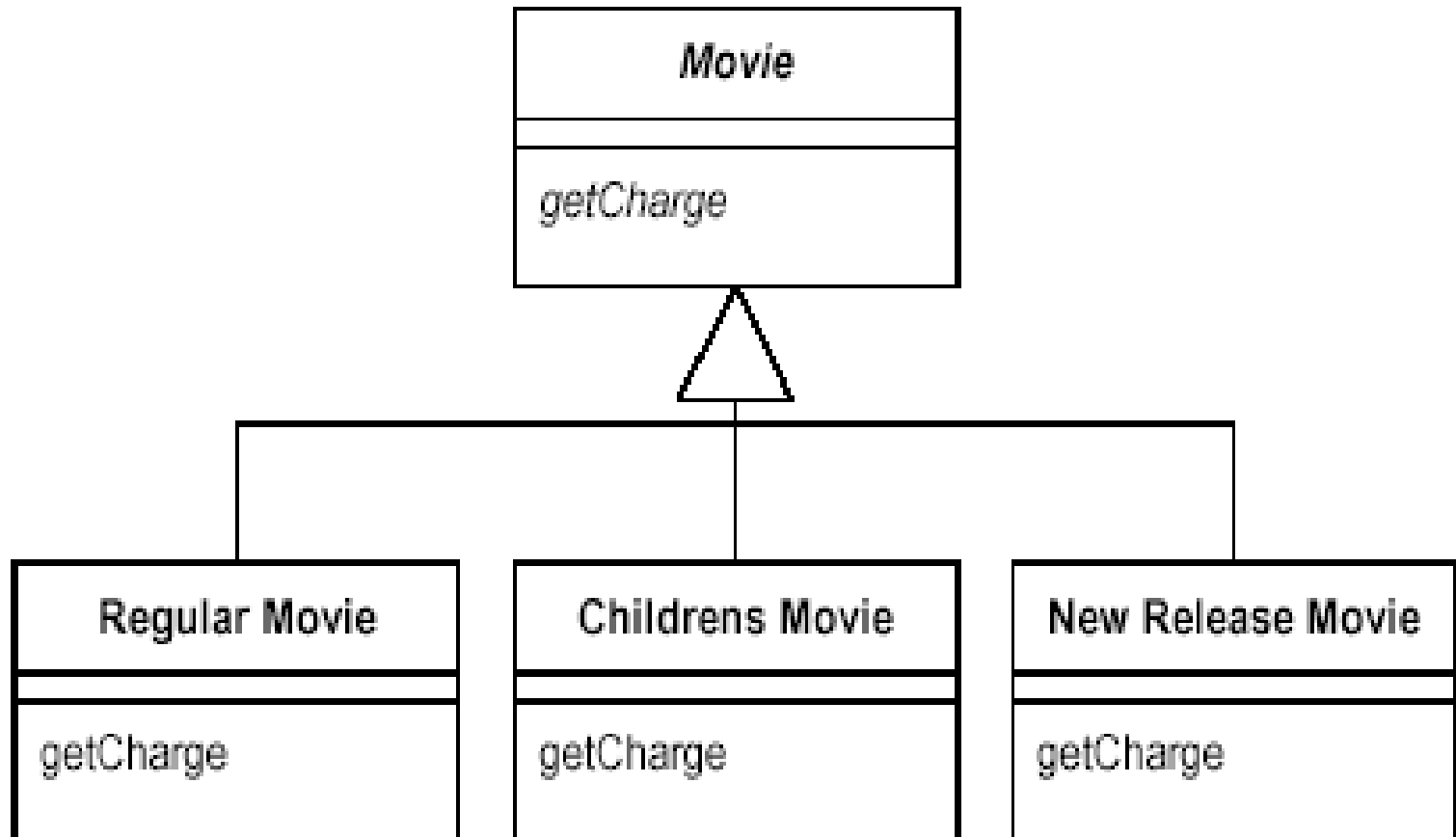
```
class Rental...  
    double getCharge() {  
        return _movie.getCharge(_daysRented);  
    }  
}
```

```
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;  
    }  
}
```

# [ Entonces queda ]



# [ Considerando el *polimorfismo* ]



# [ Replace Conditional with Polymorphism ]

- Crear la jerarquía.
- Por cada variante, crear un método en cada subclase que redefina el de la superclase.
- Copiar al método de cada subclase la parte del condicional correspondiente.
- Compilar y testear.
- Borrar de la superclase la sección (branch) del condicional que se copió.
- Compilar y testear.
- Repetir para todos los branches del condicional.
- Hacer que el método de la superclase sea abstracto.

# [Resultado de Repl.Cond w/ Pol.]

**Class Movie...**

**abstract double**

**getCharge(int daysRented);**

**Class RegularMovie**

**double getCharge(int daysRented){**

**double result = 2;**

**if (daysRented > 2) result += (daysRented - 2) \* 1.5;**

**return result;**

**}**

**Class ChildrensMovie**

**double getCharge(int daysRented){**

**double result = 1.5;**

**if (daysRented > 3) result += (daysRented - 3) \* 1.5;**

**return result;}**

**Class NewReleaseMovie**

**double getCharge(int daysRented){**

**return daysRented \* 3;}**

# [ Catálogo de Refactorings ]

- Extract Method
- Rename
- Move Method
- Replace Temp With Query
- Encapsulate Field
- Pull Up/Down Method/Field
- Replace Conditional with Polimorfism
- Extract Superclass/Subclass
- etc

# [Malos olores]

- Código duplicado
  - Extract Method
  - Pull Up Method
  - Form Template Method
- Métodos largos
  - Extract Method
  - Decompose Conditional
  - Replace Temp with Query
- Clases grandes
  - Extract Class
  - Extract Subclass
- Muchos parámetros
  - Replace Parameter with Method
  - Preserve Whole Object
  - Introduce Parameter Object

# [Malos olores (2)]

- Cambios divergentes
  - Extract Class
- “Shotgun surgery”
  - Move Method/Field
- Envidia de atributo (Feature Envy)
  - Move Method
- Sentencias Switch
  - Replace Conditional with Polymorphism
- Lazy class
  - Inline Class
- Generalidad especulativa
  - Collapse Hierarchy
  - Inline Class
- Cadena de mensajes
  - Hide Delegate
  - Extract Method

# [Malos olores (3)]

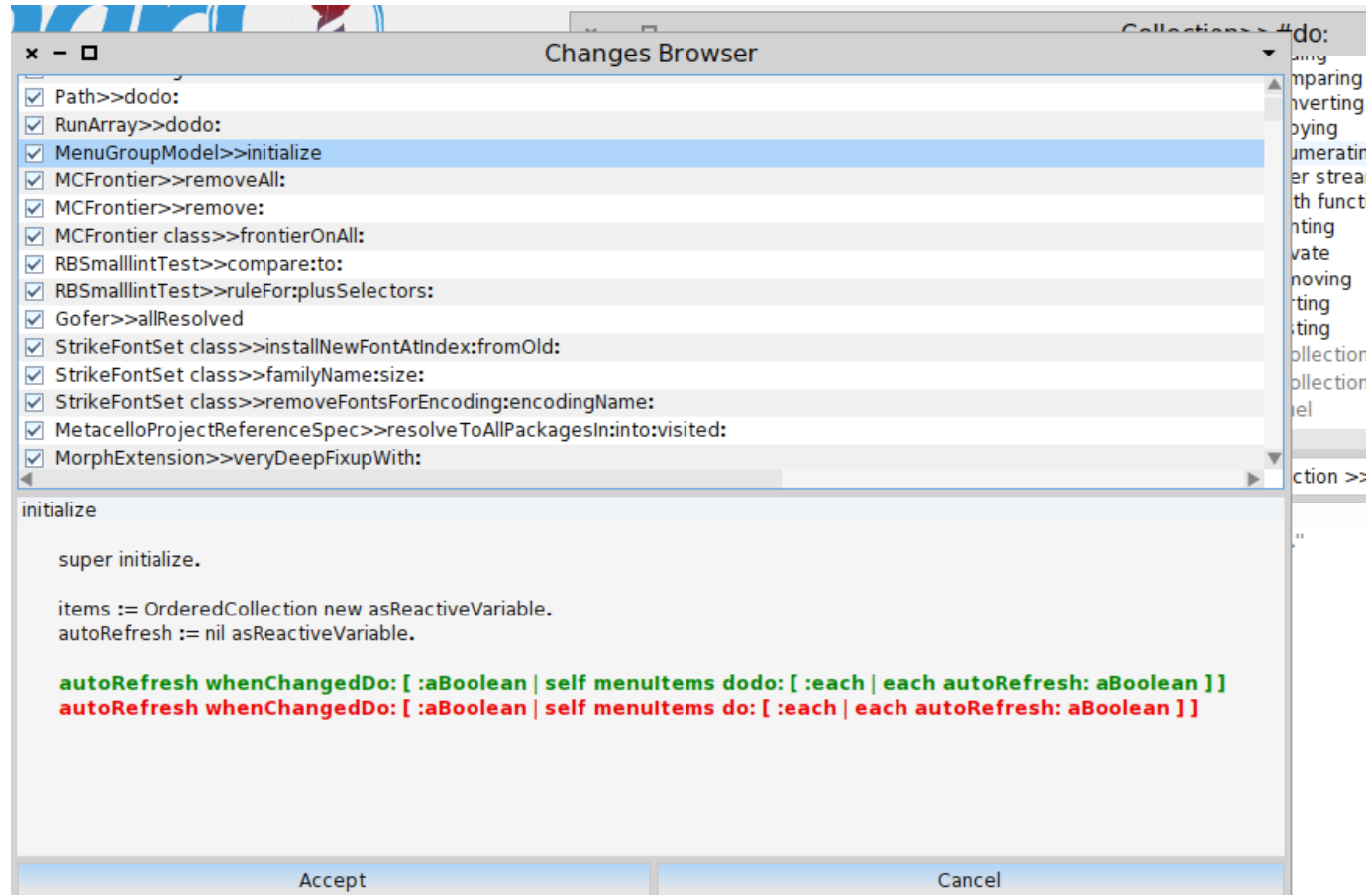
- Middle man
  - Remove Middle man
- Inappropriate Intimacy
  - Move Method/Field
- Data Class
  - Move Method
- Legado rechazado
  - Push Down Method/Field
- Comentarios
  - Extract Method
  - Rename Method

# [ Catálogo de Fowler ]

- Refactoring manual
- Formato:
  - Nombre
  - Motivación
  - Mecánica
  - Ejemplo
- Por qué necesitamos aprenderlo?

# Rename Method con el Refactoring Browser (RB)

Selecciono  
el método,  
botón derecho,  
“Rename  
method”



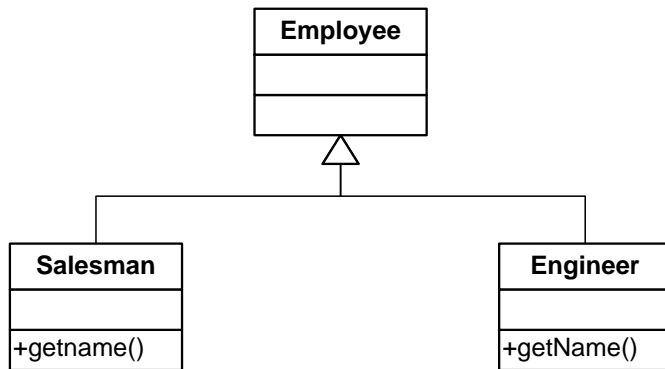
# [“Rename” con el RB]

- Se complica cuando hay otras implementaciones del método viejo que no queramos renombrar

# [Rename Method. Mecánica]

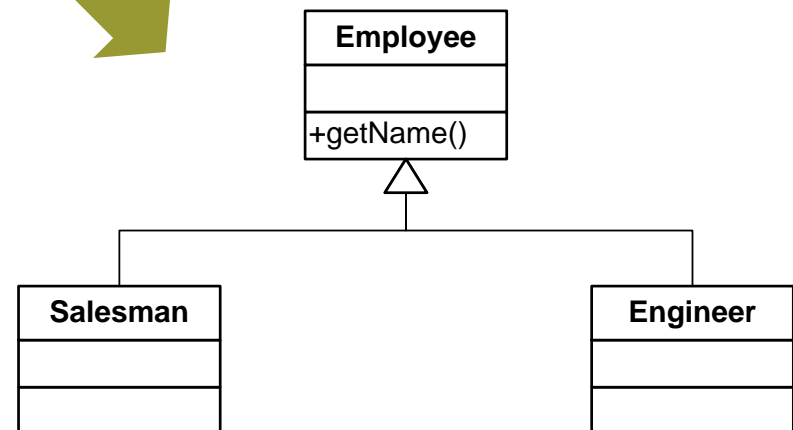
- Crear un método con el nuevo nombre (que no esté usado)
- Copiar el código del método viejo al nuevo
- Compilar
- Cambiar el cuerpo del método viejo para que llame al nuevo
- Compilar y testear
- Encontrar todas las llamadas al método viejo y cambiarlas para que llamen al nuevo; compilar y testear después de cada cambio
- Borrar el método viejo (o marcarlo *deprecated*)
- Compilar y testear.

# [ Otro caso: Pull Up Method ]



Puede ocurrir:

- distinto nombre en las subclases
- cuerpo no totalmente idéntico
- uso de var. declarada en subclases
- uso de métodos definidos en subclases



# [Además]

- No hay herramientas que sugieran refactorings, aunque hay algunas que encuentran bad smells, como el critic browser
- Ninguna herramienta de refactoring puede determinar cuales son las abstracciones significativas que hacen al programa más fácil de entender o extender
- Muchas veces sólo el desarrollador sabe cómo mejorar el diseño

# [ ¿Por qué refactoring es importante? ]

- Nuestra única defensa contra el deterioro del software.
- Facilitar la incorporación de código
- Permite *agregar patrones* después de haber escrito el programa; permite *transformar un programa en framework*.
- Permite preocuparse por la generalidad mañana; hoy solo hay que hacerlo andar  
“*Make it work. Make it right. Make it fast*”. Kent Beck.
- “*Necessary for beautiful software*”. Ralph Johnson



# [Referencias]

- Agile Manifesto. <http://agilemanifesto.org/>
- “Refactoring. Improving the Design of Existing Code”. Martin Fowler. Addison Wesley. 1999.
- “Technical Excellence”. David Bernstein. <https://www.linkedin.com/pulse/technical-excellence-david-bernstein>