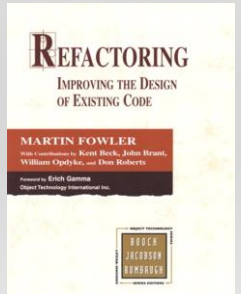
Présentation du tutoriel :

Premier Exemple de Refactoring

Écrit à partir de l'exemple *simplifié* du premier chapitre de

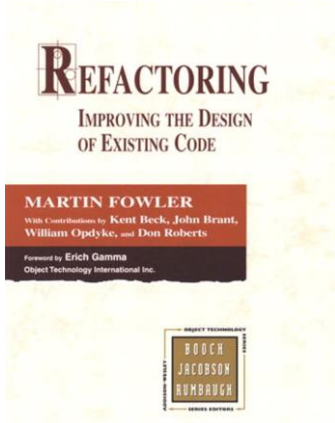


Tutoriel complet disponible sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler

Isabelle BLASQUEZ

Un refactoring (remaniement) consiste
à changer la structure interne d'un logiciel
sans en changer son comportement observable
(M. Fowler)

Ce tutoriel s'appuie sur l'exemple *simplifié* du premier chapitre du livre [Refactoring, Improving the Design of Existing Code de Martin Fowler]



Présentation et Compréhension de l'existant

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_presentationExemple.md

Isabelle BLASQUEZ

L'existant ...

Contexte :
Programme destiné à calculer et afficher le relevé de compte des locations d'un client d'un vidéo-club.

Images : https://www.realestatemarketingblog.org/how-to-make-a-viral-real-estate-video/

Sources :
3 classes Java :

→ **`Customer`** pour le client
→ **`Movie`** pour la vidéo
→ **`Rental`** pour la location

disponible sur : https://github.com/iblasquez/Refactoring_PremierExempleFowler/tree/master/src

Legacy code ... (1/2)

```
public class Movie {

   public static final int REGULAR = 0;
   public static final int NEW_RELEASE = 1;

   private String title;
   private int priceCode;

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

   public int getPriceCode() {
      return priceCode;
   }

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

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

Movie

```
public class Rental {

   private Movie movie;
   private int daysRented;

   public Rental(Movie movie, int daysRented) {
      this.movie = movie;
     this.daysRented = daysRented;
   }


   public int getDaysRented() {
   return daysRented;
   }

   public Movie getMovie() {
      return movie;
   }

}
```

Rental

Legacy code ... (2/2)

```java
public class Customer {

private String name;
private List<Rental> rentals = new ArrayList<Rental>();

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

public String getName() {
   return name;
}

public void addRental(Rental rental) {
   this.rentals.add(rental);
}
```

Customer

```java
// … suite de la classe Customer

public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

      double thisAmount = 0;

      // 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;
       }

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }

} // fin classe Customer
```

Zoom sur le code de la méthode `statement`

Des règles métiers :

pour implémenter le système de tarification suivant :

→ **Location d'une vidéo ordinaire** :
2 € pour les deux premiers jours de location,
puis 1.5 € pour chaque jour de location supplémentaire

→ **Location d'une nouveauté** :
3 € pour chaque jour de location

… sachant que les vidéos pourront passer de nouveauté à ordinaire à l'aide de `setPrice` de la classe `Movie`)

Et en sortie :

→ un **affichage (format ASCII)** similaire à :

```
Rental Record for Bob
        Star Wars       6.0
        The Hobbit      2.0
Amount owed is 8.0
```

```
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

      double thisAmount = 0;

      // 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;
       }

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
```

Reverse Engineering …

Diagramme de classes

(Obtenu à partir du plug-in ObjectAid UML Explorer)

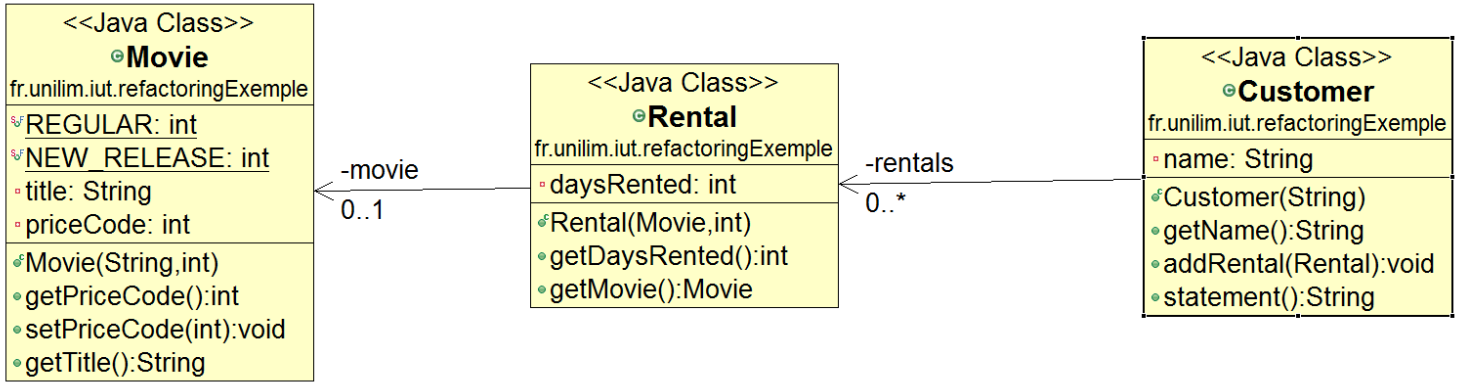
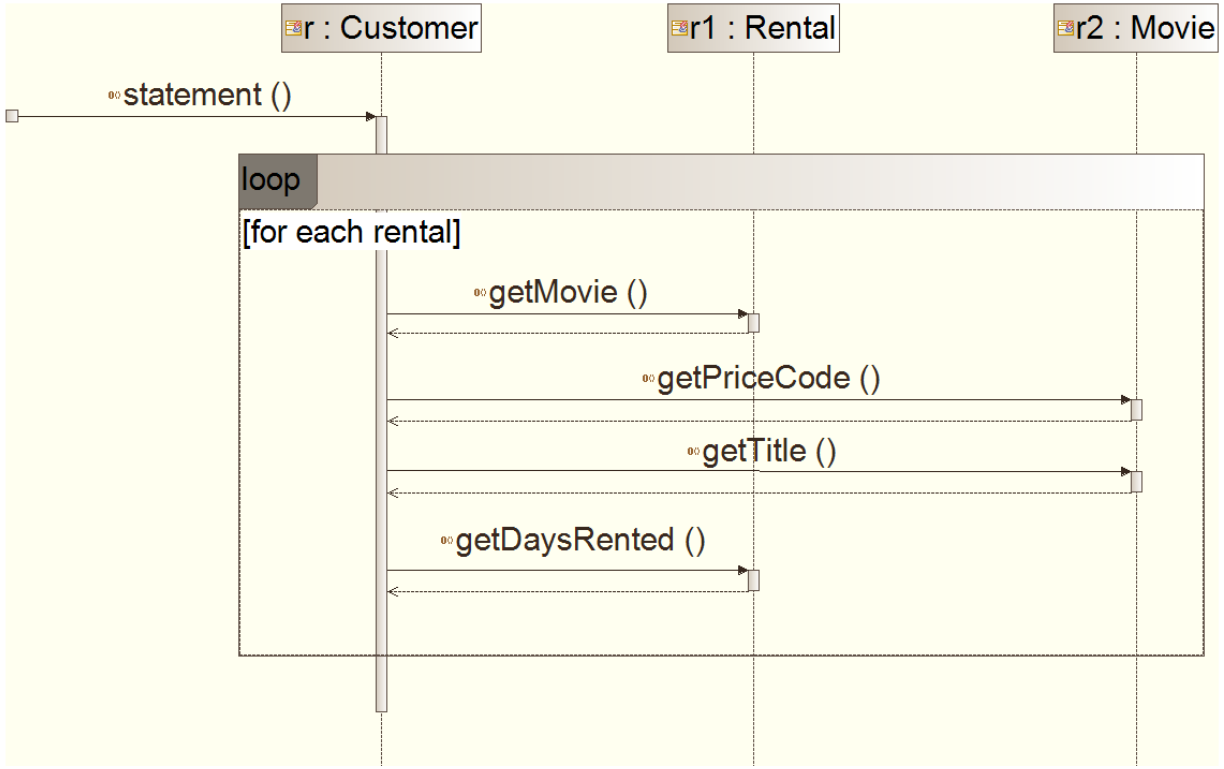


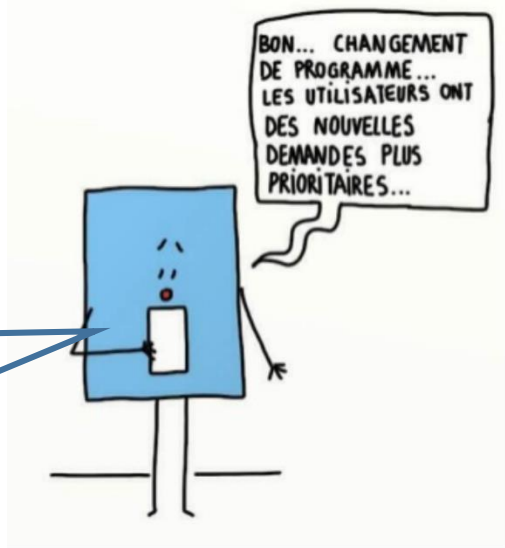
Diagramme de séquences de la méthode `statement`



Un nouveau besoin ....

On souhaiterait que
le **relevé de compte**
puisse également être
disponible au format HTML

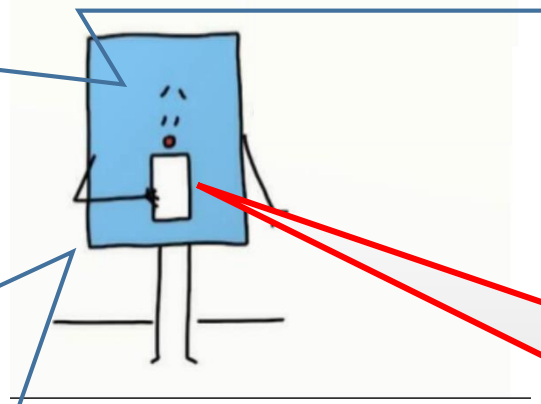
c-à-d avoir le choix entre les deux formats
(ASCII ou HTML)



Crédit image : https://uneviededev.com/

Comment répondre à ce nouveau besoin ? …

Hmmm, Hmmm…
modification pas vraiment possible, pour l'instant,
dans l'actuelle méthode `statement`
qui traite **à la fois du calcul** des frais de location
et l'affichage du relevé de compte …



Un copier/coller de `statement` en `htmlStatement` ?
⇒ Quid en cas de modification du système de tarification ?
(Pb **duplication &**
maintien **cohérence** du système …)

⇒ Un refactoring s'impose !!!!

Pas de refactoring sans test !

act[illegible]_PremierExempleFowler/blob/master/refactoring_Step0_miseEnPlaceTests.md

Un test de qualité (*clean* test) …

Propriétés du test

- **F** AST
- **I** NDEPENDANT
- **R** EPEATABLE
- **S** ELF-VALIDATING
- **T** IMELY

Pattern d'écriture du test

- **A** RRANGE
- **A** CT
- **A** SSERT

Un premier test pour une vidéo ordinaire ...

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class CustomerTest {

    @Test
    public void statementForOneRegularMovie() {

        Customer customer = new Customer("Alice");
        Movie movie = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental = new Rental(movie, 3); // 3 days rental
        customer.addRental(rental);


        String statement = customer.statement();


        String expected = "Rental Record for Alice\n" +
                "\tThe Lord of the Rings\t3.5\n" +
                "Amount owed is 3.5\n";
        assertEquals(expected, statement);
    }
}
```

Le nom du test explicite le comportement testé

ARRANGE

ACT

ASSERT

Couverture de code …

```
    public String statement() {

        double totalAmount = 0;

        String result = "Rental Record for " + this.getName() + "\n";

        for (Rental each : rentals) {

            double thisAmount = 0;

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

… Pour une couverture totale des vidéos ordinaires, il faudrait aussi tester une location de moins de 2 jours …

Mais testons avant le comportement des nouveautés !

Un deuxième test pour une nouveauté …

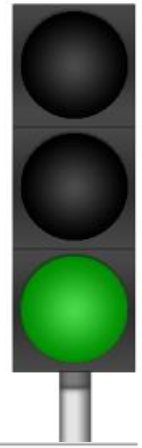
```java
import static org.junit.Assert.*;
import org.junit.Test;

public class CustomerTest {

// …

      @Test
    public void statementForOneNewReleaseMovie() {

        Customer customer = new Customer("Bob");
        Movie movie = new Movie("Star Wars", Movie.NEW_RELEASE);
        Rental rental = new Rental(movie, 3); // 3 day rental
        customer.addRental(rental);

        String statement = customer.statement();

        String expected = "Rental Record for Bob\n" +
                "\tStar Wars\t9.0\n" +
                "Amount owed is 9.0\n";
        assertEquals(expected, statement);
    }

}
```

Le nom du test explicite le comportement testé

Un dernier test pour plusieurs vidéos dont une location de moins de 2 jours ...

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class CustomerTest {

// …

 @Test
    public void statementForManyMovies() {
        Customer customer = new Customer("Bob");
        Movie movie1 = new Movie("Star Wars", Movie.NEW_RELEASE);
        Rental rental1 = new Rental(movie1, 2); // 2 day rental
        Movie movie2 = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental2 = new Rental(movie2, 1); // 1 day rental
        customer.addRental(rental1);
        customer.addRental(rental2);

        String statement = customer.statement();

        String expected = "Rental Record for Bob\n" +
                "\tStar Wars\t6.0\n" +
                "\tThe Lord of the Rings\t2.0\n" +
                "Amount owed is 8.0\n";
        assertEquals(expected, statement);
    }
}
```

Le nom du test explicite le comportement testé



JUnit

Finished after 0,016 seconds

Runs: 3/3 Errors: 0 Failures: 0

fr.unilim.iut.refactoringExemple.CustomerTest [Runner: JUnit 4] (0,000 s)

- statementForManyMovies (0,000 s)
- statementForOneRegularMovie (0,000 s)
- statementForOneNewReleaseMovie (0,000 s)

On ne vise pas le 100% de couverture de code ...

Coverage

Element	Coverage
refactoringExemple	98,4 %
src/main/java	97,3 %
fr.unilim.iut.refac	97,3 %
Movie.java	78,9 %
Customer.java	100,0 %
Rental.java	100,0 %

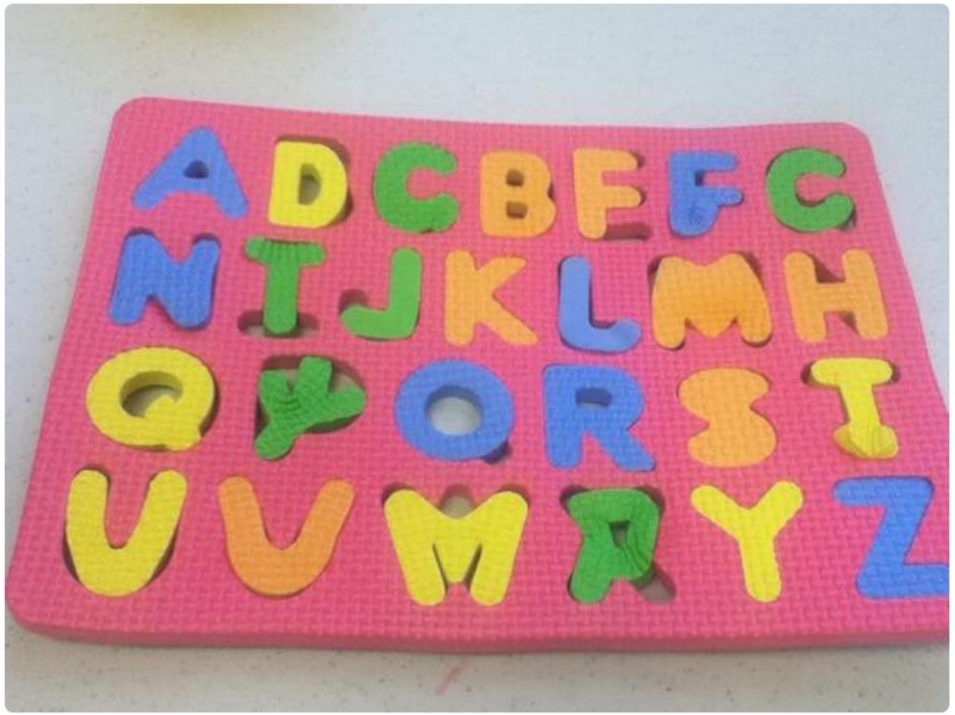
Couverture de code

Movie.java	78,9 %
Movie	78,9 %
setPriceCod	0,0 %
Movie(String	100,0 %
getPriceCod	100,0 %
getTitle()	100,0 %

`setPrice` non couverte par exemple ... Intérêt de couvrir getteur & setteur ?

D'autre part (à méditer) ...

100% code coverage
"@bloerwald: SUCCESS: 26/26 (100%) Tests passed "



Et maintenant, let’s go refactoring !

… Et n’oubliez pas
de relancer les tests
aussi fréquemment que possible
pour vérifier que
le comportement est toujours garanti

Connaissez-vous Infinitest ?

infinitest

Source : https://uneviededev.com/

Isabelle BLASQUEZ

Refactoring

Etape 1 :

Isoler le calcul du montant des frais d'une location de l'affichage dans le relevé

(à l'aide d'un `Extract method`)

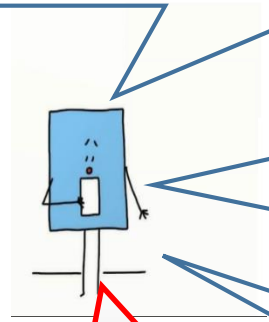
Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step1_ExtractMethod.md

Isabelle BLASQUEZ

Par où commencer ? ....

Quid de la méthode `statement` ?

Méthode `statement` un peu longue ...



... Deux traitements entremêlés : calcul ET affichage

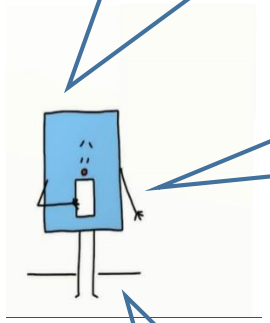
⇒ Intention du code un peu confuse ...

⇒ Et si on commençait par isoler le calcul du montant des frais d'une location ? ...

```
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

      double thisAmount = 0;

      // 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;
       }

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
```

Pour isoler, extraire le code dans une nouvelle méthode …

Extract Method
à l'aide des outils de Refactoring de votre IDE



Un nom de méthode qui montre l'intention du code: **getAmount**

Modification du code
⇒ N'oubliez pas de relancer les tests

```
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

     double thisAmount = 0;

    // determine amounts for each line
    thisAmount = getAmount(each, thisAmount);

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
```

```
private double getAmount(Rental each, double thisAmount) {

   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;
   }
return thisAmount;
}
```

Zoom sur l'Extract Method

Un des refactorings les plus utilisés ...

Problem

You have a code fragment that can be grouped together.

```
void printOwing() {
  printBanner();

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

Solution

Move this code to a separate new method (or function) and replace the old code with a call to the method.

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

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

Why Refactor

The more lines found in a method, the harder it is to figure out what the method does. This is the main reason for this refactoring.

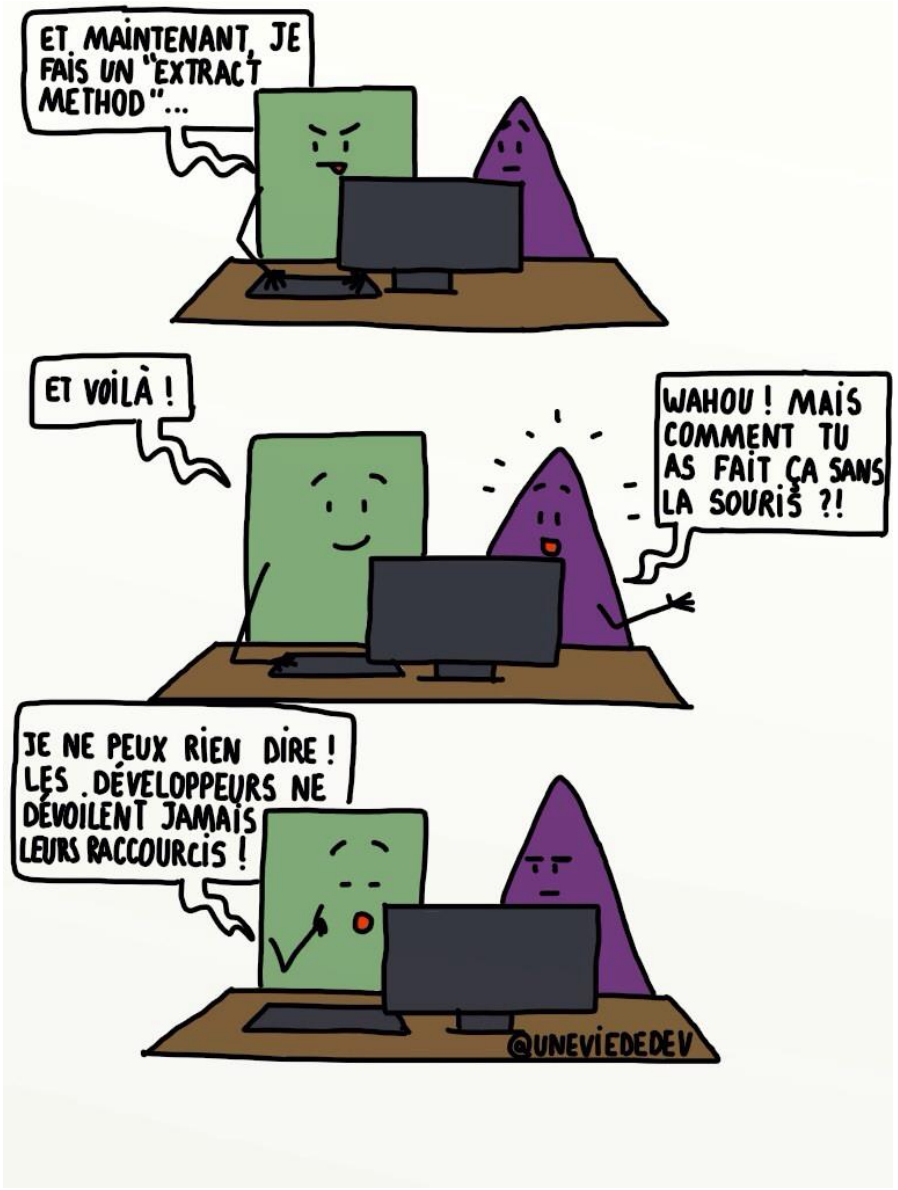
Besides eliminating rough edges in your code, extracting methods is also a step in many other refactoring approaches.

Benefits

- More readable code! Be sure to give the new method a name that describes the method's purpose: `createOrder()`, `renderCustomerInfo()`, etc.
- Less code duplication. Often the code that is found in a method can be reused in other places in your program. So you can replace duplicates with calls to your new method.
- Isolates independent parts of code, meaning that errors are less likely (such as if the wrong variable is modified).

Source : https://sourcemaking.com/refactoring/extract-method

Apprenez à maîtriser votre IDE via les raccourcis claviers …



Source : https://uneviededev.com/

Extract Method

ALT+SHIFT+M

(Eclipse)

Pour améliorer l'`Extract Method` réalisé par l'IDE, ⇒ 2 questions à se poser …

Pertinence des paramètres d'entrée
de la nouvelle méthode générée automatiquement par l' `Extract Method` **? (**`getAmount`**)**

Lisibilité de la nouvelle méthode générée automatiquement par l'`Extract Method` **?**

Pertinence des paramètres d'entrée de `getAmount` ?

`each` est bien
un paramètre d'entrée

`thisAmount`
est plutôt
une variable locale

```
private double getAmount(Rental each, double thisAmount) {

   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;
   }
return thisAmount;
}
```

```
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

    // determine amounts for each line
    double thisAmount = getAmount(each);

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }

private double getAmount(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;
   }
return thisAmount;
}
```

Quid de la lisibilité de getAmount ?

```
private double getAmount(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;
    }
return thisAmount;
}
```

Renommage des variables

ALT+SHIFT+R

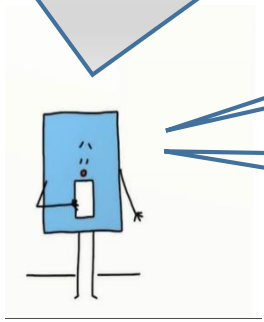
(Eclipse)

each → rental

thisAmount → result

Quid de la **lisibilité** ?

⇒ le nommage montre-t-il **l'intention du code** ?



each ?

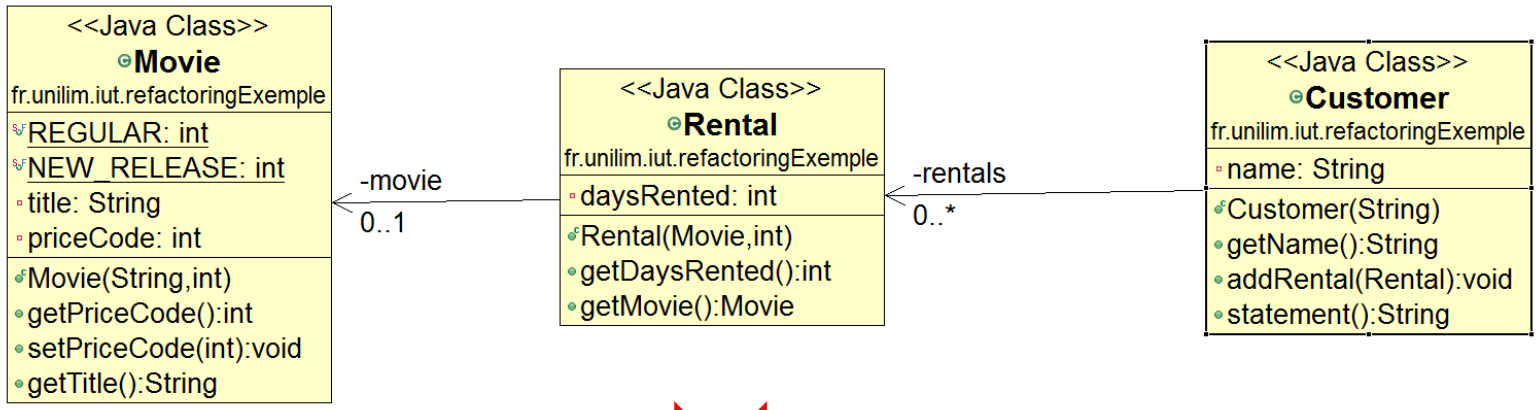
⇒ une location

thisAmount ?

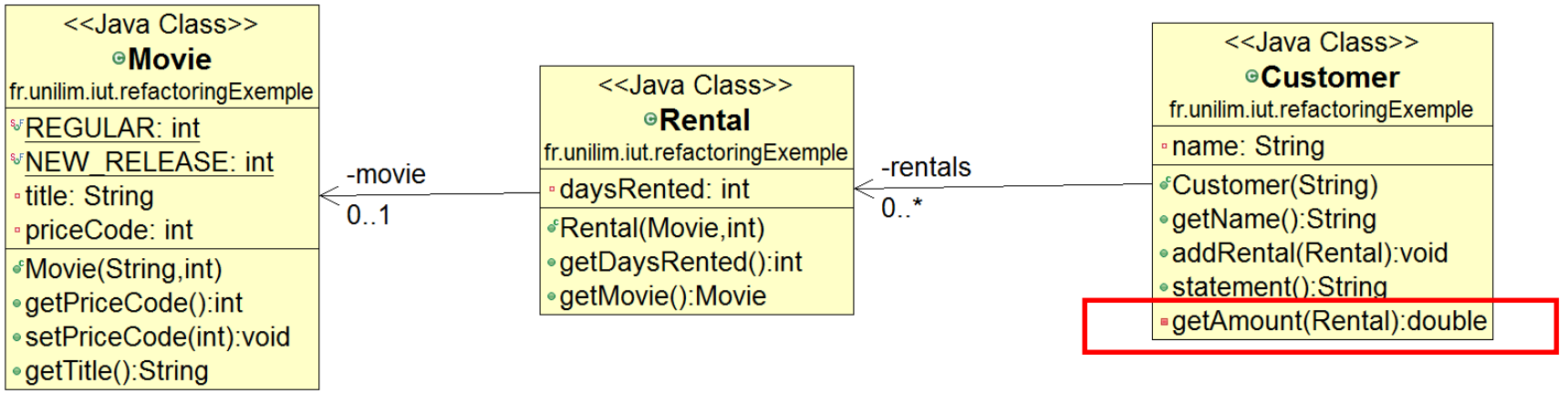
⇒ Le résultat du calcul du montant

```
private double getAmount(Rental rental) {

    double result = 0;

    switch (rental.getMovie().getPriceCode()) {

        case Movie.REGULAR:
            result += 2;
            if (rental.getDaysRented() > 2) {
                result += (rental.getDaysRented() - 2) * 1.5;
            }
            break;

        case Movie.NEW_RELEASE:
            result += rental.getDaysRented() * 3;
            break;
    }
return result;
}
```

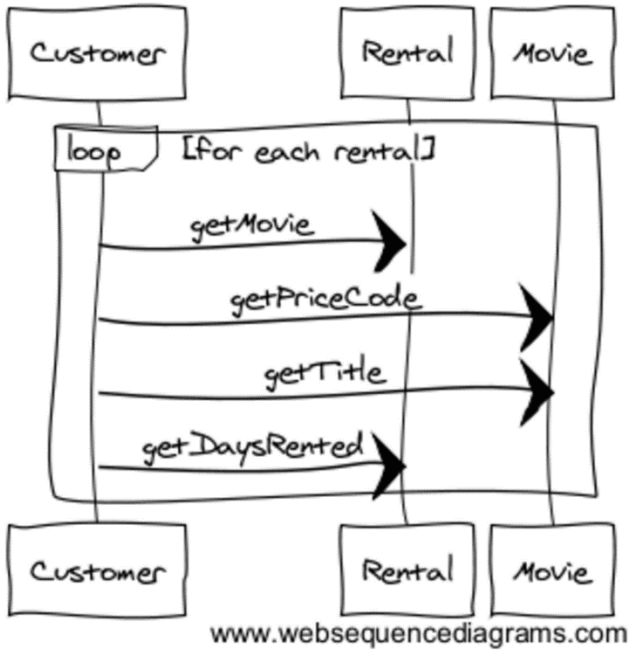
Evolution du diagramme de classes après l'`Extract Method`



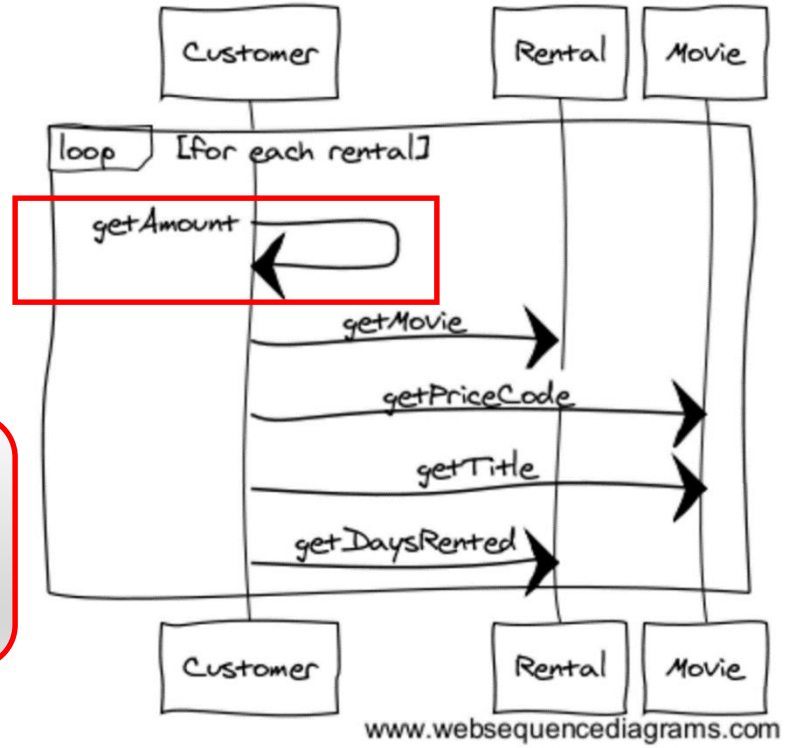
Etape 1 : Isolation
calcul montant des frais d'une location
& affichage relevé



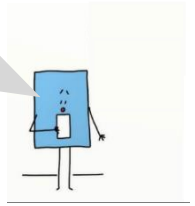
Evolution du diagramme de séquence après l'`Extract Method`



Etape 1 : Isolation
calcul montant des frais
d'une location
& affichage relevé



MAIS … les frais de **location** sont-ils bien de la responsabilité du **client** (`Customer`) ???...



Refactoring

Etape 2 :

Rendre la classe `Rental` responsable du calcul des frais d'une location

(à l'aide d'un `Move method`)

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step2_MoveMethod.md

Isabelle BLASQUEZ

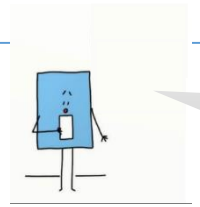
Déplacer `getAmount` vers la classe `Rental` à l'aide d'un `Move Method` de l'IDE

Customer

```
public class Customer {
// …
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

      // determine amounts for each line
      double thisAmount = each.getCharge();

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
}
```

Rental

```
public 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;
    }

   return result;
   }
}
```



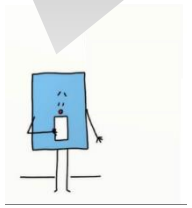
Renommer en **`getCharge`** pour montrer l'intention du code
⇒ **frais** *d'une location*

Inliner la variable locale `thisAmount` par `getCharge`

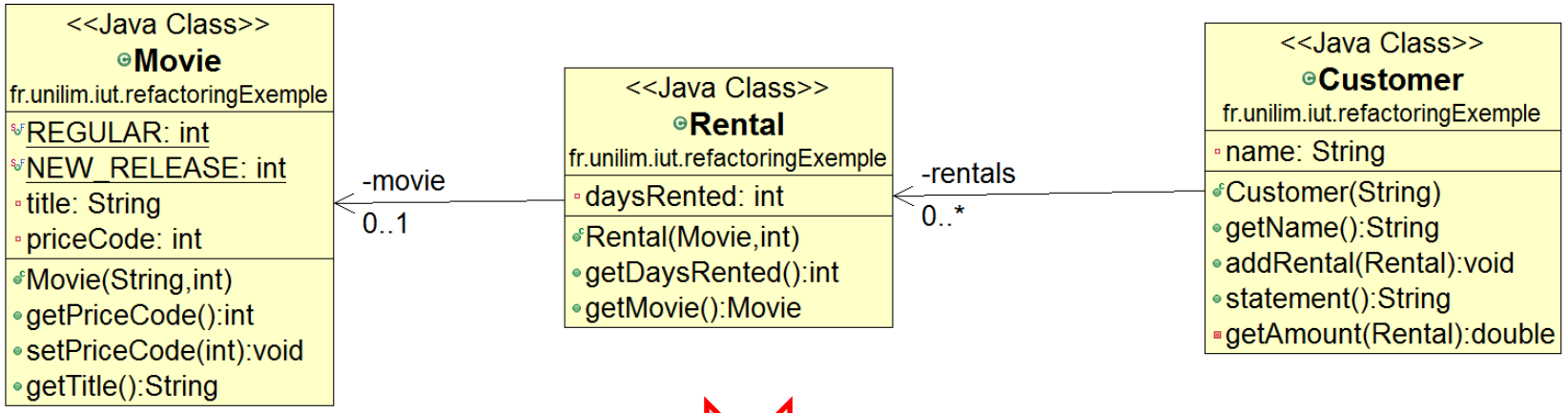
Customer

```
public class Customer {
// …
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
}
```

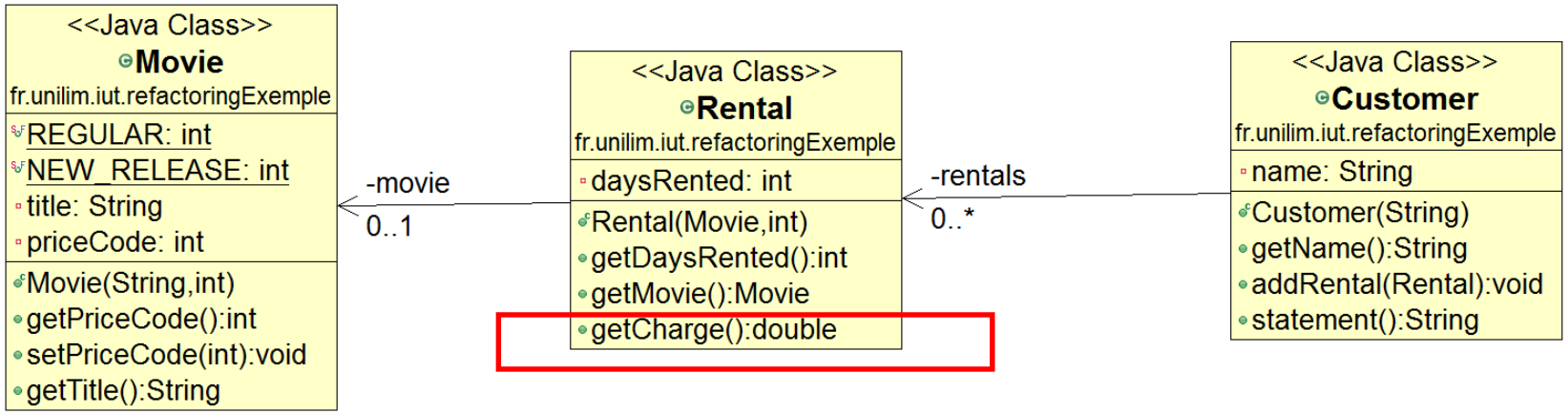
ATTENTION !! ***inliner*** n'est possible que si la variable temporaire ne subit qu'une seule affectation.



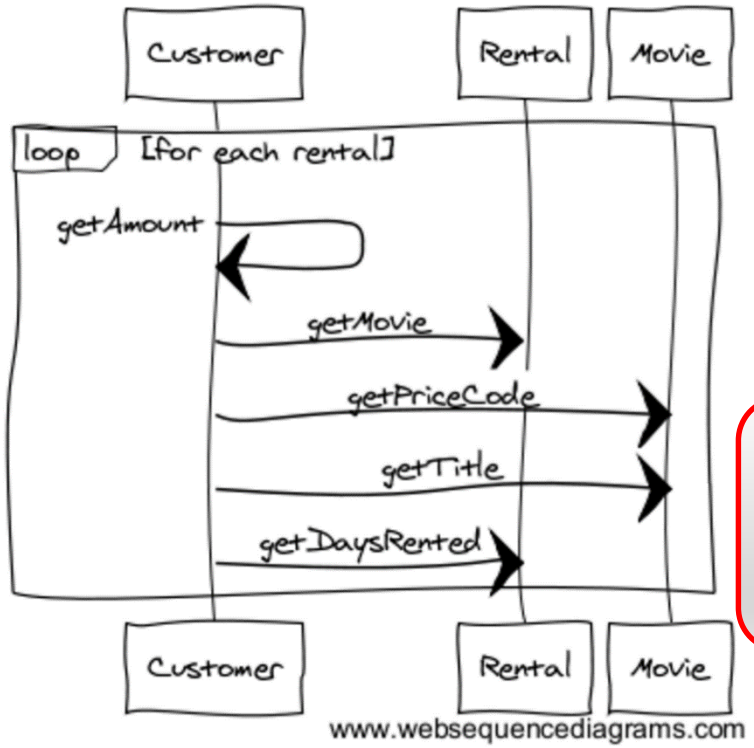
Evolution du diagramme de classes après le Move Method



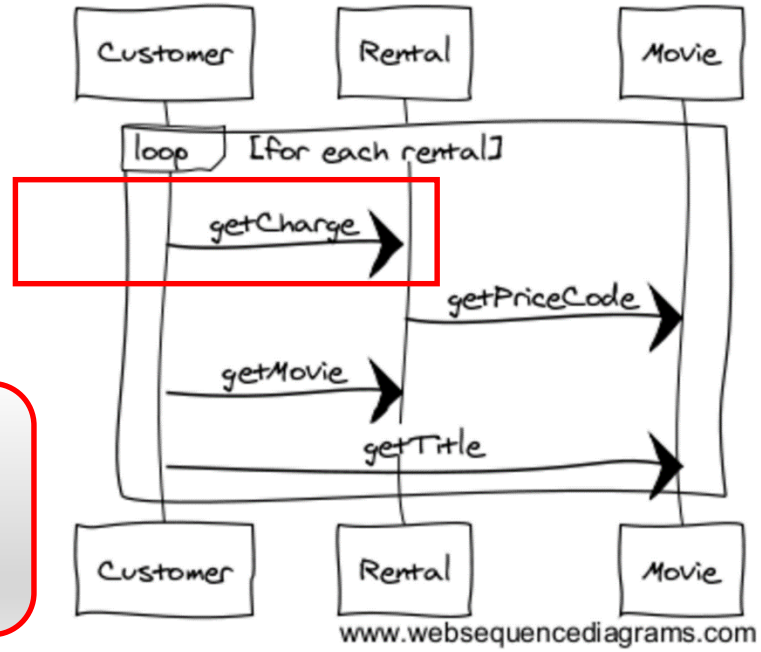
Etape 2 : Rendre la classe Rental responsable du calcul des frais d'une location



Evolution du diagramme de séquence après `Move Method`



Etape 2 : Rendre la classe `Rental` responsable du calcul des frais d'une location

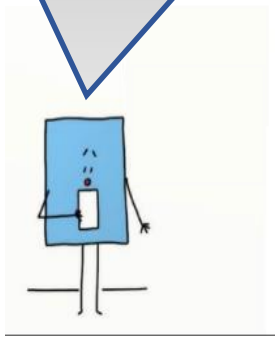


Et maintenant…Pourrait-on encore améliorer la méthode `statement` avant l'ajout de la nouvelle fonctionnalité ?

Customer

```
public class Customer {
// …
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
}
```

Hmmm, Hmmm
Il semblerait qu'il y ait encore
un calcul de montant (`totalAmount`)
dans l'affichage…
⇒
Le calcul de tous les frais de location ne pourrait-il pas être isolé de l'affichage ? ….



Refactoring

Etape 3 :

Isoler le calcul de la somme des frais de toutes les locations de l'affichage du relevé

(à l'aide d'un `Replace Temp With Query`)

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step3_ReplaceTempWithQuery.md

Isabelle BLASQUEZ

Extraire la méthode `getTotalCharge` nécessite au préalable d'isoler le calcul du montant total de l'affichage du relevé

```java
public class Customer {
// …
public String statement() {

   double totalAmount = 0;

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {

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

   // add footer lines
   result += "Amount owed is " + totalAmount + "\n";

   return result;
   }
}
```

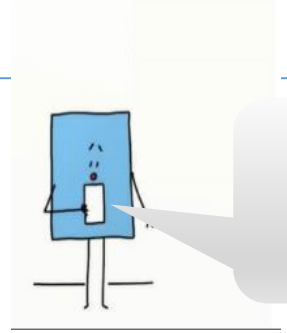
```java
public class Customer {
// …
public String statement() {

String result = "Rental Record for " + this.getName() + "\n";

for (Rental each : rentals) {
result += "\t" + each.getMovie().getTitle() + "\t"
                                + each.getCharge() + "\n";
}
```

Affichage

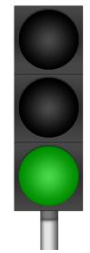
```java
double totalAmount = 0;
for (Rental each : rentals) {
totalAmount += each.getCharge();
}
```

Calcul

```java
result += "Amount owed is " + totalAmount + "\n";
return result;
}
```



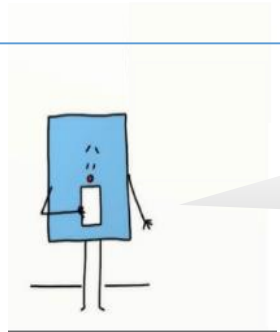
Dupliquer le code, un mal parfois nécessaire pour mieux isoler …



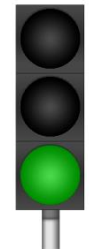
Extraire la méthode `getTotalCharge` et inliner la variable locale `totalAmount`

```
public class Customer {
// …
public String statement() {// …


String result = "Rental Record for " + this.getName() + "\n";

for (Rental each : rentals) {
result += "\t" + each.getMovie().getTitle() + "\t"
                               + each.getCharge() + "\n";
}

double totalAmount = 0;
for (Rental each : rentals) {
totalAmount += each.getCharge();
}


result += "Amount owed is " + totalAmount + "\n";
return result;
}
```

```
public class Customer {
// …

public String statement() {

   String result = "Rental Record for " + this.getName() + "\n";

   for (Rental each : rentals) {
      result += "\t" + each.getMovie().getTitle() + "\t"
                                          + each.getCharge() + "\n";
   }

   result += "Amount owed is " + getTotalCharge() + "\n";
   return result;
   }


private double getTotalCharge() {
   double totalCharge = 0;

   for (Rental each : rentals) {
      totalCharge += each.getCharge();
   }
   return totalCharge;
}

}
```



Avez-vous remarqué qu'on a supprimé les commentaires qui devenaient inutiles : le code montrant bien son intention …



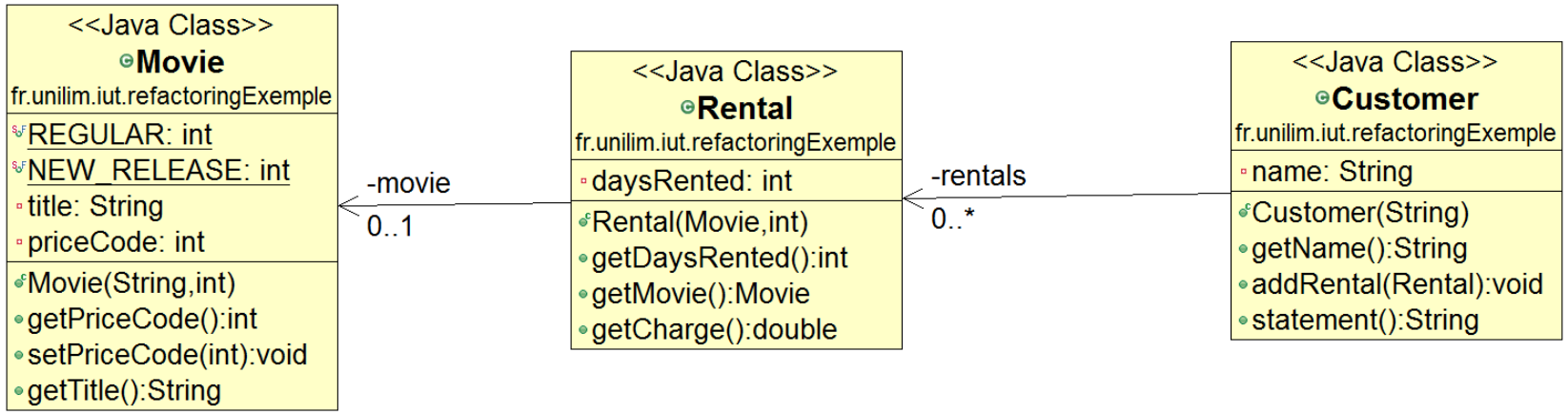
A propos des commentaires (à méditer …)

Overcommented Code

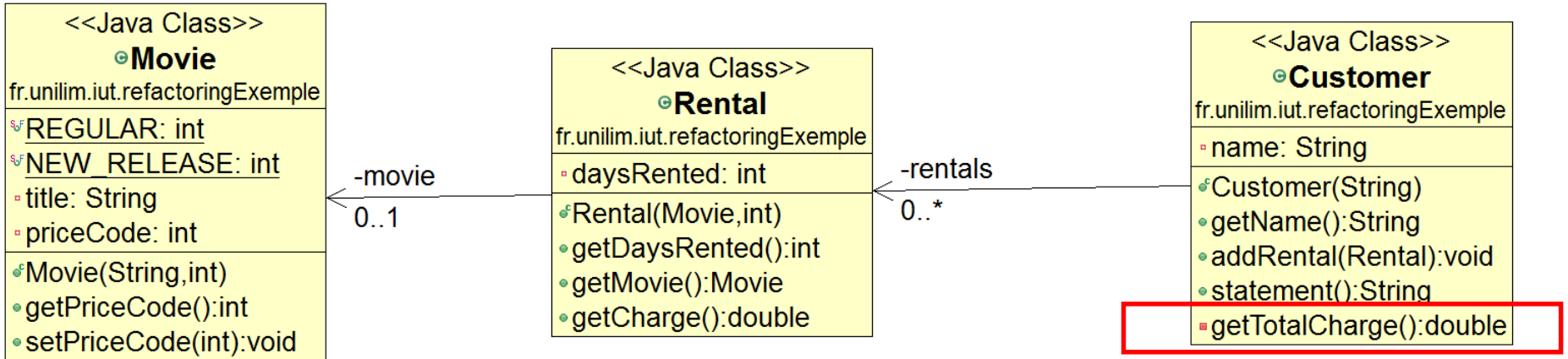
Code comments

Sources :
https://twitter.com/CiaranMcNulty/status/517654863623487488
https://twitter.com/nzkoz/status/538892801941848064
https://twitter.com/wraithgar/status/601497059577987073

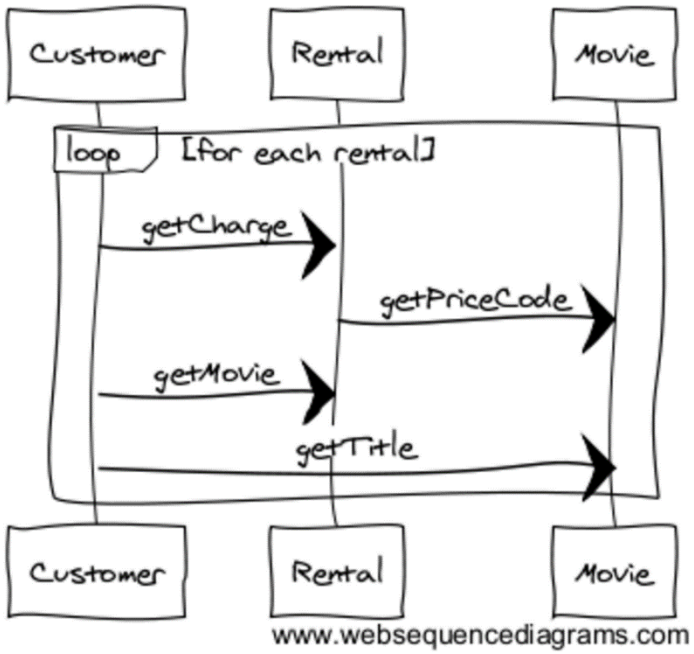
Evolution du diagramme de classes



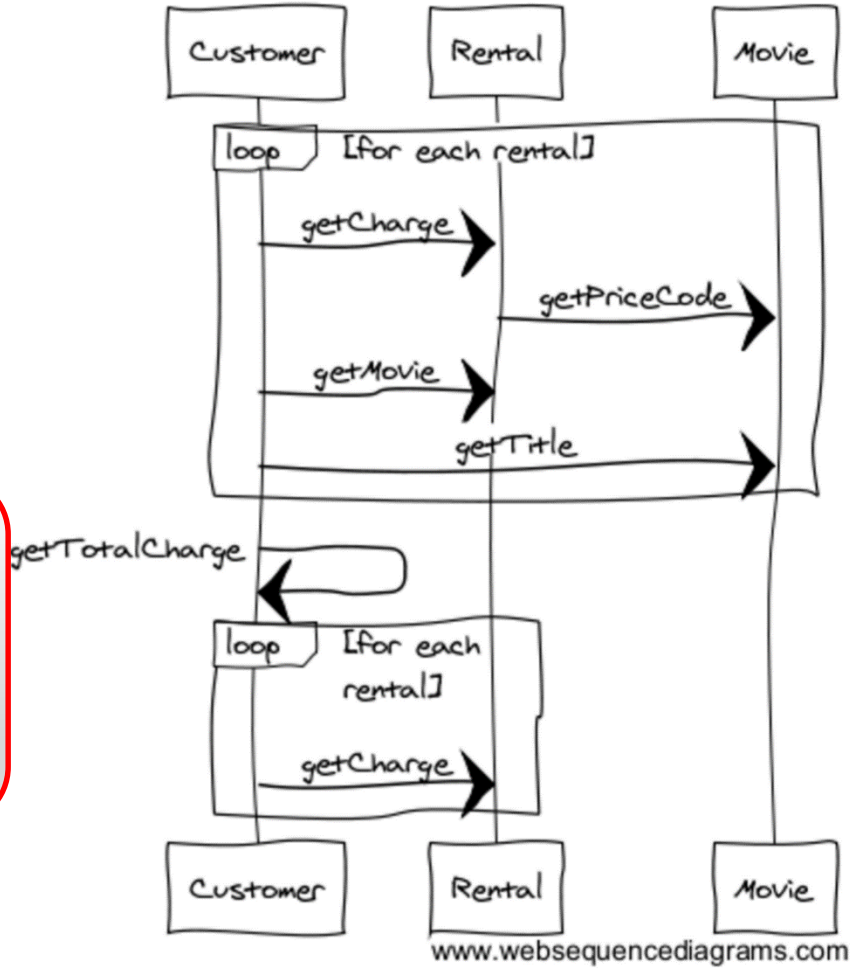
Etape 3 : Isoler le calcul de la somme des frais de toutes les locations de l'affichage du relevé



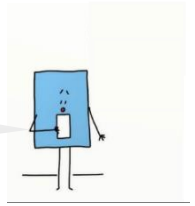
Evolution du diagramme de séquences



Etape 3 : Isoler
le calcul de la somme
des frais de toutes
les locations
de l'affichage du relevé



Les calculs des frais et l'affichage sont désormais isolés …
La nouvelle fonctionnalité peut être ajoutée ! …



Isabelle BLASQUEZ

Ajout nouvelle fonctionnalité

Etape 4 :

Mettre en place le nouvel affichage au format HTML

via la méthode `htmlStatement`

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step4_Ajout_htmlStatement.md

Isabelle BLASQUEZ

Copier-Coller `statement` en `htmlStatement` et adapter le formatage à du HTML

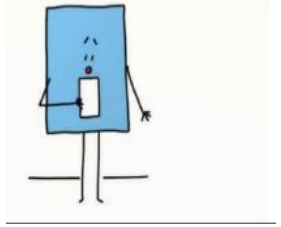
```
public class Customer {
// …

public String htmlStatement() {

    String result = "<h1>Rental Record for <em>" + this.getName()
                                                + "</em></h1><p>\n";

    result +="<table><tboby>";

     for (Rental each : rentals) {
        result +=  "<tr><td>"+each.getMovie().getTitle()
                      + "</td><td>" + each.getCharge() + "</tr></td>";
    }
    result +="</table></tboby>";

    result += "<p>Amount owed is <em>" + getTotalCharge()
                                                    + "</em><p>\n";
   return result;
}

}
```

Rental Record for *Alice*

The Lord of the Rings	3.5

Amount owed is *3.5*

Et, si on couvrait ce nouveau comportement par des tests ….



Nouveau test pour couvrir `htmlStatement`

```java
public class CustomerTest {
// …

    @Test
    public void htmlStatementForOneRegularMovie() {
        Customer customer = new Customer("Alice");
        Movie movie = new Movie("The Lord of the Rings", Movie.REGULAR);
        Rental rental = new Rental(movie, 3); // 3 day rental
        customer.addRental(rental);

        String htmlStatement = customer.htmlStatement();

        String expected = "<h1>Rental Record for <em>Alice</em></h1><p>\n"
                + "<table><tboby><tr><td>The Lord of the Rings</td><td>3.5</tr></td></table></tboby>"
                + "<p>Amount owed is <em>3.5</em><p>\n";
          assertEquals(expected, htmlStatement);
     }
}
```

ARRANGE

ACT

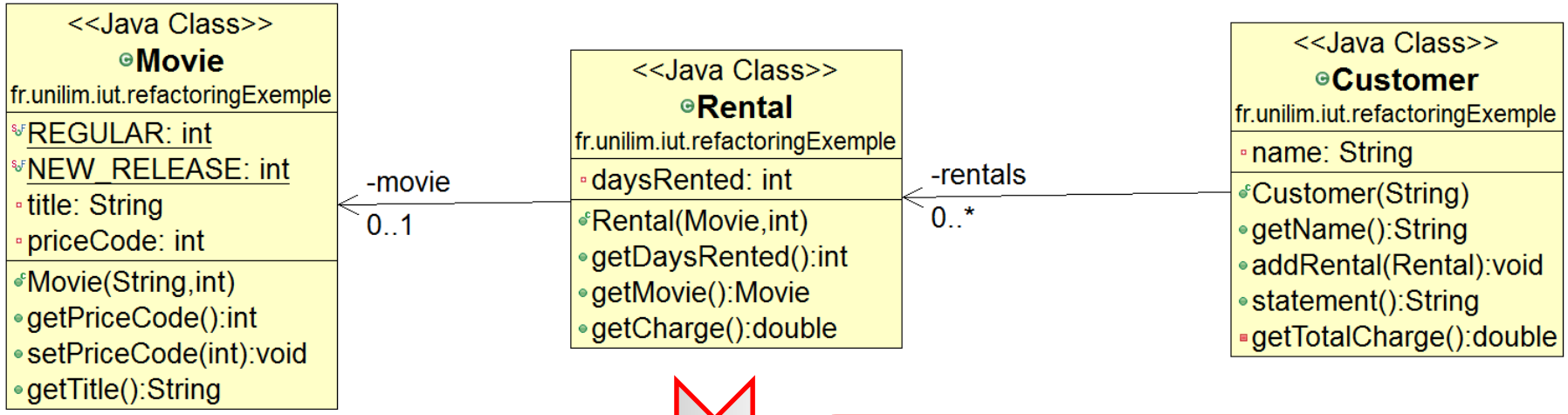
ASSERT

Rental Record for *Alice*

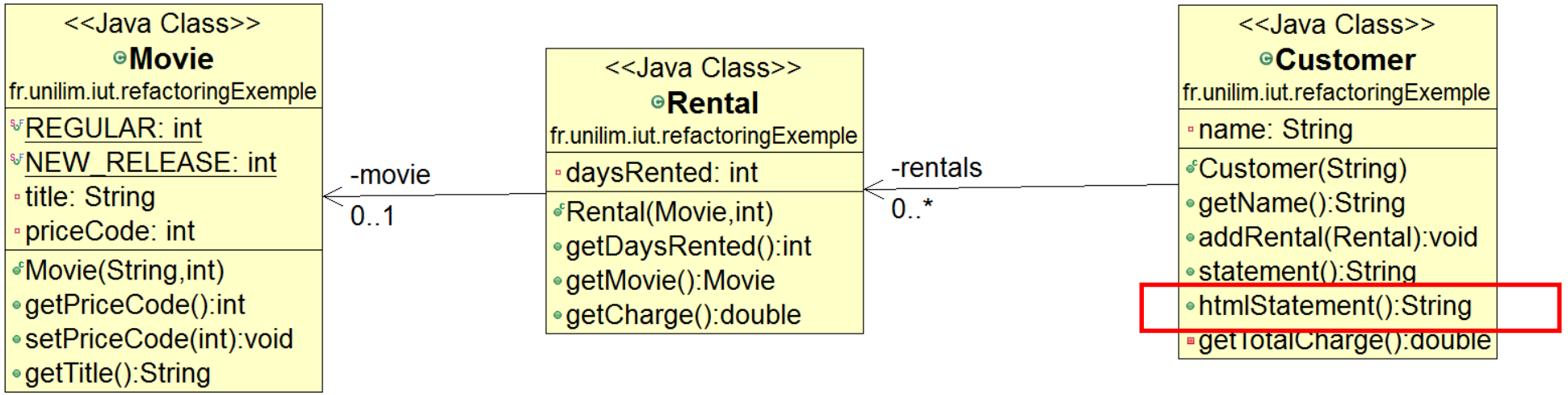
The Lord of the Rings	3.5

Amount owed is *3.5*

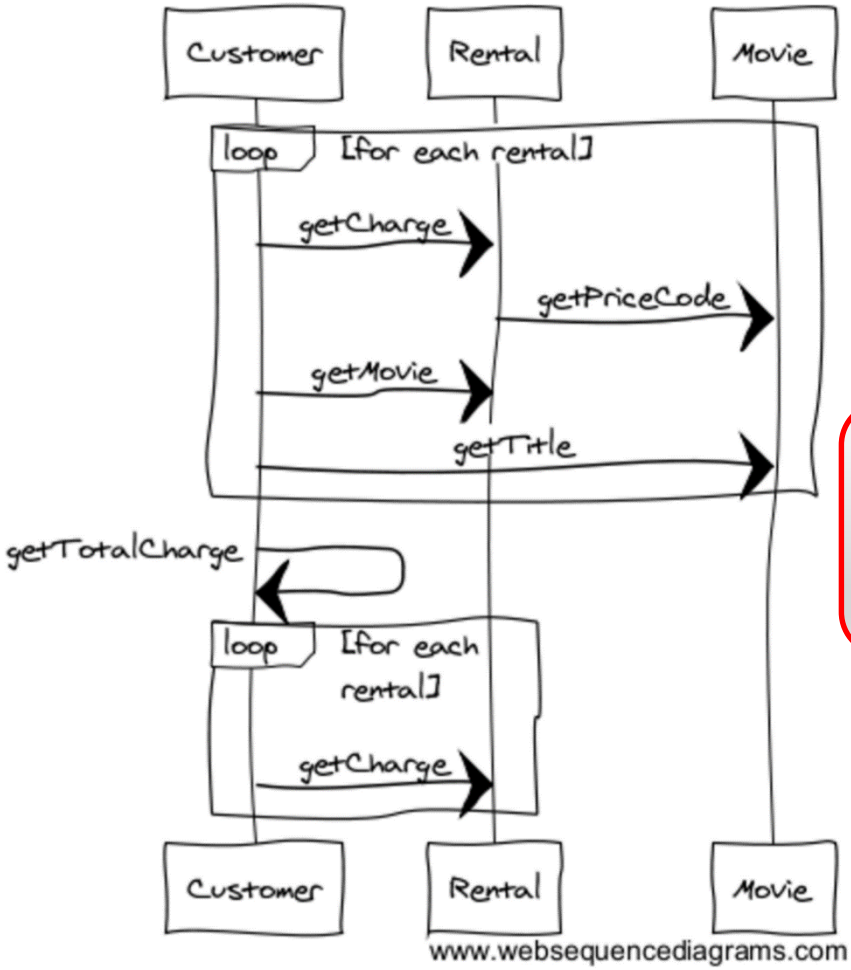
Evolution du diagramme de classes



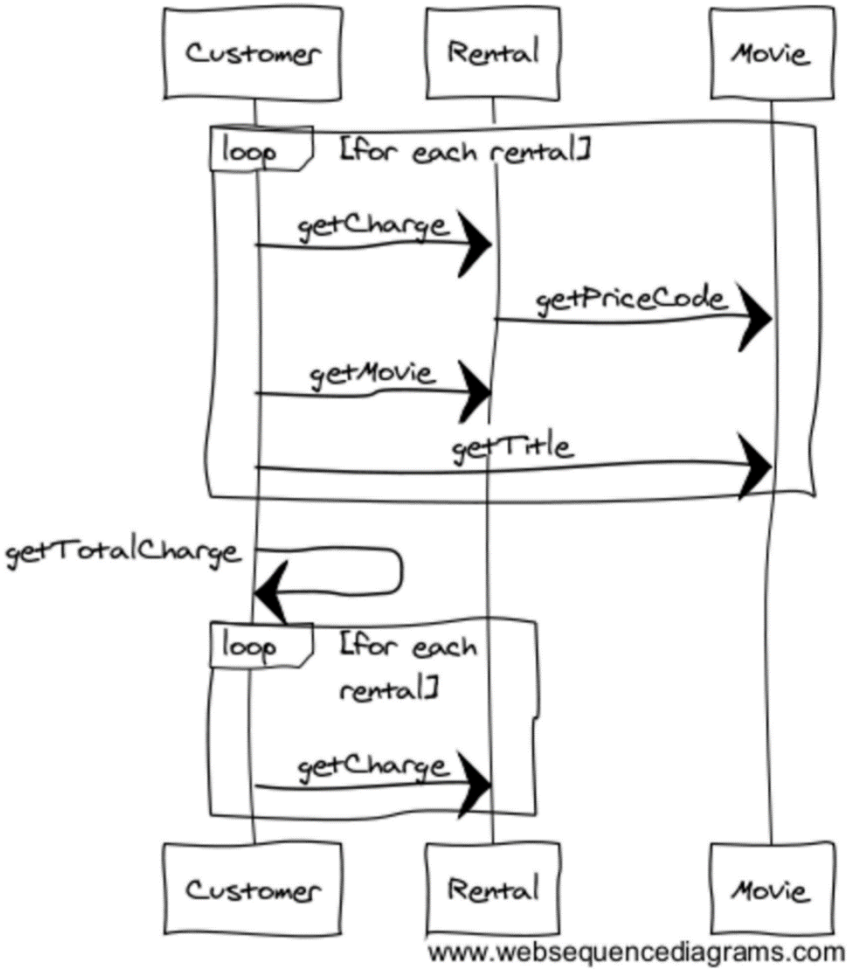
Etape 4 : Mettre en place le nouvel affichage au format HTML



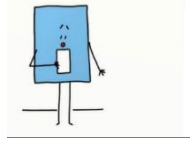
Evolution du diagramme de séquences



Etape 4 :
Mettre en place le nouvel affichage au format HTML



Ajout du `htmlStatement` donc rien ne change pour `statement` ...



Isabelle BLASQUEZ

Un nouveau besoin ….

Le client a un nouveau besoin…

Il souhaite pouvoir changer, dans un futur proche, la façon dont il classifie ses vidéos mais il ne connaît pas exactement ce nouveau classement (plus de deux types c'est sûr…), ni le nouveau système de tarification qu'il adoptera …

Besoin d'un refactoring pour mettre en place ce nouveau besoin ?

Même si lancer le calcul des frais de location semble bien être de la responsabilité de la location (Rental), (puisque dépendant de la durée de location)

ne semble-t-il pas judicieux,
pour **modifier et étendre facilement le système de tarification et l'ajout de nouveaux types de vidéos,**
que ce soit plutôt la **vidéo** qui connaisse la règle de tarification à laquelle elle est soumise ?

⇒ Hmm, Hmmm… **un refactoring semble s'imposer pour lancer le calcul des frais de location depuis une location, tout en déléguant la gestion des règles de tarification à la vidéo**

Refactoring
en vue de répondre au nouveau besoin d'un système de tarification évolutif …

Etape 5 :

Déléguer la gestion des règles de tarification à la vidéo

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step5_DeleguerGetCharge.md

Isabelle BLASQUEZ

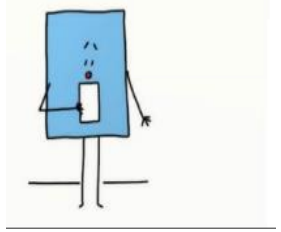
Et que nous dit le code ?

Un petit coup d'œil sur `getCharge` ...

```
public class Rental {

// …

public double getCharge() {

   double result = 0;

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

       case Movie.NEW_RELEASE:
          result += getDaysRented() * 3;
          break;
   }
   return result;
}

}
```

Loi de Demeter !!!
Effectivement `getCharge` serait plus approprié dans la classe `Movie` ...

Il va donc falloir déplacer `getCharge` vers `Movie` ...



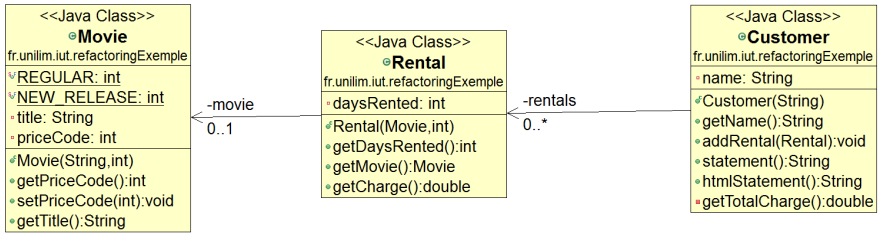
Déplacer `getCharge` vers `Movie` avec ou sans délégation ?

Où est utilisé `getCharge` ? ⇒ Uniquement dans **`Customer:`**

→ dans **`statement`** et **`htmlstatement : each.getCharge()`**

→ dans **`getTotalCharge: each.getCharge()`**

Si on déplaçait simplement `getCharge` dans `Movie`

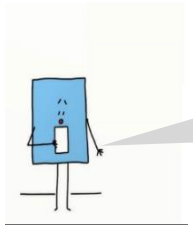


⇒ Comme **`Customer`** « ne parle pas directement » à **`Movie`** (cf diagramme de classes)

l'appel **`each.getCharge()`** dans **`Customer`** deviendrait **`each.getMovie().getCharge(each)`**

Mais, `each` (une location) à la fois message et paramètre d'entrée ?!? …

… et **Loi de Demeter !!!**



Une délégation de `getCharge` s'impose pour conserver l'appel `each.getCharge()` dans `Customer`

Utilisation du Move Methode de l'IDE pour déplacer getCharge vers Movie avec délégation ...

Movie

```
public class Movie {
// …

   public double getCharge(Rental rental) {

      double result = 0;

      switch (rental.getMovie().getPriceCode()) {

          case Movie.REGULAR:
             result += 2;
             if (rental.getDaysRented() > 2) {
               result += (rental.getDaysRented() - 2) * 1.5;
             }
          break;

         case Movie.NEW_RELEASE:
             result += rental.getDaysRented() * 3;
         break;
      }

     return result;
   }

}
```

Rental

```
public class Rental {
// …

   public double getCharge() {
      return movie.getCharge(this);
   }

}
```

Customer

```
public class Customer {
// Appel à each.getCharge()

}
```

Pourrait-on encore améliorer ce refactoring ?

notamment peut-être le

```
switch (rental.getMovie().getPriceCode())
```

Amélioration du refactoring pour la classe Movie

```
switch (rental.getMovie().getPriceCode())
```

renvoie une **Movie** …

```
switch (this.getPriceCode())
```

Movie

```
public class Movie {
// …

   public double getCharge(Rental rental) {

      double result = 0;

      switch (this.getPriceCode()) {

           case Movie.REGULAR:
              result += 2;
              if (rental.getDaysRented() > 2) {
                result += (rental.getDaysRented() - 2) * 1.5;
              }
           break;

          case Movie.NEW_RELEASE:
              result += rental.getDaysRented() * 3;
          break;
      }

     return result;
   }

}
```

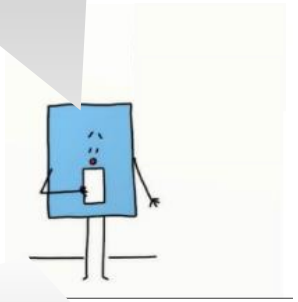
Zoom sur `getCharge` : une autre amélioration de conception possible ?

Movie

```
public class Movie {
// …

   public double getCharge(Rental rental) {

      double result = 0;

      switch (this.getPriceCode()) {

          case Movie.REGULAR:
             result += 2;
             if (rental.getDaysRented() > 2) {
               result += (rental.getDaysRented() - 2) * 1.5;
             }
          break;

         case Movie.NEW_RELEASE:
             result += rental.getDaysRented() * 3;
         break;
      }

     return result;
   }

}
```

Est-il vraiment nécessaire de passer un objet de la classe `Rental`,

Alors que seul un entier correspondant au nombre de jours de location nous intéresse ?



Nous choisissons donc de découpler la classe **`Rental`** de la classe **`Movie`**

Découpler la classe Rental de Movie

Différentes étapes pour mener ce refactoring

`rental.getDaysRented()`

Extract Variable

`daysRented`

Passer `daysRented` **comme paramètre d'entrée**

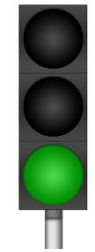
Le renommer éventuellement `numberOfDaysRented`

Movie

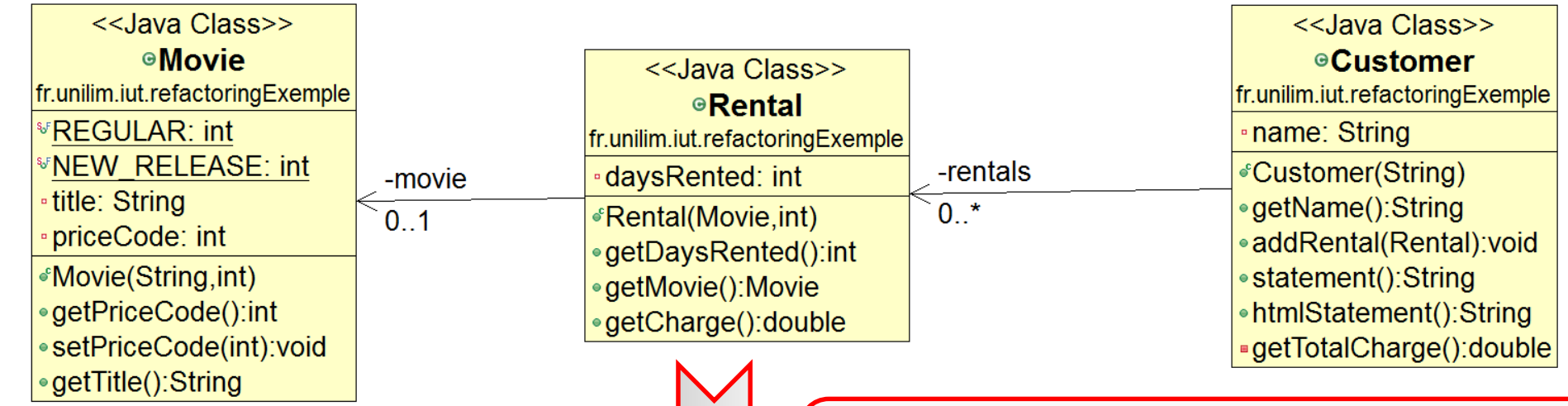
```
public class Movie {
// …
    public double getCharge(int numberOfDaysRented) {

        double result = 0;

        switch (this.getPriceCode()) {

            case Movie.REGULAR:
                result += 2;
                if (numberOfDaysRented > 2) {
                  result += (numberOfDaysRented - 2) * 1.5;
                }
            break;

            case Movie.NEW_RELEASE:
                result += numberOfDaysRented * 3;
            break;
        }
      return result;
    }
}
```

Rental

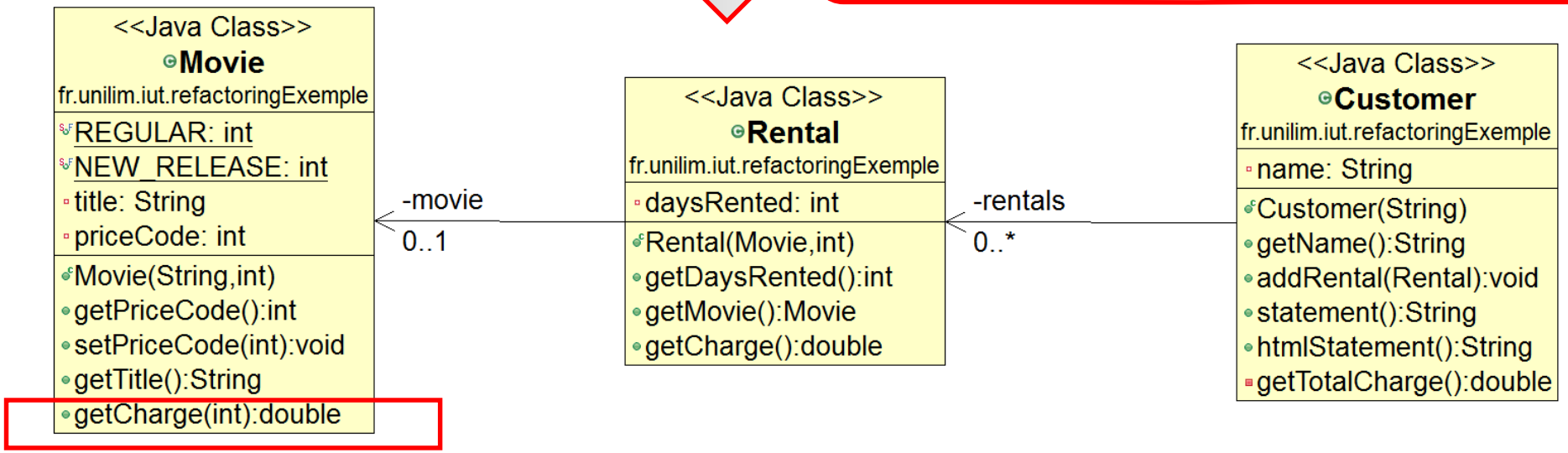
```
public class Rental {
private Movie movie;
private int daysRented;
// …

    public double getCharge() {
        return movie.getCharge(daysRented);
    }
}
```



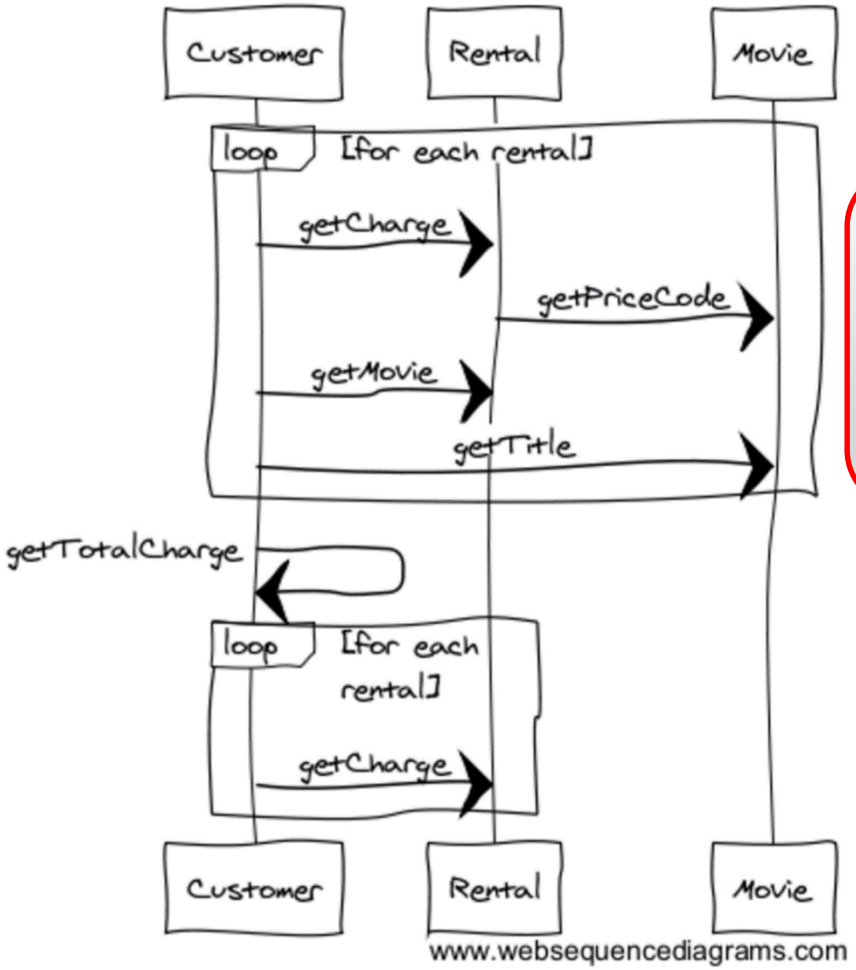
Evolution du diagramme de classes



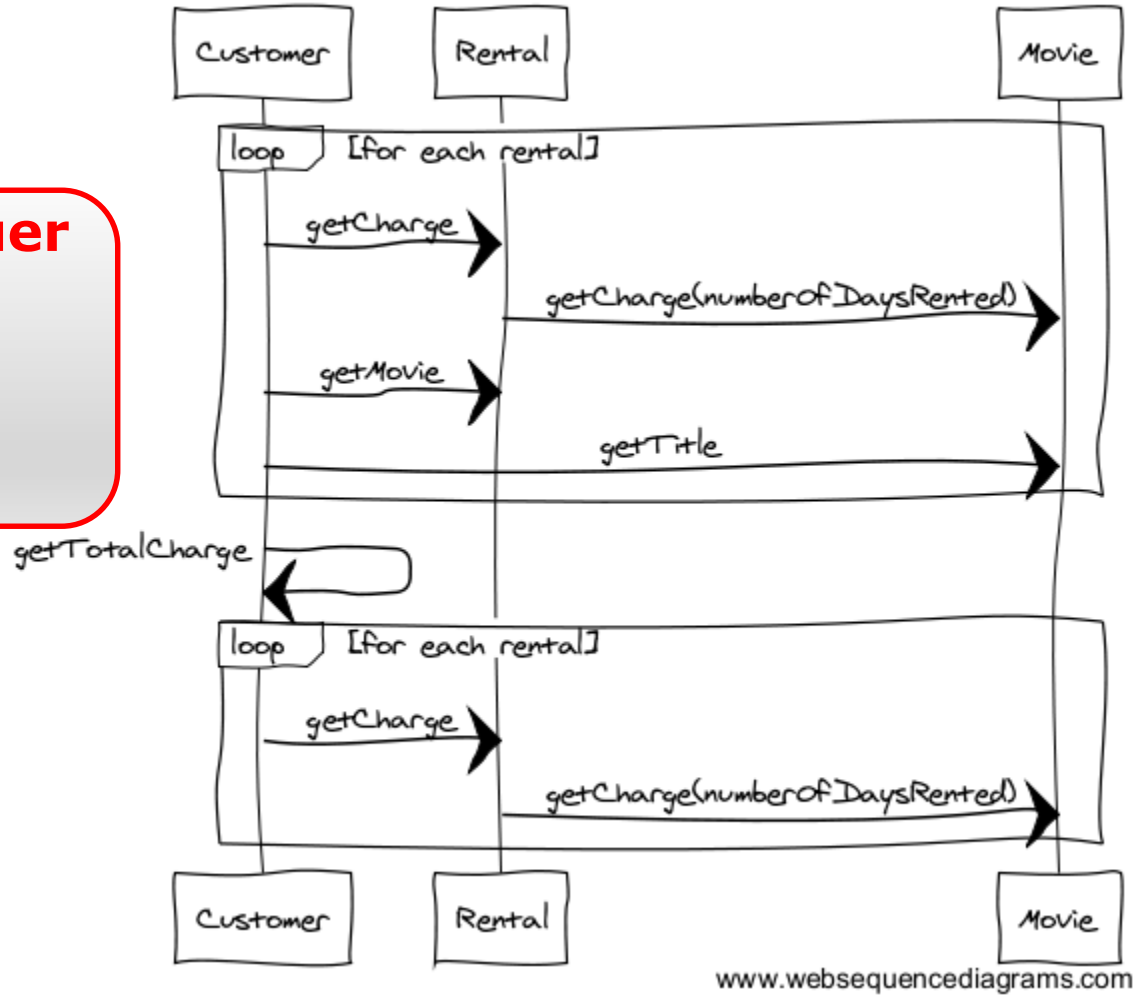
Etape 5 : Déléguer la gestion des règles de tarification à la vidéo



Evolution du diagramme de séquences (statement)



Etape 5 : Déléguer la gestion des règles de tarification à la vidéo



Refactoring
en vue de répondre rapidement
à un nouveau besoin

Etape 6 :

Mettre en place un système de tarification évolutif et modulable grâce à un *State Pattern*

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/refactoring_Step6_StatePattern.md

Isabelle BLASQUEZ

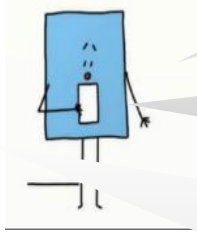
Actuellement …

<<Java Class>>
Movie
fr.unilim.iut.refactoringExemple

- REGULAR: int
- NEW_RELEASE: int
- title: String
- priceCode: int

- Movie(String,int)
- getPriceCode():int
- setPriceCode(int):void
- getTitle():String
- getCharge(int):double

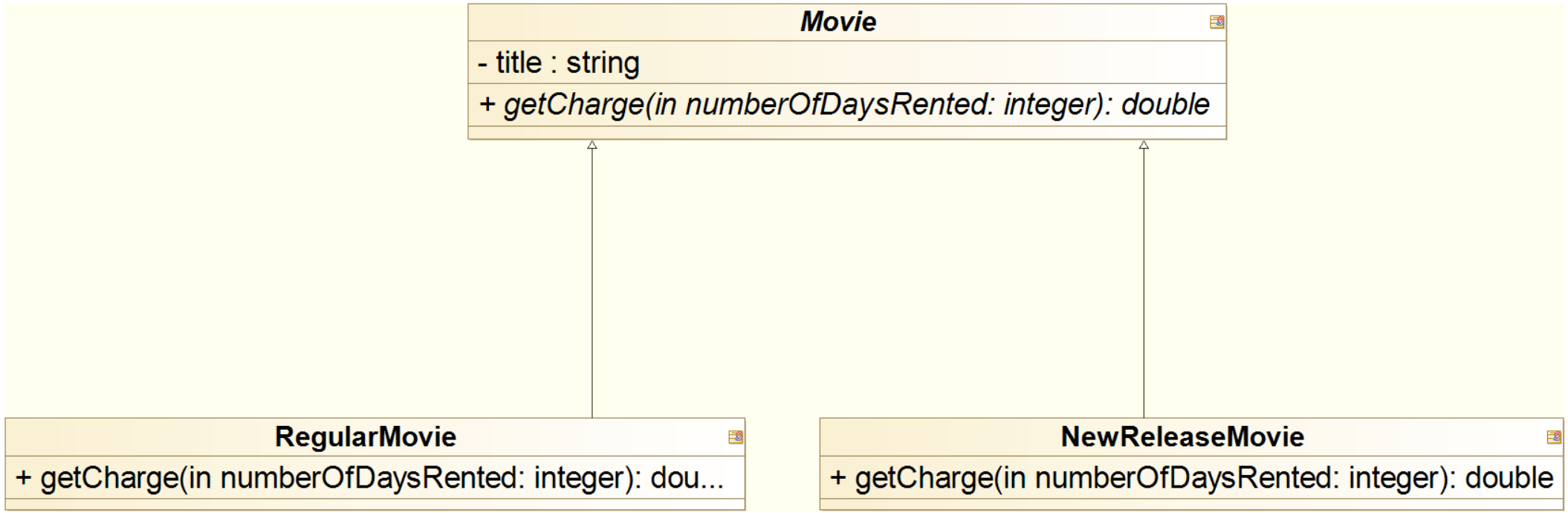
La classe `Movie` traite et centralise actuellement toutes les règles de calculs des différents tarifs de location proposant ainsi plusieurs réponses possibles à une même question (`getCharge`).

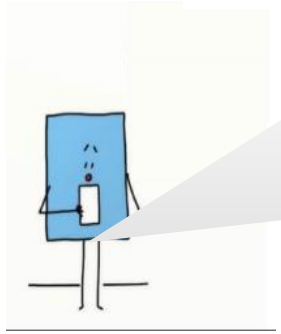


Hmm… L'ajout d'un nouveau tarif demande pour l'instant une modification de `Movie` et notamment l'ajout de lignes de code supplémentaires dans `getCharge`

Et si on essayait de faire en sorte que le tarif soit lui-même le seul garant (responsable) de sa propre règle de calcul ?
⇒ Une classe par tarif serait alors nécessaire
… et ajouter un nouveau tarif reviendrait simplement à ajouter une nouvelle classe
La mise en place d'un héritage s'impose !!!!

Une première proposition d'héritage simple directement sur `Movie`

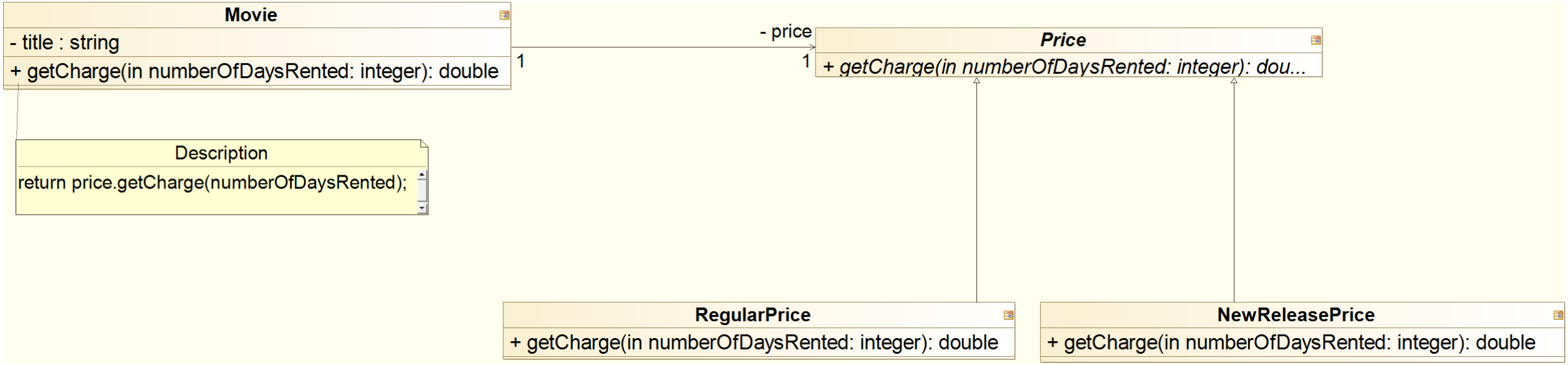


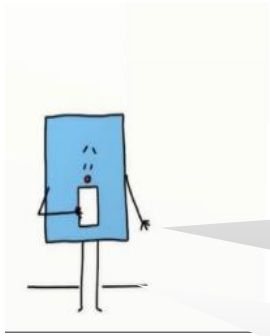


Les vidéos doivent pouvoir changer de tarifs au cours de leur vie :
Actuellement, elles passent de ***nouveautés*** à ***ordinaire***
au bout d'un certain temps

⇒ **Cet héritage n'est donc pas satisfaisant pour notre application !!!**

Un héritage à l'aide d'un patron d'Etat (`State pattern`)





Une vidéo a un *tarif* particulier et ses frais de location dépendent de ce *tarif*.

Le patron d'Etat (State Pattern) permet à un objet de modifier son comportement lorsque son état interne change.

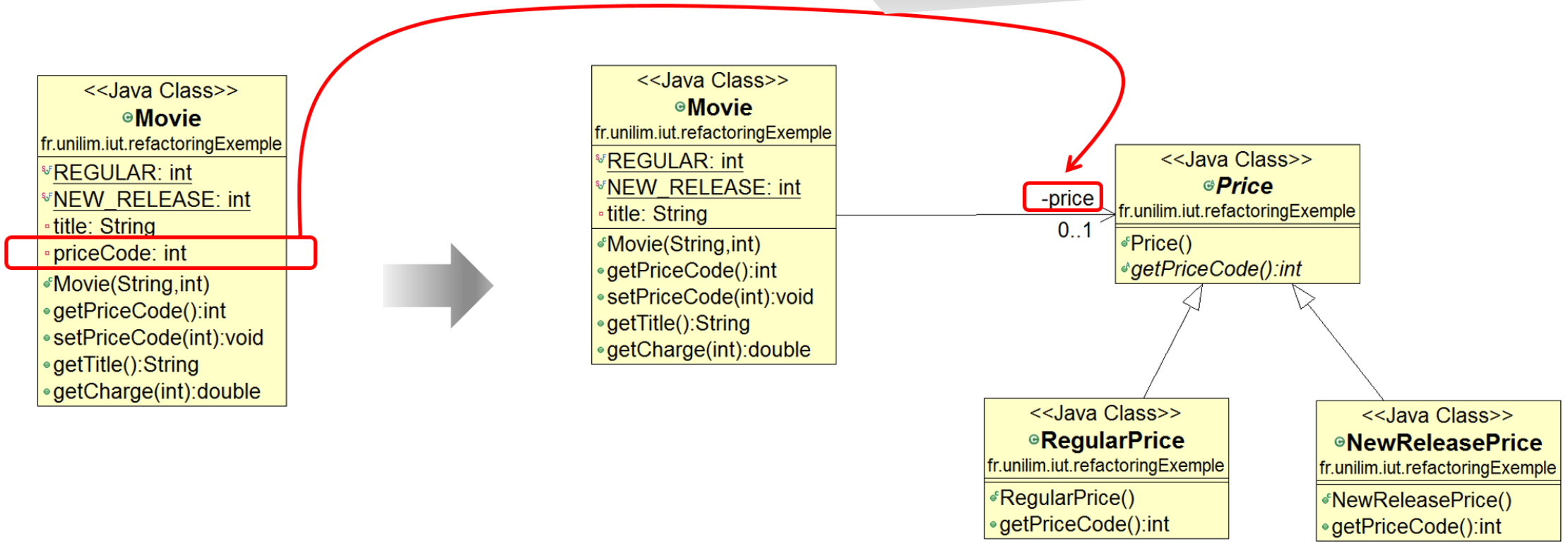
Système de tarification et facilement modulable car …

→ Ajouter un nouveau tarif de vidéo ⇒ ajouter une classe fille à `Price`

→ Modifier l'état (tarif) de la vidéo ⇒ possible à tout moment via un `setPrice` dans `Movie`.

Mise en place du nouveau système de tarification à base d'états via le refactoring `Replace Type Code with State/Strategy`

2. Transformer le tarif du statut de *simple attribut* (`int`) au statut d'*état* pour la vidéo (`Price`)



1. Créer un héritage de tarifs

Isabelle BLASQUEZ

Création d'un système de tarification via un héritage de tarifs

Price

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

RegularPrice

```
public class RegularPrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.REGULAR;
    }
}
```

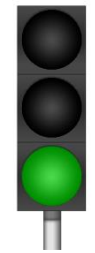
NewReleasePrice

```
public class NewReleasePrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.NEW_RELEASE;
    }
}
```

Associer un tarif à une vidéo (coupler Price à Movie)

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

   private String title;
   private Price price;


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

   public int getPriceCode() {
      return price.getPriceCode();
   }

   public void setPriceCode(int priceCode) {
       switch (priceCode) {
          case REGULAR:
             price = new RegularPrice();
             break;
          case NEW_RELEASE:
             price = new NewReleasePrice();
             break;
          default:
             throw new
IllegalArgumentException("Incorrect Price Code");
       }
}
```

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


   public double getCharge(int numberOfDaysRented) {

       double result = 0;

          switch (this.getPriceCode()) {
              case Movie.REGULAR:
                  result += 2;
                  if (numberOfDaysRented > 2) {
                     result +=
                     (numberOfDaysRented - 2) * 1.5;
                   }
                  break;
               case Movie.NEW_RELEASE:
                  result += numberOfDaysRented * 3;
                  break;
               }

       return result;
      }
}
```



Movie

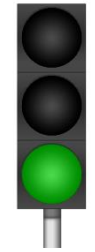
Déléguer la règle de gestion de calcul des frais de location au système de tarification (`Price`)...

Movie

```
public class Movie {
// …

public double getCharge(int numberOfDaysRented) {
   return price.getCharge(numberOfDaysRented);
}
```

Price

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

    public double getCharge(int numberOfDaysRented) {
       double result = 0;

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

        return result;
    }
}
```



Déplacer la méthode `getCharge` avec délégation

(pour ne pas modifier le code existant appelant déjà `getCharge` sur un objet de type Price)

… Et faire en sorte que que chaque tarif soit lui-même responsable de sa propre règle de calcul !

Price

```
public abstract class Price {
    public abstract int getPriceCode();
    public abstract double getCharge(int numberOfDaysRented);
}
```

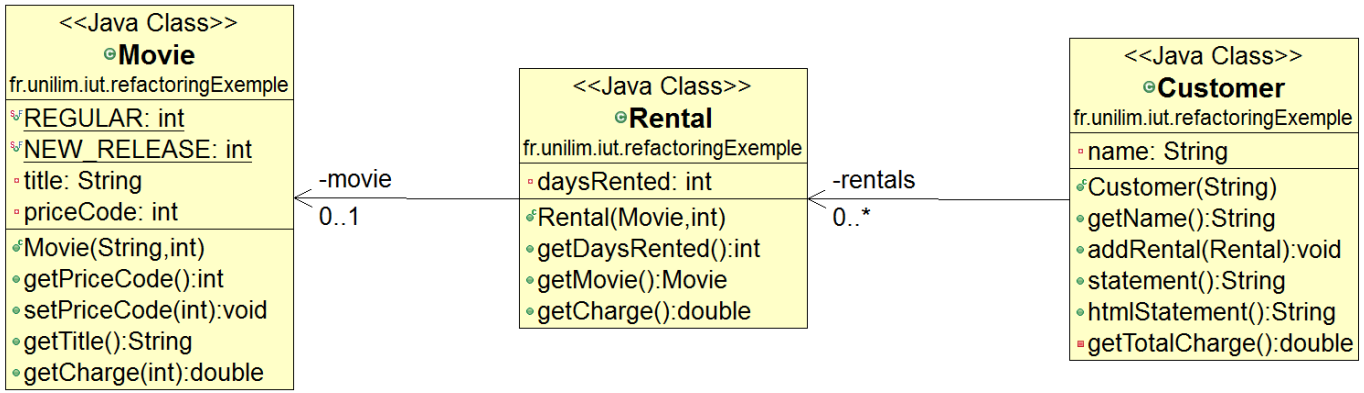
RegularPrice

```
public class RegularPrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.REGULAR;
    }

    @Override
    public double getCharge(int numberOfDaysRented) {
        double result = 2;
        if (numberOfDaysRented > 2) {
            result += (numberOfDaysRented - 2) * 1.5;
        }
        return result;
    }
}
```

NewReleasePrice

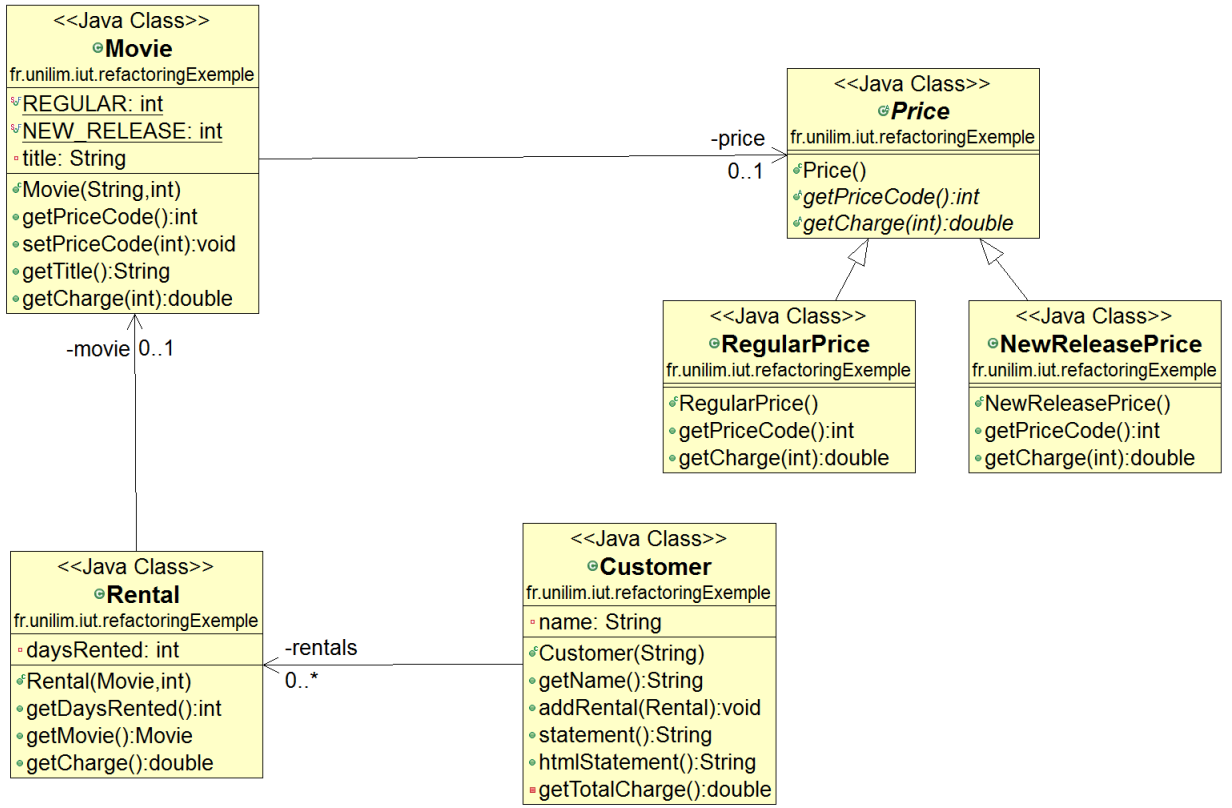
```
public class NewReleasePrice extends Price {
    @Override
    public int getPriceCode() {
        return Movie.NEW_RELEASE;
    }

    @Override
    public double getCharge(int numberOfDaysRented) {
        return numberOfDaysRented * 3;
    }
}
```

Evolution du diagramme de classes

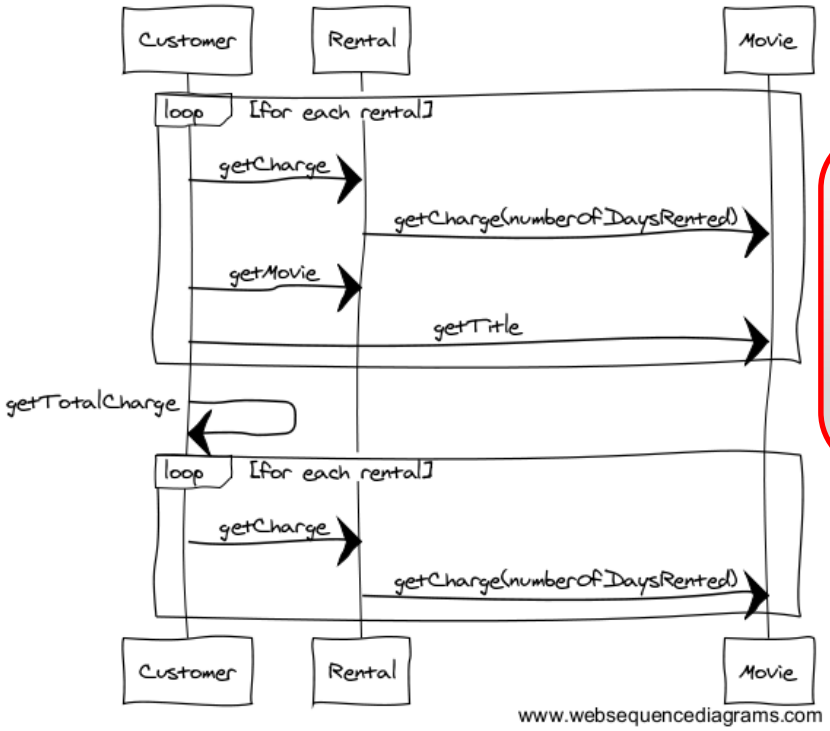


Tout semble prêt pour ajouter/modifier de nouveaux tarifs

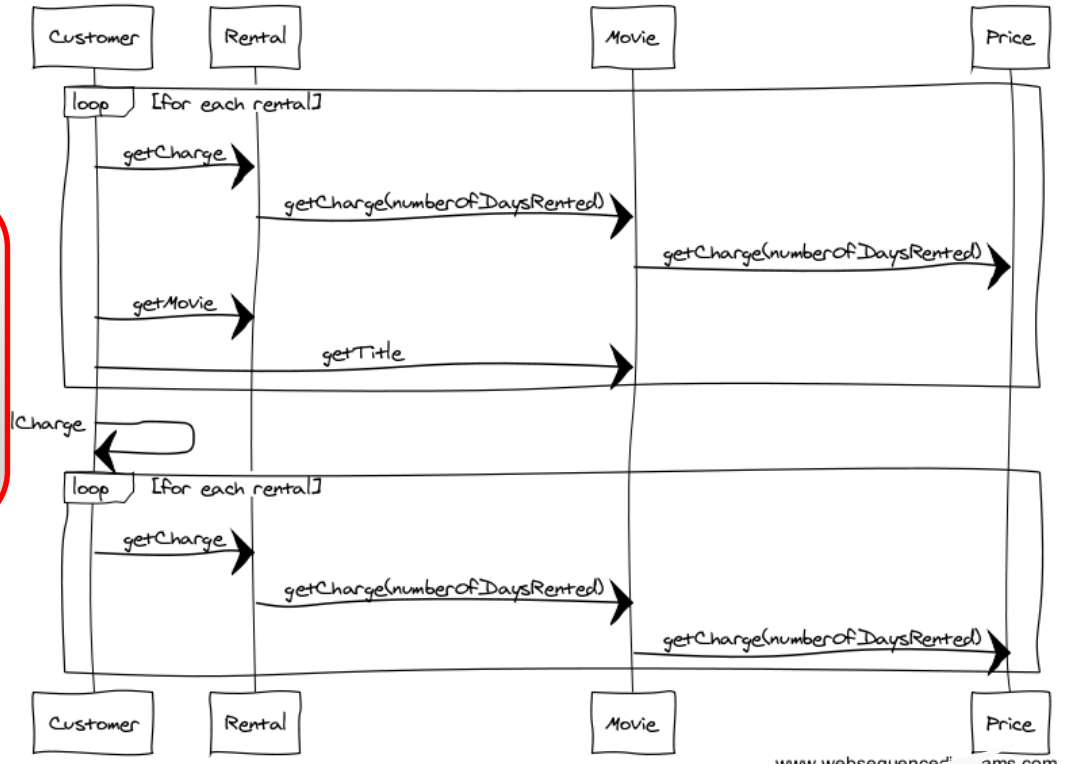
Etape 6 : Mettre en place un système de tarification évolutif et modulable



Evolution du diagramme de séquences (statement)



Etape 6 : Mettre en place un système de tarification évolutif et modulable

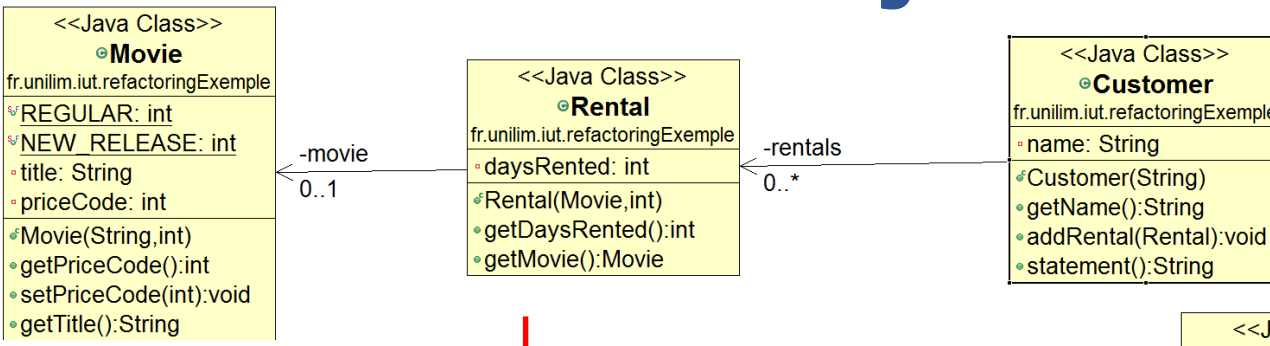


Le tarif est désormais responsable de la règle de calcul des frais de location …

Isabelle BLASQUEZ

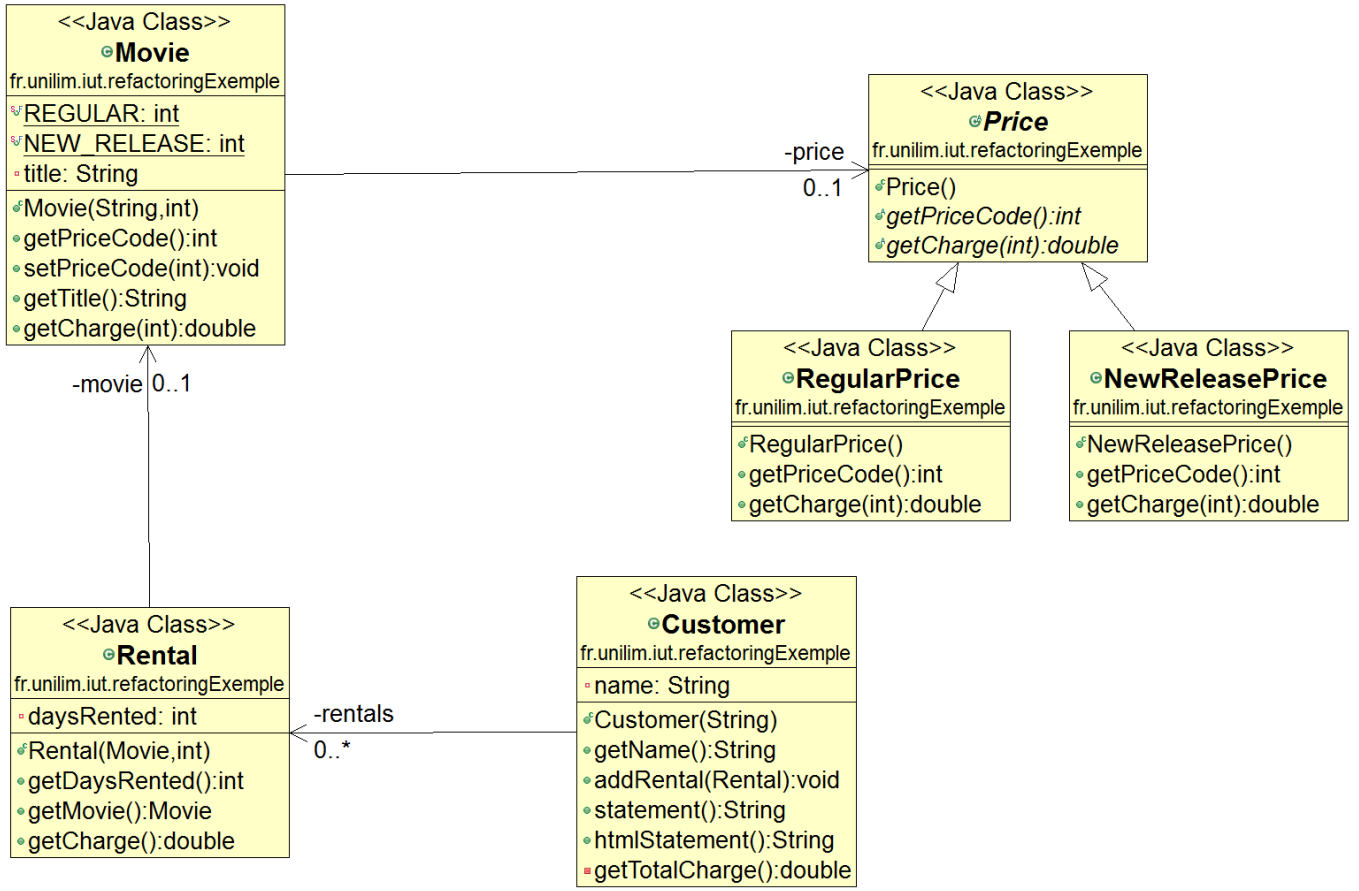
Comparaison de la conception en début et fin de refactoring

Evolution de la conception suite au refactoring et à l'ajout des nouvelles fonctionnalités



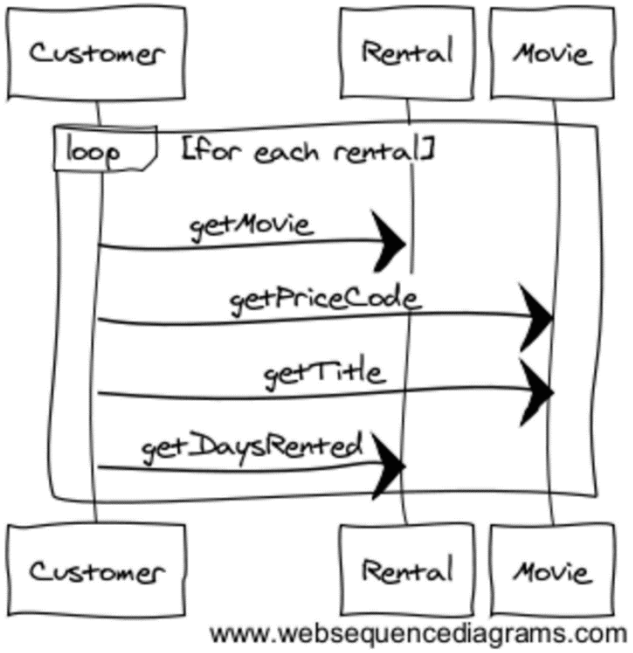
Du format simple ASCII…

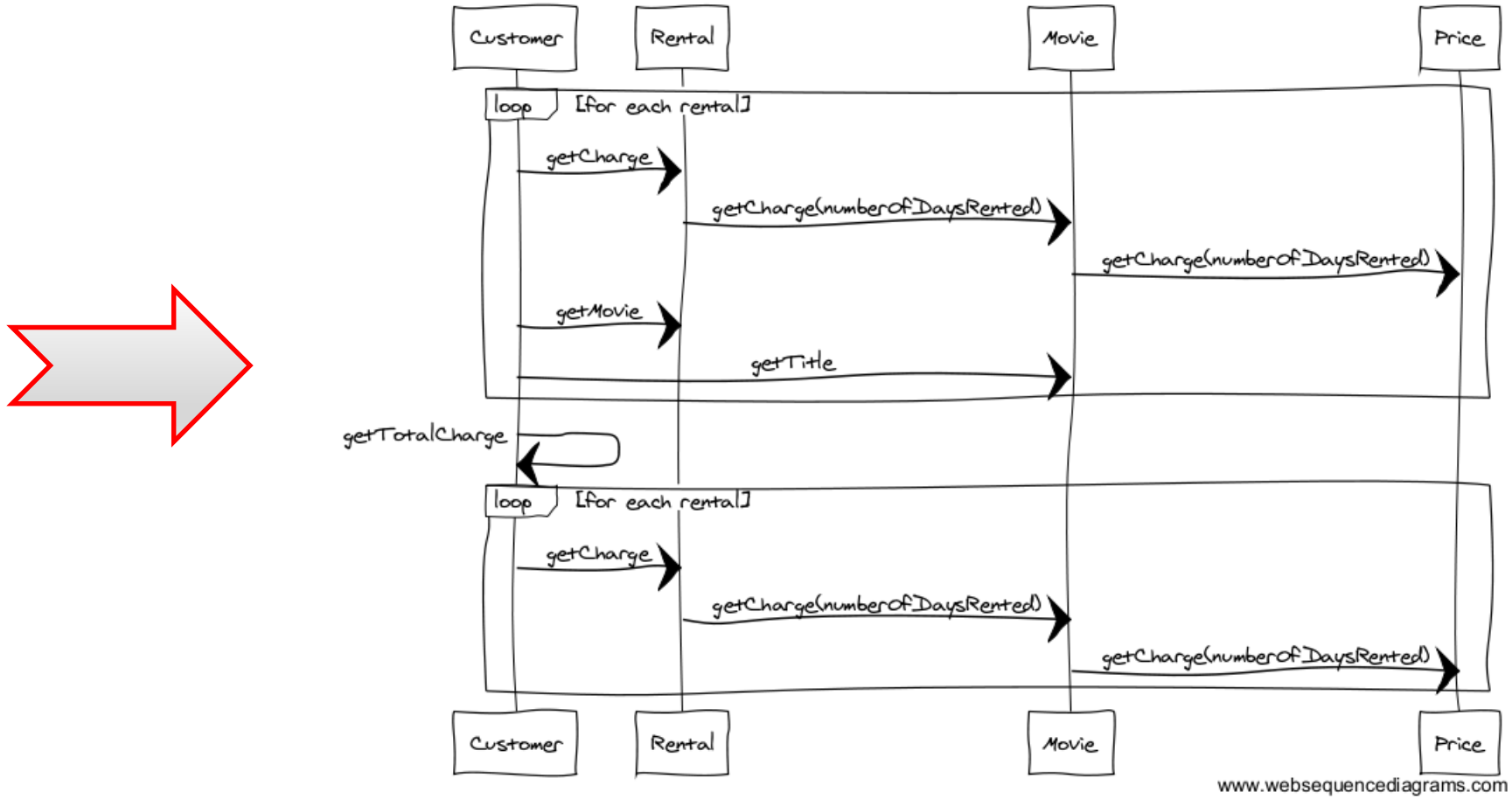
… à la prise en compte du format HTML et à la mise en place d'un système de tarification évolutif …



Isabelle BLASQUEZ

Evolution du diagramme de séquences (statement)
suite au refactoring et à l'ajout des nouvelles fonctionnalités





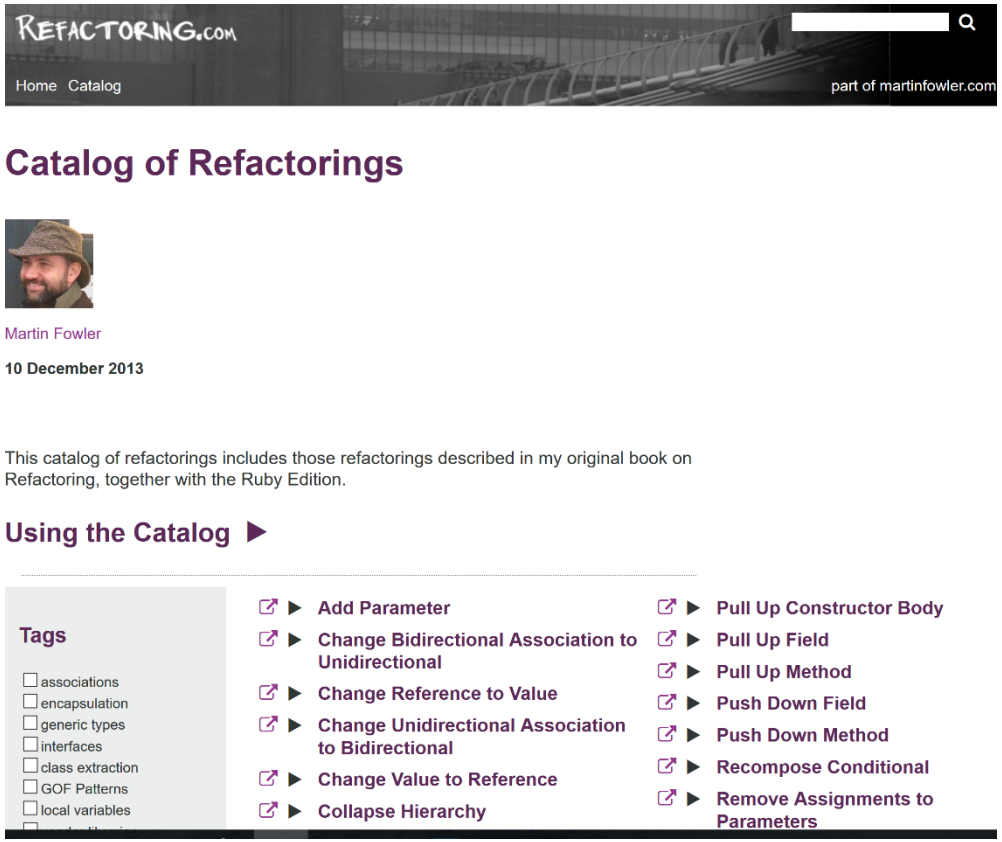
Quelques références

autour

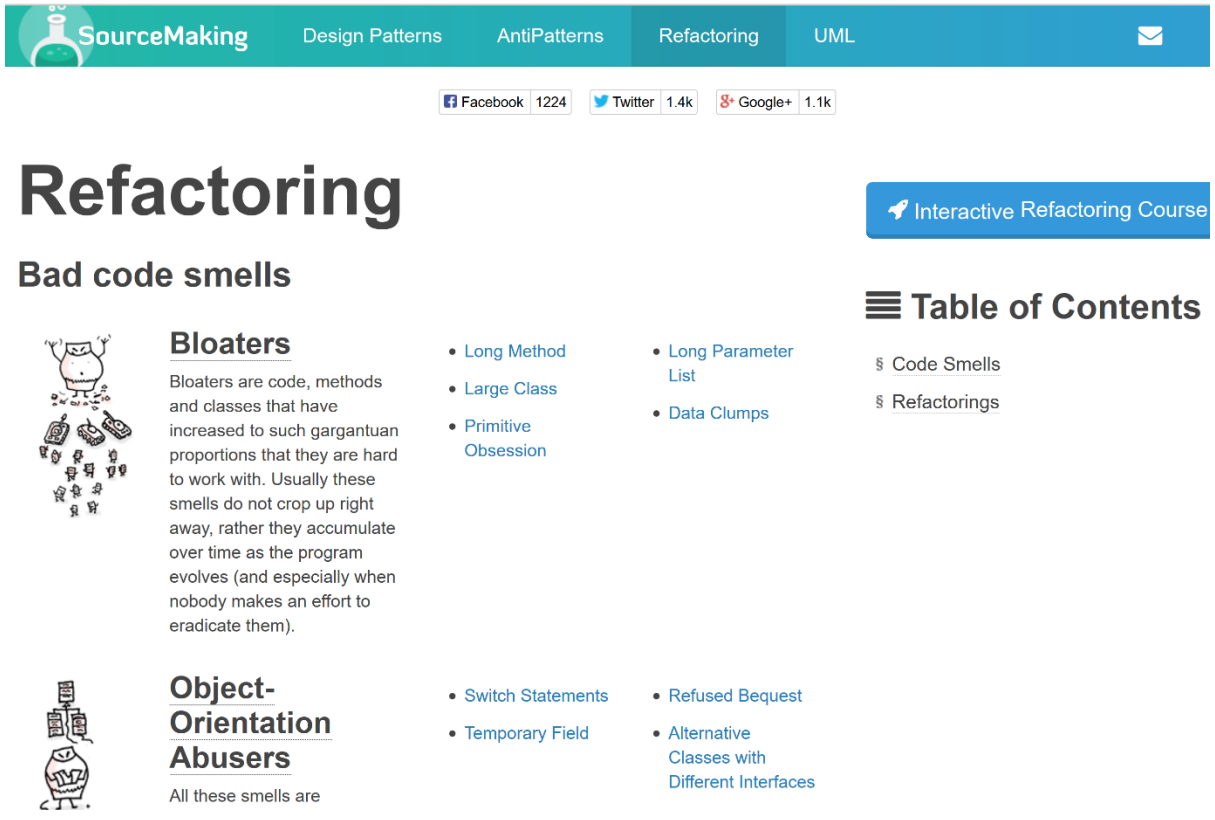
du refactoring ...

Tutoriel détaillé sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/references.md

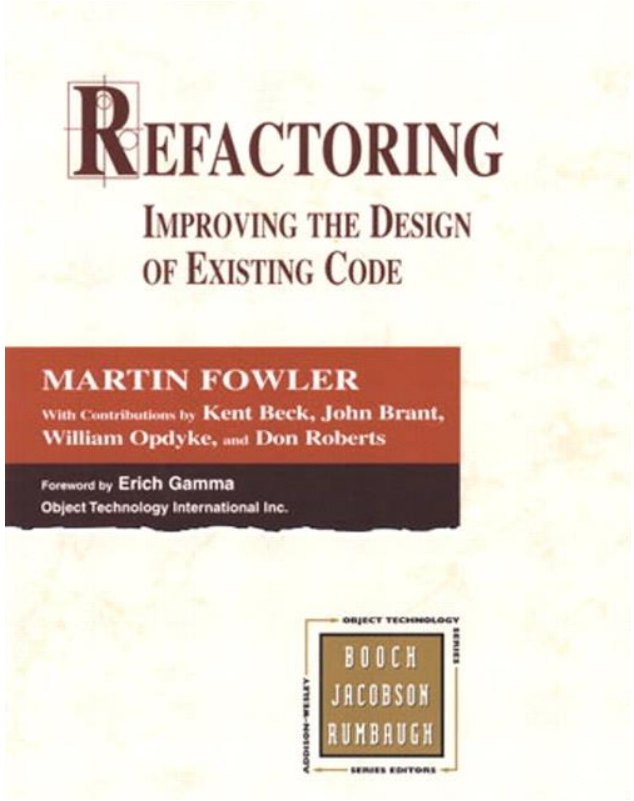
Catalogues de refactoring en ligne …

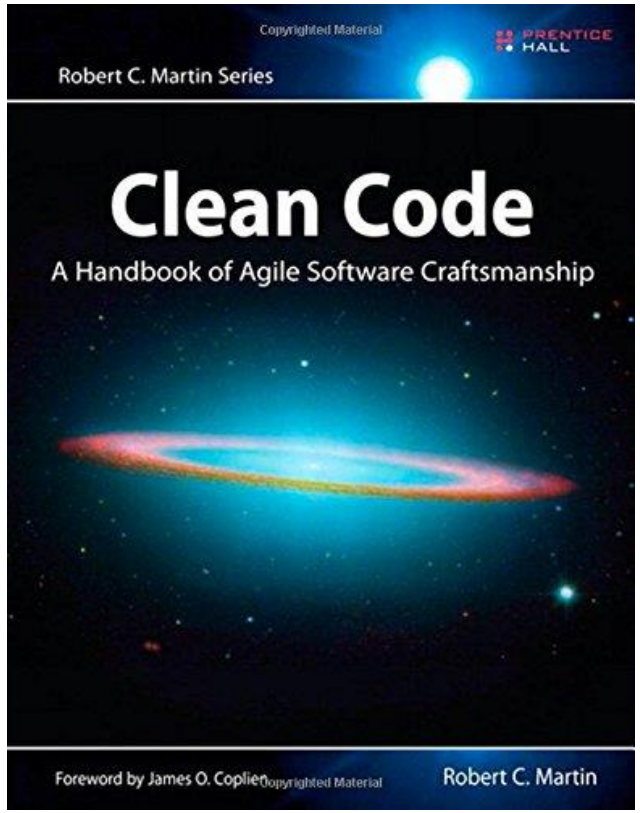


http://refactoring.com/catalog/



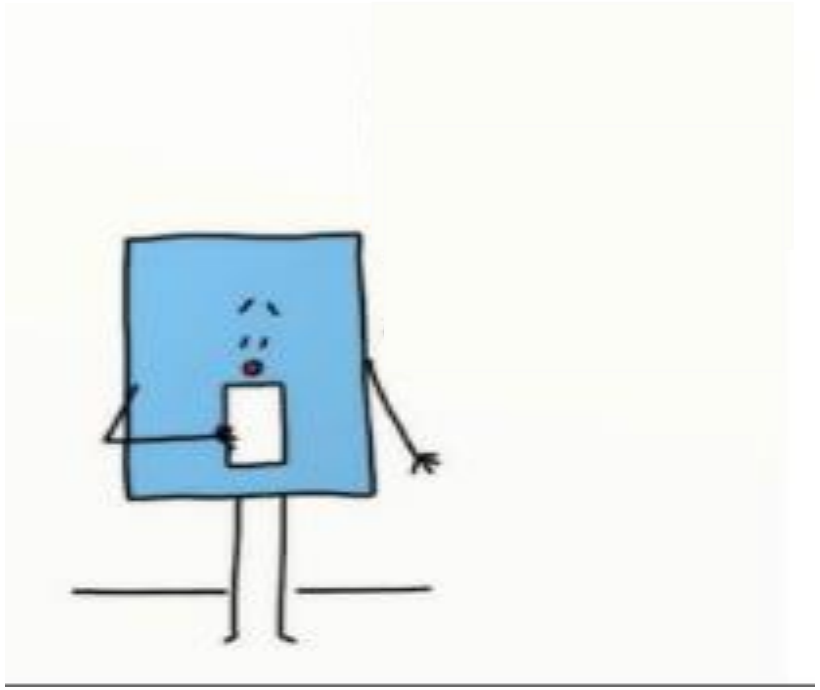
https://sourcemaking.com/



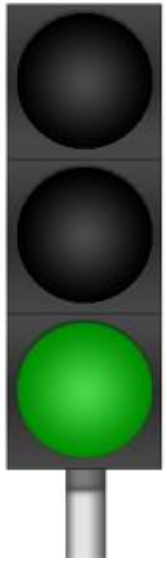


D’autres liens sur :
https://github.com/iblasquez/Refactoring_PremierExempleFowler/blob/master/references.md

Crédits Images



https://uneviededev.com/



http://www.passetoncode.fr