Introduction to Refactoring

Sutee Sudprasert

Credits

- Refactoring : Improving the design of existing code - Martin Fowler
- Design Patterns - GOF

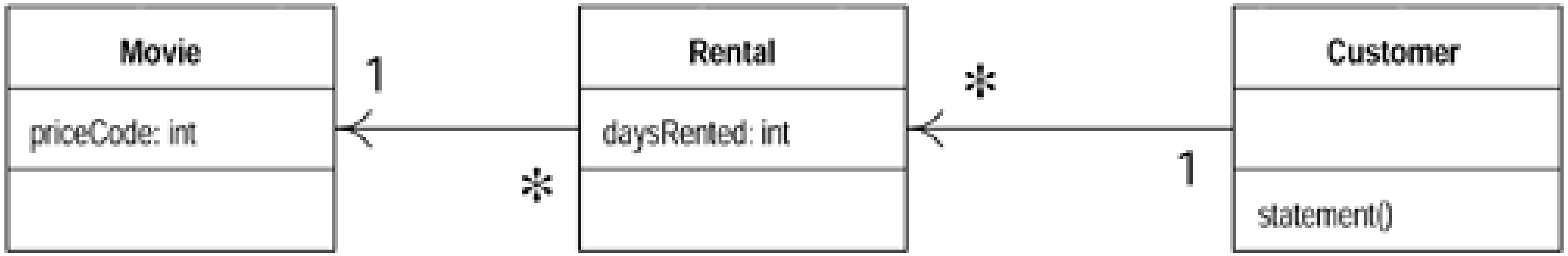
What is refactoring?

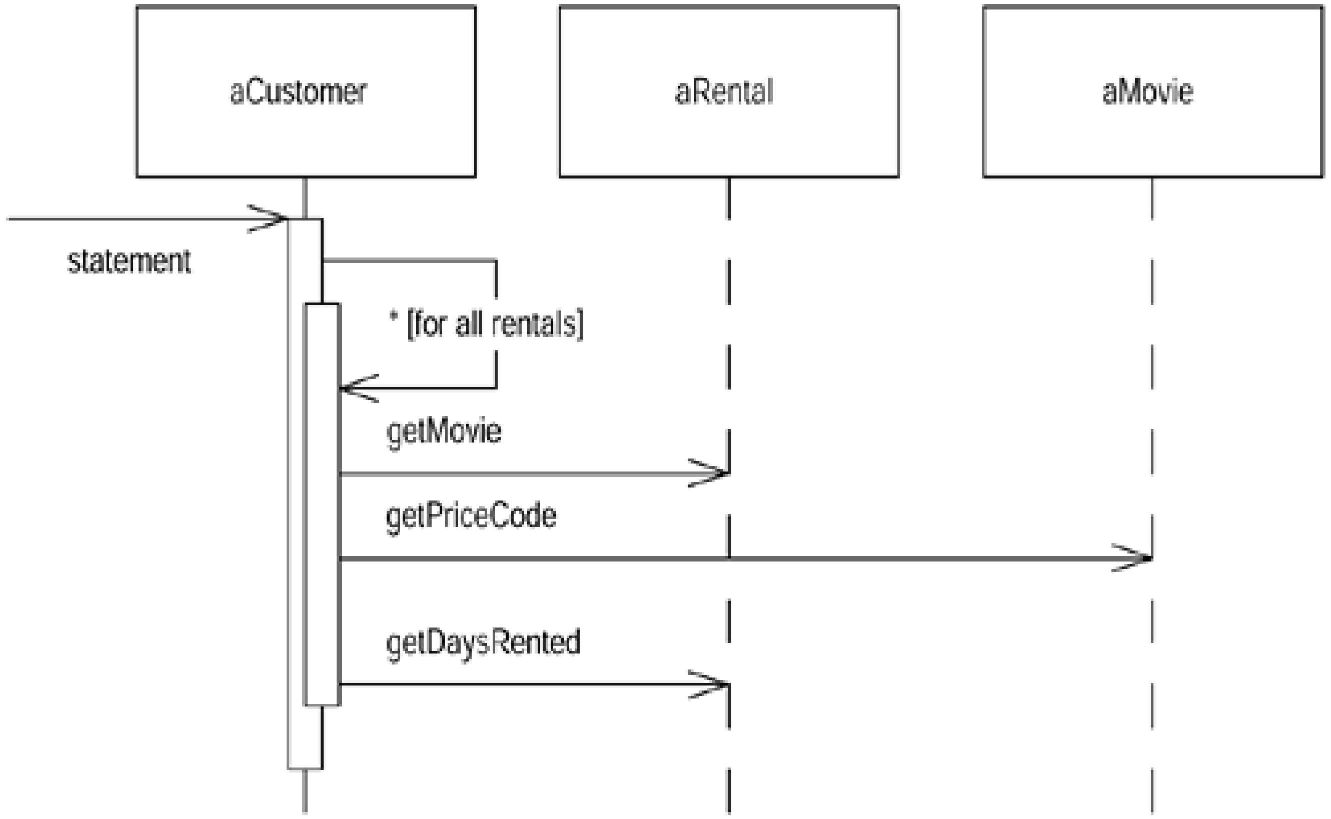
- “Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.” - Refactoring : Preface
  - You are improving the design of the code after it has been writing.
  - A good design comes first, and the coding comes second.
  - A good design may turn to bad over time the code will be modified.
- With refactoring you can take a bad design, and rework it into well-designed code.

Refactoring, a First Example (In Python)

- The sample program is a program to calculate and print a statement of a customer's charges at a video store.
  - Input: movies that a customer rented and for how long
  - Output: the charges which depend on
    - how long the movie is rented
    - identifies the type movie (regular, children's, new releases)

The starting point





The starting point

- python code

What’s wrong with this code?

- It is not well designed and certainly no object oriented.
  - There’s nothing wrong with a ***quick and dirty simple program***
  - But there are some real problems with this program if this is a representative fragment of a ***more complex system***
- The statement routine in Customer class is too long and does many of things that it does should really be done by the other classes
- A poorly designed system is hard to change because it is hard to figure out where the changes are needed (it is easy to make a mistake and introduce bugs)

What’s wrong of this code?

- Suppose the users would like a statement printed in HTML
  - it is impossible to reuse any of behavior of the current statement method
  - you can just copy the statement method and make whatever changes you need
- What happens when the charging rules change?
  - you have to fix both `statement` and `htmlStatement`
  - if you are writing a program that you don't expect to change, then cut and paste is fine.

What’s wrong of this code?

- What if the user want to make changes to the way the classify movies, but they haven’t yet decided on the change they are going to make? These changes will affect both
  - the way renters are charged for movie
  - the way that frequent renter points are calculated
- The statement method is where the changes have to be made to deal with changes in classification and charging rules
- Furthermore, as the rules grow in complexity it's going to be harder to figure out where to make the changes and harder to make them without making a mistake.

First step : Extract Method

```
# determine amount for each line
if rental.movie.price_code == Movie.REGULAR:
    this_amount += 2.0
    if rental.days_rented > 2:
        this_amount += (rental.days_rented - 2) * 1.5
elif rental.movie.price_code == Movie.NEW_RELEASE:
    this_amount += rental.days_rented * 3
elif rental.movie.price_code == Movie.CHILDRENS:
    this_amount += 1.5
    if rental.days_rented > 3:
        this_amount += (rental.days_rented - 3) * 1.5
```

⇩

```
amount_for(rental) : return this_amount
```

Second step : Rename Variables

it doesn’t make sense for this context

```
def amount_for(self, rental):
    this_amount = 0.0
    # determine amount for each line
    if rental.movie.price_code == Movie.REGULAR:
        this_amount += 2.0
        if rental.days_rented > 2:
            this_amount += (rental.days_rented - 2) * 1.5
    elif rental.movie.price_code == Movie.NEW_RELEASE:
        this_amount += rental.days_rented * 3
    elif rental.movie.price_code == Movie.CHILDRENS:
        this_amount += 1.5
        if rental.days_rented > 3:
            this_amount += (rental.days_rented - 3) * 1.5
    return this_amount
```

Second step : Rename Variables

```
def amount_for(self, rental):
    result = 0.0
    # determine amount for each line
    if rental.movie.price_code == Movie.REGULAR:
        result += 2.0
        if rental.days_rented > 2:
            result += (rental.days_rented - 2) * 1.5
    elif rental.movie.price_code == Movie.NEW_RELEASE:
        result += rental.days_rented * 3
    elif rental.movie.price_code == Movie.CHILDRENS:
        result += 1.5
        if rental.days_rented > 3:
            result += (rental.days_rented - 3) * 1.5
    return result
```

Second step : Rename Variables

Is renaming worth the effort?

```python
def amount_for(self, rental):
    result = 0.0
    # determine amount for each line
    if rental.movie.price_code == Movie.REGULAR:
        result += 2.0
        if rental.days_rented > 2:
            result += (rental.days_rented - 2) * 1.5
    elif rental.movie.price_code == Movie.NEW_RELEASE:
        result += rental.days_rented * 3
    elif rental.movie.price_code == Movie.CHILDRENS:
        result += 1.5
        if rental.days_rented > 3:
            result += (rental.days_rented - 3) * 1.5
    return result
```

Second step : Rename Variables

Is renaming worth the effort?

```
def amount_for(self, rental):
    result = 0.0
    # determine amount for each line
    if rental.movie.price_code == Movie.REGULAR:
        result += 2.0
        if rental.days_rented > 2:
            result += (rental.days_rented - 2) * 1.5
    elif rental.movie.price_code == Movie.NEW_RELEASE:
        result += rental.days_rented * 3
    elif rental.movie.price_code == Movie.CHILDRENS:
        result += 1.5
        if rental.days_rented > 3:
            result += (rental.days_rented - 3) * 1.5
    return result
```

Any fool can write code that a computer can understand.
Good programmers write code that humans can understand.

Third step : Move Method

```
class Customer(object):
...
def amount_for(self, rental):
  result = 0.0
  # determine amount for each line
  if rental.movie.price_code == Movie.REGULAR:
    result += 2.0
    if rental.days_rented > 2:
      result += (rental.days_rented - 2) * 1.5
  elif rental.movie.price_code == Movie.NEW_RELEASE:
    result += rental.days_rented * 3
  elif rental.movie.price_code == Movie.CHILDRENS:
    result += 1.5
    if rental.days_rented > 3:
      result += (rental.days_rented - 3) * 1.5
  return result
```

This method doesn’t use any data from Customer class

Third step : Move Method

```
class Rental(object):
...
def get_charge(self):
    result = 0.0
    # determine amount for each line
    if self.movie.price_code == Movie.REGULAR:
        result += 2.0
        if self.days_rented > 2:
            result += (self.days_rented - 2) * 1.5
    elif self.movie.price_code == Movie.NEW_RELEASE:
        result += self.days_rented * 3
    elif self.movie.price_code == Movie.CHILDRENS:
        result += 1.5
        if self.days_rented > 3:
            result += (self.days_rented - 3) * 1.5
    return result
```

Third step : Move Method

```
class Customer(object):
...
def amount_for(self, rental):
    return rental.get_charge()

def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        this_amount = self.amount_for(rental)
    ...
```

Third step : Move Method

```
class Customer(object):
...
def amount_for(self, rental):
    return rental.get_charge()

def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        this_amount = self.amount_for(rental)
    ...
```

```
class Customer(object):
...
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        this_amount = rental.get_charge()
    ...
```

Forth step : Replace Temp with Query

```python
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        this_amount = rental.get_charge()

        # add requent renter points
        frequent_renter_points += 1

        # add bonus for a two day new release rental
        if rental.movie.price_code == Movie.NEW_RELEASE and rental.days_rented > 1:
            frequent_renter_points += 1

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, this_amount)
        total_amount += this_amount

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)

    return result
```

Forth step : Replace Temp with Query

```
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        this_amount = rental.get_charge()

        # add requent renter points
        frequent_renter_points += 1

        # add bonus for a two day new release rental
        if rental.movie.price_code == Movie.NEW_RELEASE and rental.days_rented > 1:
            frequent_renter_points += 1

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, this_amount)
        total_amount += this_amount

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)

    return result
```

Temps are often a problem in that they cause a lot of parameters to be passed around when they don't have to be

Forth step : Replace Temp with Query

```python
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        # add requent renter points
        frequent_renter_points += 1

        # add bonus for a two day new release rental
        if rental.movie.price_code == Movie.NEW_RELEASE and rental.days_rented > 1:
            frequent_renter_points += 1

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())
        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)

    return result
```

Fifth step : Extract Method

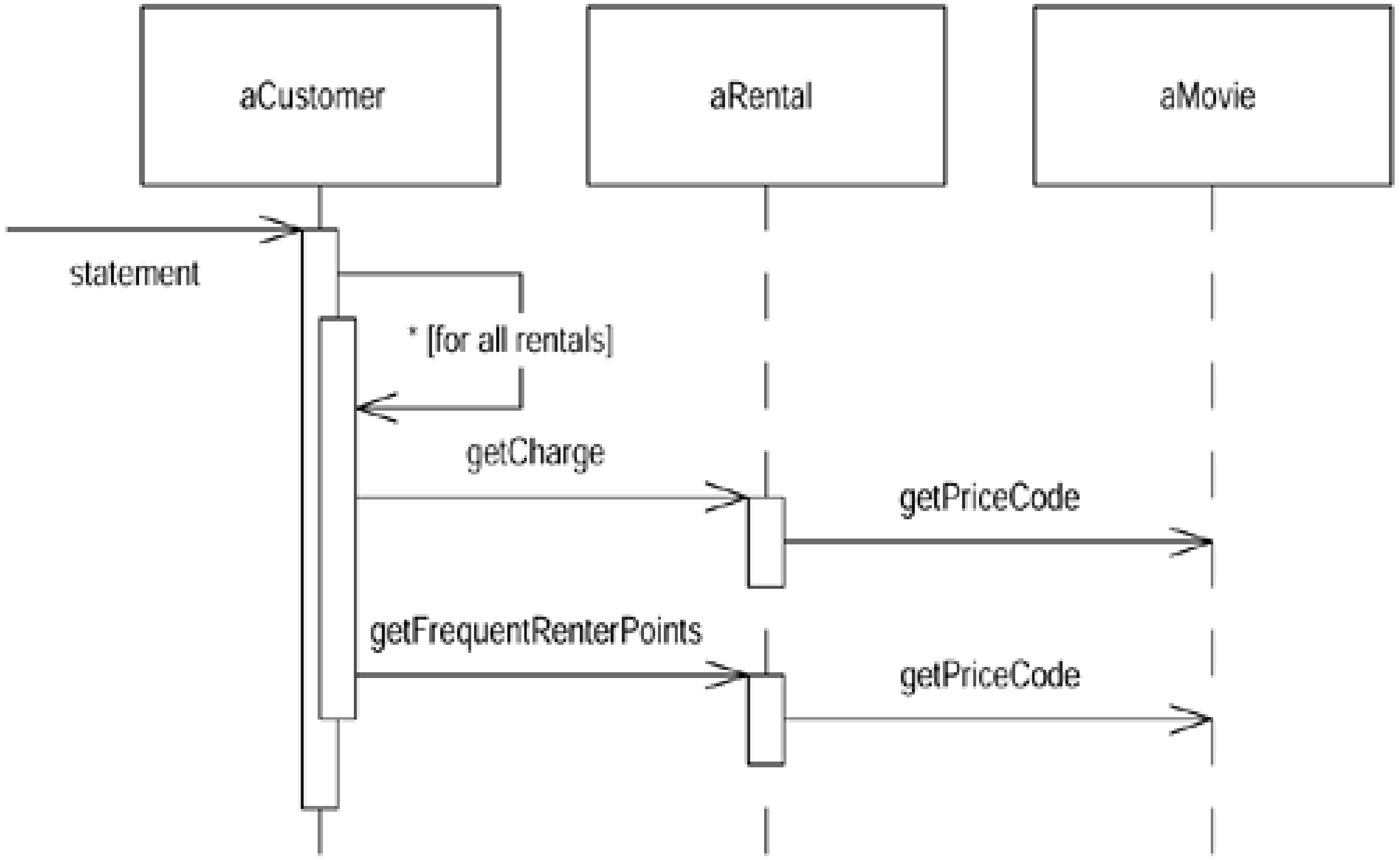
```python
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        # add requent renter points
        frequent_renter_points += 1

        # add bonus for a two day new release rental
        if rental.movie.price_code == Movie.NEW_RELEASE and rental.days_rented > 1:
            frequent_renter_points += 1

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())
        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)

    return result
```

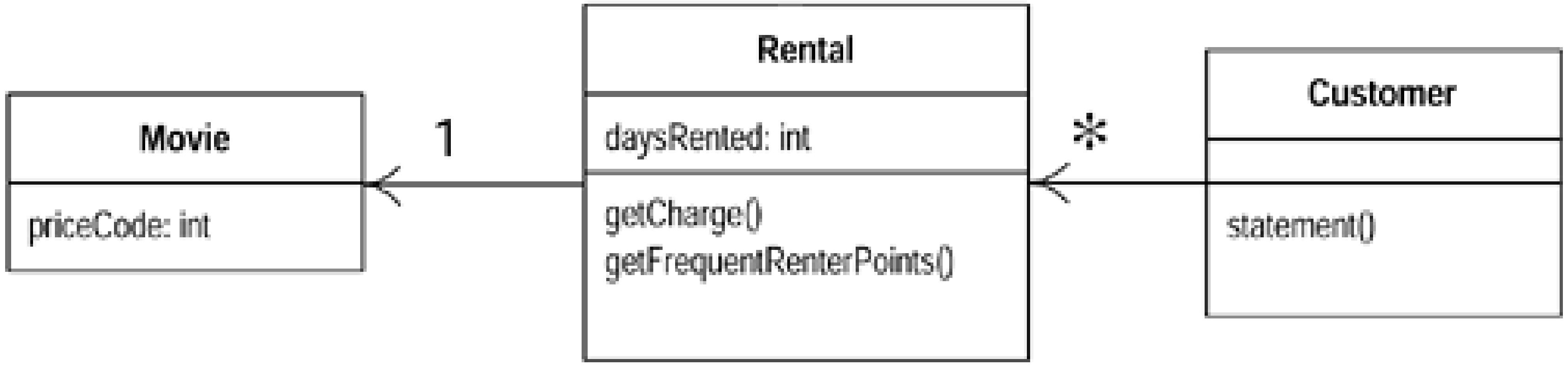
Fifth step : Extract Method

```
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:
        frequent_renter_points += rental.get_frequent_renter_points()

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())
        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)
    return result
```

```
class Rental(object):
...
def get_frequent_renter_points(self):
    if self.movie.price_code == Movie.NEW_RELEASE and self.days_rented > 1:
        return 2
    else:
        return 1
```

Sequence diagrams and Class diagram





One loop, One function

```
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:
        frequent_renter_points += rental.get_frequent_renter_points()

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())
        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)
    return result
```

You should make each loop perform only one function.

One loop, One function

how many performing functions are in this loop?

```
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:
        frequent_renter_points += rental.get_frequent_renter_points()

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())
        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)
    return result
```

You should make each loop perform only one function.

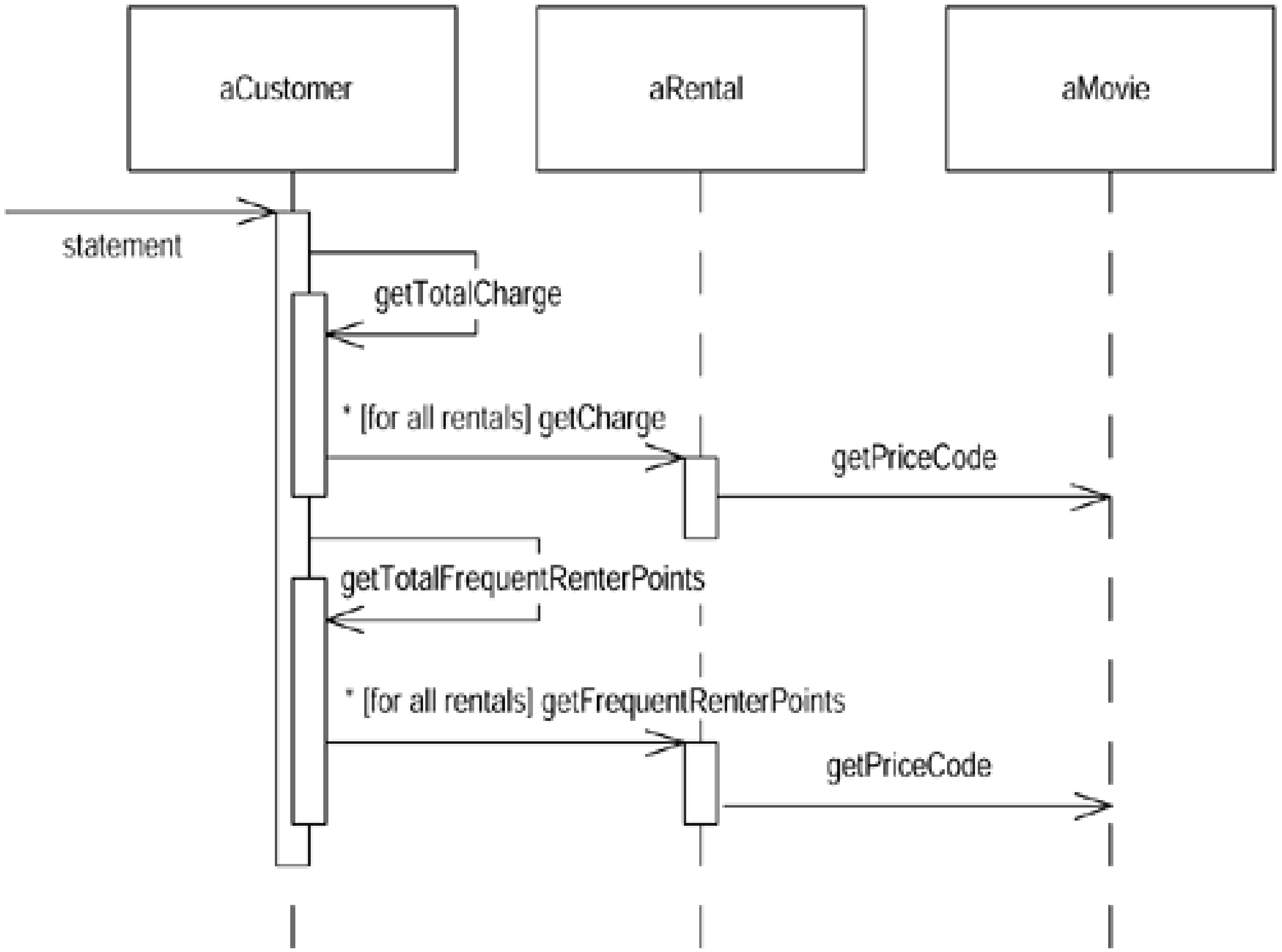
Sixth step : Replace Temp with Query

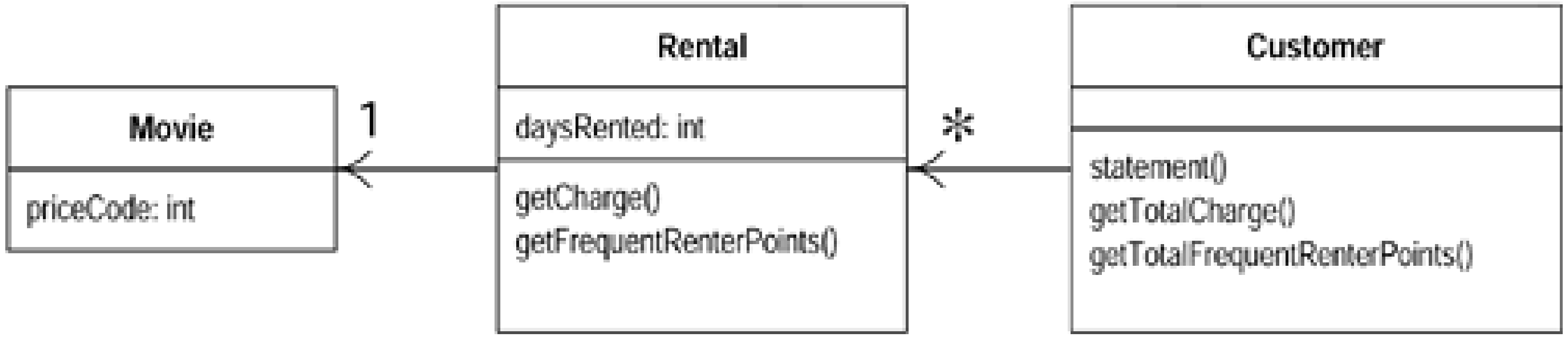
```
def statement(self):
    total_amount = 0.0
    frequent_renter_points = 0
    result = 'Rental Record for %s\n' % (self.name)
    for rental in self._rentals:

        frequent_renter_points += rental.get_frequent_renter_points()

        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        total_amount += rental.get_charge()

    # add footer lines
    result += 'Amount owed is %.1f\n' % (total_amount)
    result += 'You earned %d frequent renter points' % (frequent_renter_points)
    return result
```

Sixth step : Replace Temp with Query

```
def get_total_charge(self):
    result = 0.0
    for rental in self._rentals:
        result += rental.get_charge()
    return result

def get_total_frequent_renter_points(self):
    result = 0.0
    for rental in self._rentals:
        result += rental.get_frequent_renter_points()
    return result

def statement(self):
    result = 'Rental Record for %s\n' % (self.name)

    for rental in self._rentals:
        # show figures for this rental
        result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

    # add footer lines
    result += 'Amount owed is %.1f\n' % (self.get_total_charge())
    result += 'You earned %d frequent renter points' % \
        (self.get_total_frequent_renter_points())
    return result
```

Sequence diagrams and Class diagram


aCustomer
aRental
aMovie
statement
getTotalCharge
* [for all rentals] getCharge
getPriceCode
getTotalFrequentRenterPoints
* [for all rentals] getFrequentRenterPoints
getPriceCode

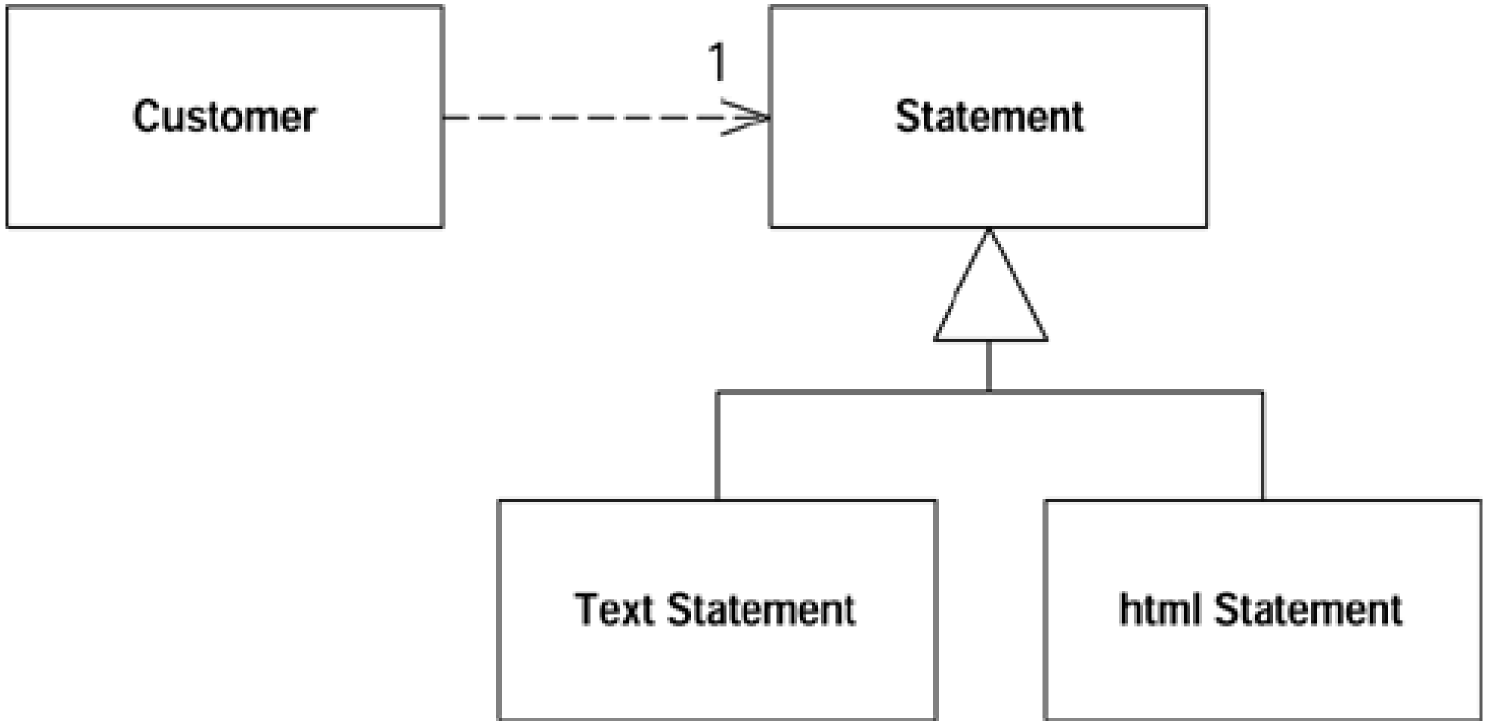


HTML Statement

```
def html_statement(self):
    result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (self.name)
    for rental in self._rentals:
        # show figures for this rental
        result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

    # add footer lines
    result += '<p>You owe <em>%.1f</em><p>\n' % (self.get_total_charge())
    result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
        (self.get_total_frequent_renter_points())
    return result
```

statement and html_statement methods perform similar steps in the same order, yet the steps are different.

Seventh step : Form Template Method

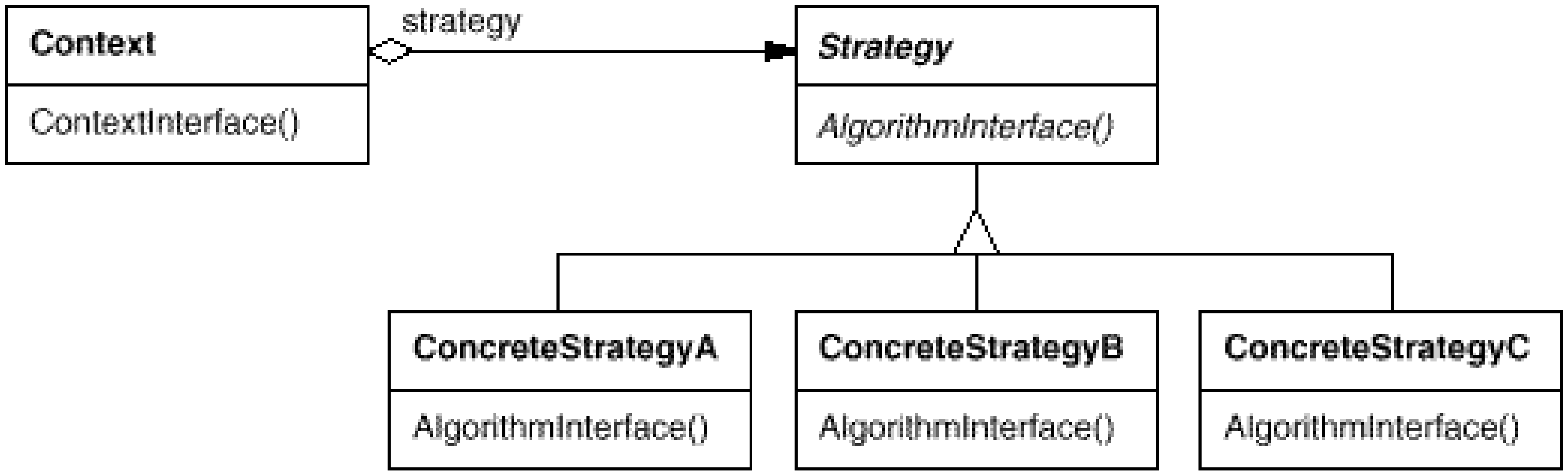


First, we have to create a separate ***strategy*** hierarchy for printing the statements and move the two statement methods over to the subclasses.

Design Patterns : Strategy

- Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

- Structure



Seventh step : Form Template Method

```python
class Statement(object):
    def value(self, customer):
        raise NotImplementedError

class TextStatement(Statement):
    def value(self, customer):
        result = 'Rental Record for %s\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += 'Amount owed is %.1f\n' % (customer.get_total_charge())
        result += 'You earned %d frequent renter points' % \
            (customer.get_total_frequent_renter_points())
        return result

class HtmlStatement(Statement):
    def value(self, customer):
        result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge())
        result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
            (customer.get_total_frequent_renter_points())
        return result
```

Seventh step : Form Template Method

```
class Customer(object):
...
def html_statement(self):
    return HtmlStatement().value(self)

def statement(self):
    return TextStatement().value(self)
```

Now, we can separate the varying code from the similar code by using Extract Method to extract the pieces that are different between the two methods.

Seventh step : Form Template Method

```
class TextStatement(Statement):
    def value(self, customer):
        result = 'Rental Record for %s\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += 'Amount owed is %.1f\n' % (customer.get_total_charge())
        result += 'You earned %d frequent renter points' % \
            (customer.get_total_frequent_renter_points())
        return result

class HtmlStatement(Statement):
    def value(self, customer):
        result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge())
        result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
            (customer.get_total_frequent_renter_points())
        return result
```

Seventh step : Form Template Method

header

```
class TextStatement(Statement):
    def value(self, customer):
        result = 'Rental Record for %s\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += 'Amount owed is %.1f\n' % (customer.get_total_charge())
        result += 'You earned %d frequent renter points' % \
            (customer.get_total_frequent_renter_points())
        return result
```

header

```
class HtmlStatement(Statement):
    def value(self, customer):
        result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge())
        result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
            (customer.get_total_frequent_renter_points())
        return result
```

Seventh step : Form Template Method

header

each rental

```python
class TextStatement(Statement):
    def value(self, customer):
        result = 'Rental Record for %s\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result +=  '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += 'Amount owed is %.1f\n' % (customer.get_total_charge())
        result += 'You earned %d frequent renter points' % \
            (customer.get_total_frequent_renter_points())
        return result
```

header

each rental

```python
class HtmlStatement(Statement):
    def value(self, customer):
        result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge())
        result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
            (customer.get_total_frequent_renter_points())
        return result
```

Seventh step : Form Template Method

```
class TextStatement(Statement):
    def value(self, customer):
        result = 'Rental Record for %s\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += 'Amount owed is %.1f\n' % (customer.get_total_charge())
        result += 'You earned %d frequent renter points' % \
            (customer.get_total_frequent_renter_points())
        return result
```

header

each rental

footer

```
class HtmlStatement(Statement):
    def value(self, customer):
        result = '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)
        for rental in customer.rentals:
            # show figures for this rental
            result += '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

        # add footer lines
        result += '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge())
        result += 'On this rental you earned <em>%d</em> frequent renter points<p>' % \
            (customer.get_total_frequent_renter_points())
        return result
```

header

each rental

footer

Seventh step : Form Template Method

```
class TextStatement(Statement):
    def value(self, customer):
        result = self.header_string(customer)
        for rental in customer.rentals:
            # show figures for this rental
            result += self.each_rental_string(rental)
        # add footer lines
        result += self.footer_string(customer)
        return result

    def header_string(self, customer):
        return 'Rental Record for %s\n' % (customer.name)

    def each_rental_string(self, rental):
        return '    %s    %.1f\n' % (rental.movie.title, rental.get_charge())

    def footer_string(self, customer):
        return 'Amount owed is %.1f\n' % (customer.get_total_charge()) + \
               'You earned %d frequent renter points' % \
                    (customer.get_total_frequent_renter_points())
```

Seventh step : Form Template Method

```python
class HtmlStatement(Statement):
    def value(self, customer):
        result = self.header_string(customer)
        for rental in customer.rentals:
            # show figures for this rental
            result += self.each_rental_string(rental)
        # add footer lines
        result += self.footer_string(customer)
        return result

    def header_string(self, customer):
        return '<h1>Rentals for <em>%s</em></h1><p>\n' % (customer.name)

    def each_rental_string(self, rental):
        return '%s : %.1f<br>\n' % (rental.movie.title, rental.get_charge())

    def footer_string(self, customer):
        return '<p>You owe <em>%.1f</em><p>\n' % (customer.get_total_charge()) + \
               'On this rental you earned <em>%d</em> frequent renter points<p>' % \
                    (customer.get_total_frequent_renter_points())
```

Seventh step : Form Template Method

```
class Statement(object):

    def value(self, customer):
        result = self.header_string(customer)
        for rental in customer.rentals:
            # show figures for this rental
            result += self.each_rental_string(rental)
        # add footer lines
        result += self.footer_string(customer)
        return result

    def header_string(self, customer):
        raise NotImplementedError

    def each_rental_string(self, rental):
        raise NotImplementedError
```

Finally, pull the value method from two subclasses to their super class

Another Ugly Code

```
def get_charge(self):
    result = 0.0
    # determine amount for each line
    if self.movie.price_code == Movie.REGULAR:
        result += 2.0
        if self.days_rented > 2:
            result += (self.days_rented - 2) * 1.5
    elif self.movie.price_code == Movie.NEW_RELEASE:
        result += self.days_rented * 3
    elif self.movie.price_code == Movie.CHILDRENS:
        result += 1.5
        if self.days_rented > 3:
            result += (self.days_rented - 3) * 1.5
    return result
```

What are the problems of this code?

Another Ugly Code

- It is a bad idea to do a switch based on an attribute of another object. If you must use a switch statement, it should be on your own data, not on someone else's.

- Keeping `getCharge` in the Movie class has the least ripple effect from adding new movie types or editing the existing ones.

Eighth step : Move Method

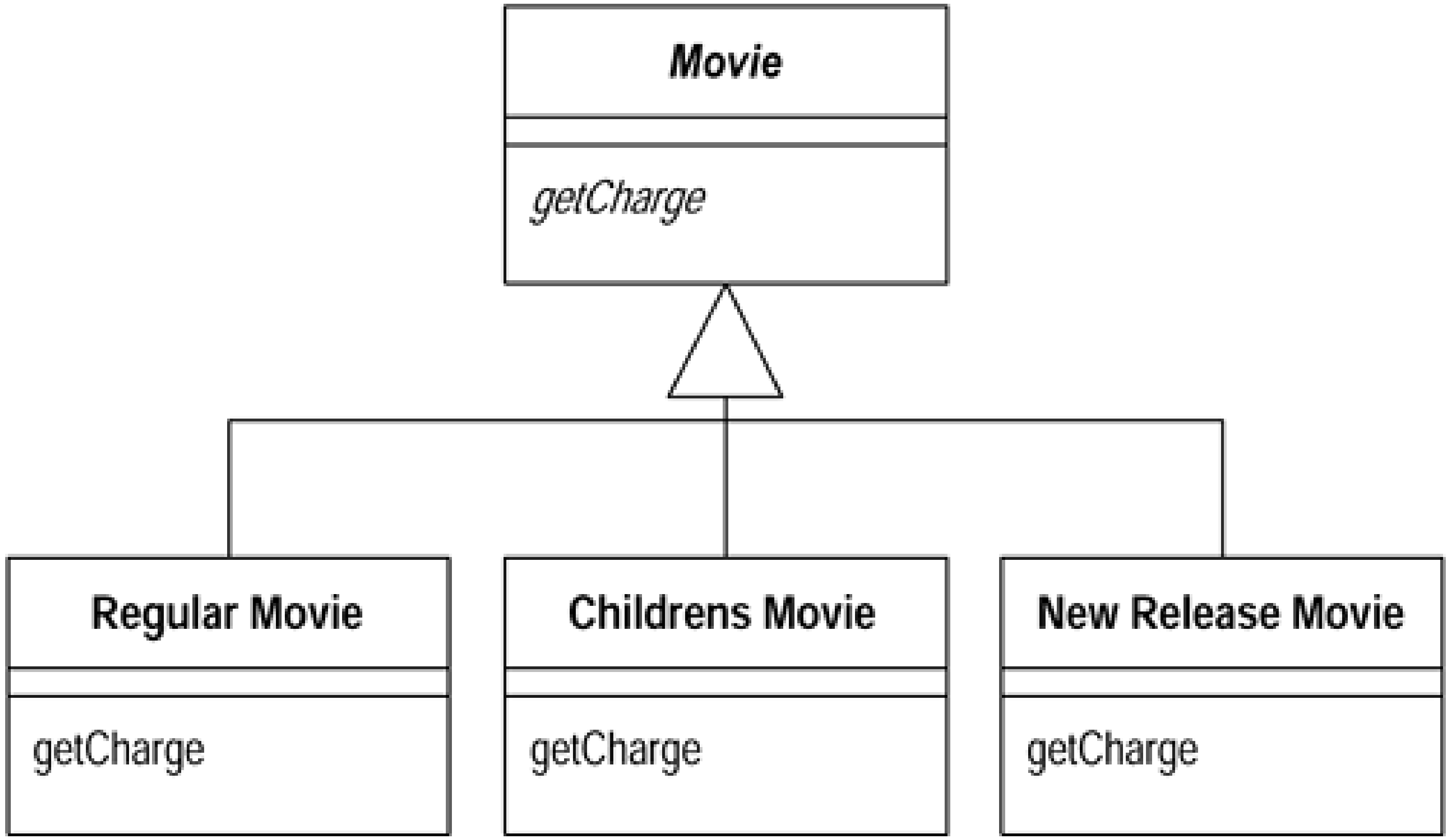
```
class Movie(object):
...
def get_charge(self, days_rented):
    result = 0.0
    if self.price_code == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.price_code == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.price_code == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.price_code == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1


class Rental(object):
...
def get_charge(self):
    return self.movie.get_charge(self.days_rented)

def get_frequent_renter_points(self):
    return self.movie.get_frequent_renter_points(self.days_rented)
```

This code still be ugly (why?)

```python
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.price_code == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.price_code == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.price_code == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result
```

This code performs 3 tasks depending on the type of the movie.

Can we have subclasses of movie?

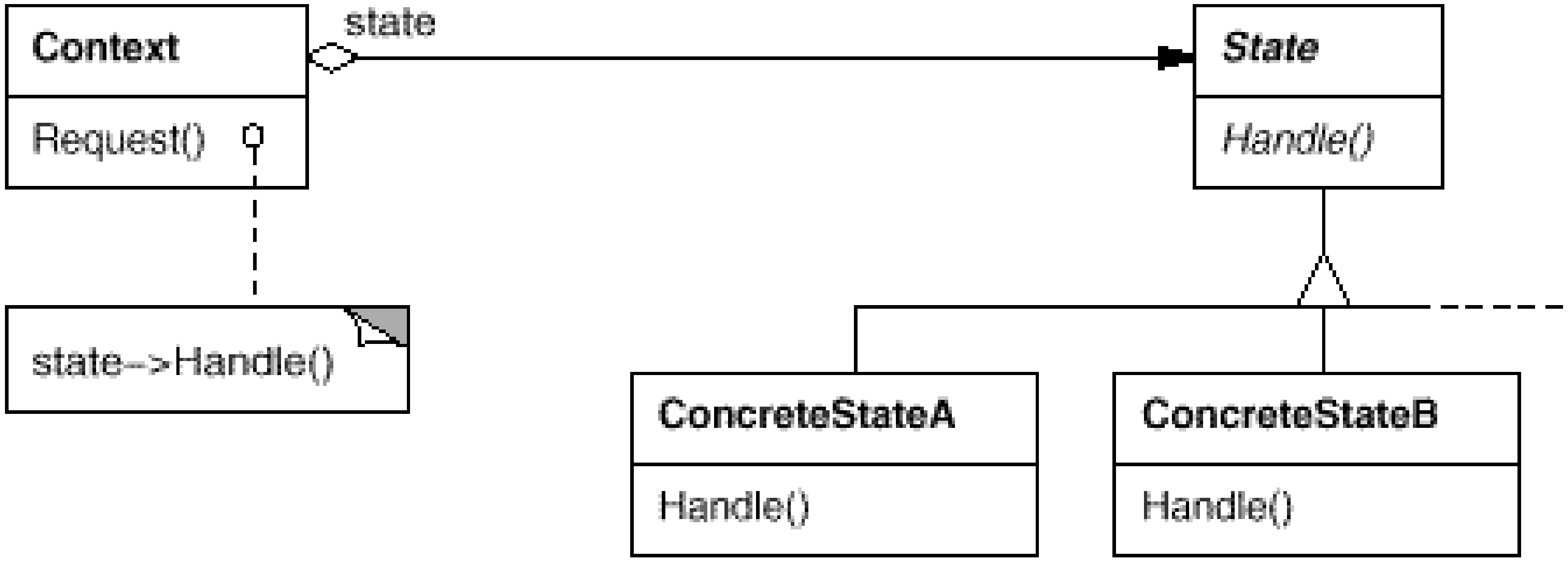


What happens if the movie type is changed?
Because an object cannot change its class during its lifetime.

Design Patterns : State

- Allow an object to alter its behavior when its internal state changes.The object will appear to change its class.

- Structure

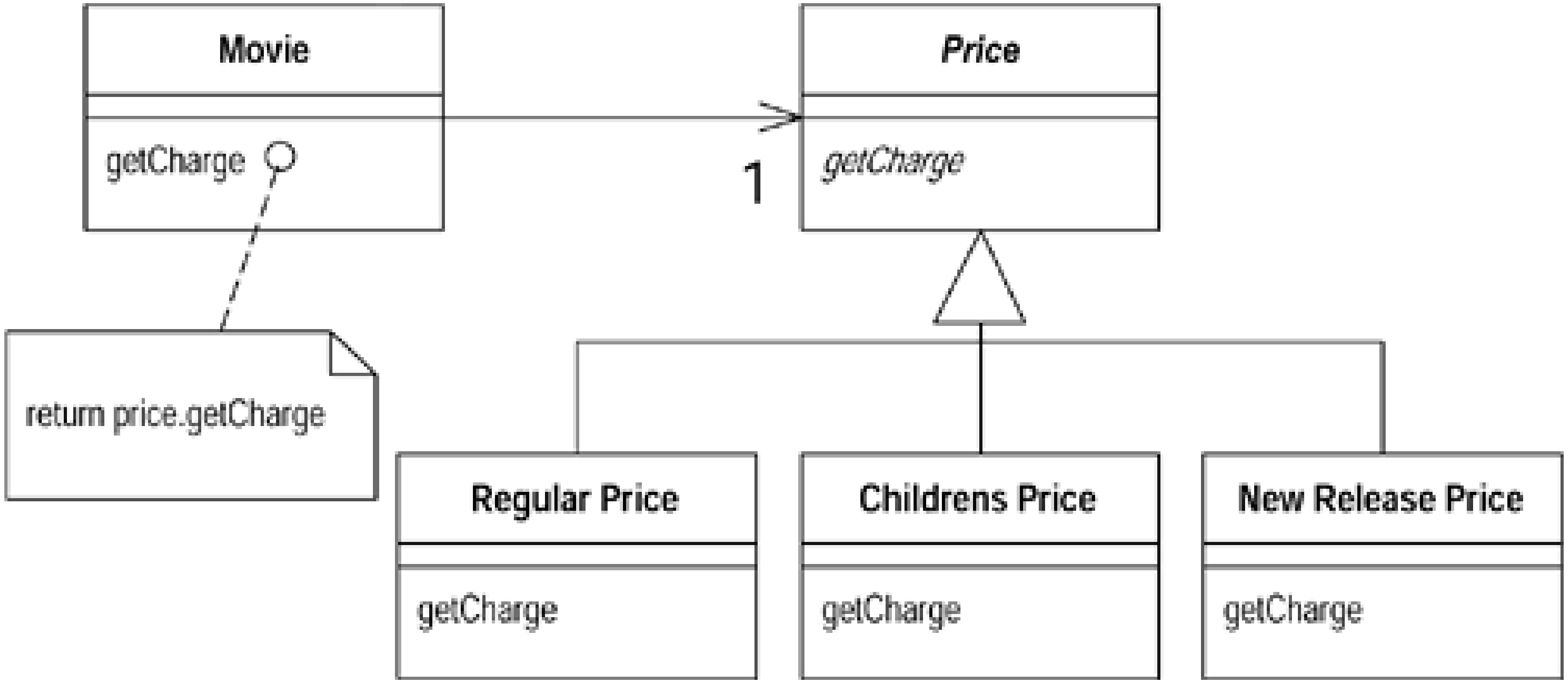


What is the different between State and Strategy?

- http://www.c-sharpcorner.com/UploadFile/rmcochran/strategy_state01172007114905AM/strategy_state.aspx

- http://stackoverflow.com/questions/1658192/what-is-the-difference-between-strategy-design-pattern-and-state-design-pattern

Using the State pattern on movie



Ninth step : Replace Type Code with State

- Self Encapsulate Field
- Movie Method
- Replace Conditional with Polymorphism

Self Encapsulate Field

```
class Price(object):
    def get_price_code(self):
        raise NotImplementedError

class ChildrensPrice(Price):
    def get_price_code(self):
        return Movie.CHILDRENS

class RegularPrice(Price):
    def get_price_code(self):
        return Movie.REGULAR

class NewReleasePrice(Price):
    def get_price_code(self):
        return Movie.NEW_RELEASE
```

Move Method

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

Move Method

```
class Movie(object):
...
@property
def price(self):
    return self._price

@price.setter
def price(self, price):
    self._price = price

@property
def price_code(self):
    return self._price.get_price_code()

@price_code.setter
def price_code(self, price_code):
    if price_code == Movie.REGULAR:
        self.price = RegularPrice()
    elif price_code == Movie.NEW_RELEASE:
        self.price = NewReleasePrice()
    elif price_code == Movie.CHILDRENS:
        self.price = ChildrensPrice()

def get_charge(self, days_rented):
    return self.price.get_charge(days_rented)

def get_frequent_renter_points(self, days_rented):
    return self.price.get_frequent_renter_points(days_rented)
```

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

RegularPrice

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

RegularPrice

NewReleasePrice

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

RegularPrice

NewReleasePrice

ChildrensPrice

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

RegularPrice

NewReleasePrice

ChildrensPrice

NewReleasePrice

Replace Conditional with Polymorphism

```
class Price(object):
...
def get_charge(self, days_rented):
    result = 0.0
    # determine amount for each line
    if self.get_price_code() == Movie.REGULAR:
        result += 2.0
        if days_rented > 2:
            result += (days_rented - 2) * 1.5
    elif self.get_price_code() == Movie.NEW_RELEASE:
        result += days_rented * 3
    elif self.get_price_code() == Movie.CHILDRENS:
        result += 1.5
        if days_rented > 3:
            result += (days_rented - 3) * 1.5
    return result

def get_frequent_renter_points(self, days_rented):
    if self.get_price_code() == Movie.NEW_RELEASE and days_rented > 1:
        return 2
    else:
        return 1
```

RegularPrice

NewReleasePrice

ChildrensPrice

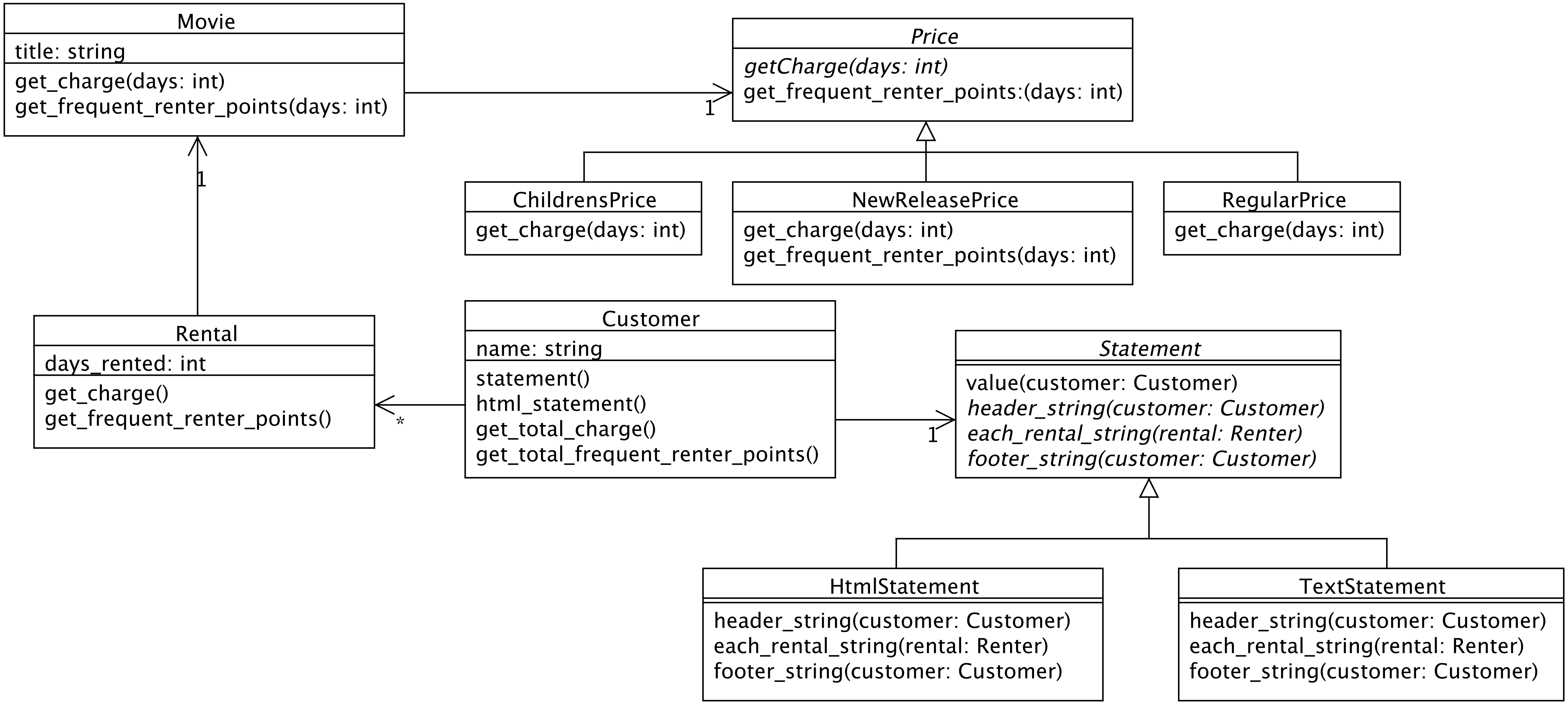
NewReleasePrice

Price

Replace Conditional with Polymorphism

```
class Price(object):
    ...
    def get_charge(self, days_rented):
        raise NotImplementedError

    def get_frequent_renter_points(self, days_rented):
        return 1

class ChildrensPrice(Price):
    ...
    def get_charge(self, days_rented):
        return 1.5+(days_rented-3)*1.5 if days_rented > 3 else 1.5

class RegularPrice(Price):
    ...
    def get_charge(self, days_rented):
        return 2.0+(days_rented-2)*1.5 if days_rented > 2 else 2.0

class NewReleasePrice(Price):
    ...
    def get_frequent_renter_points(self, days_rented):
        return 2 if days_rented > 1 else 1

    def get_charge(self, days_rented):
        return days_rented * 3
```

Class Diagram



Movie
title: string
get_charge(days: int)
get_frequent_renter_points(days: int)

Price
getCharge(days: int)
get_frequent_renter_points:(days: int)

ChildrensPrice
get_charge(days: int)

NewReleasePrice
get_charge(days: int)
get_frequent_renter_points(days: int)

RegularPrice
get_charge(days: int)

Rental
days_rented: int
get_charge()
get_frequent_renter_points()

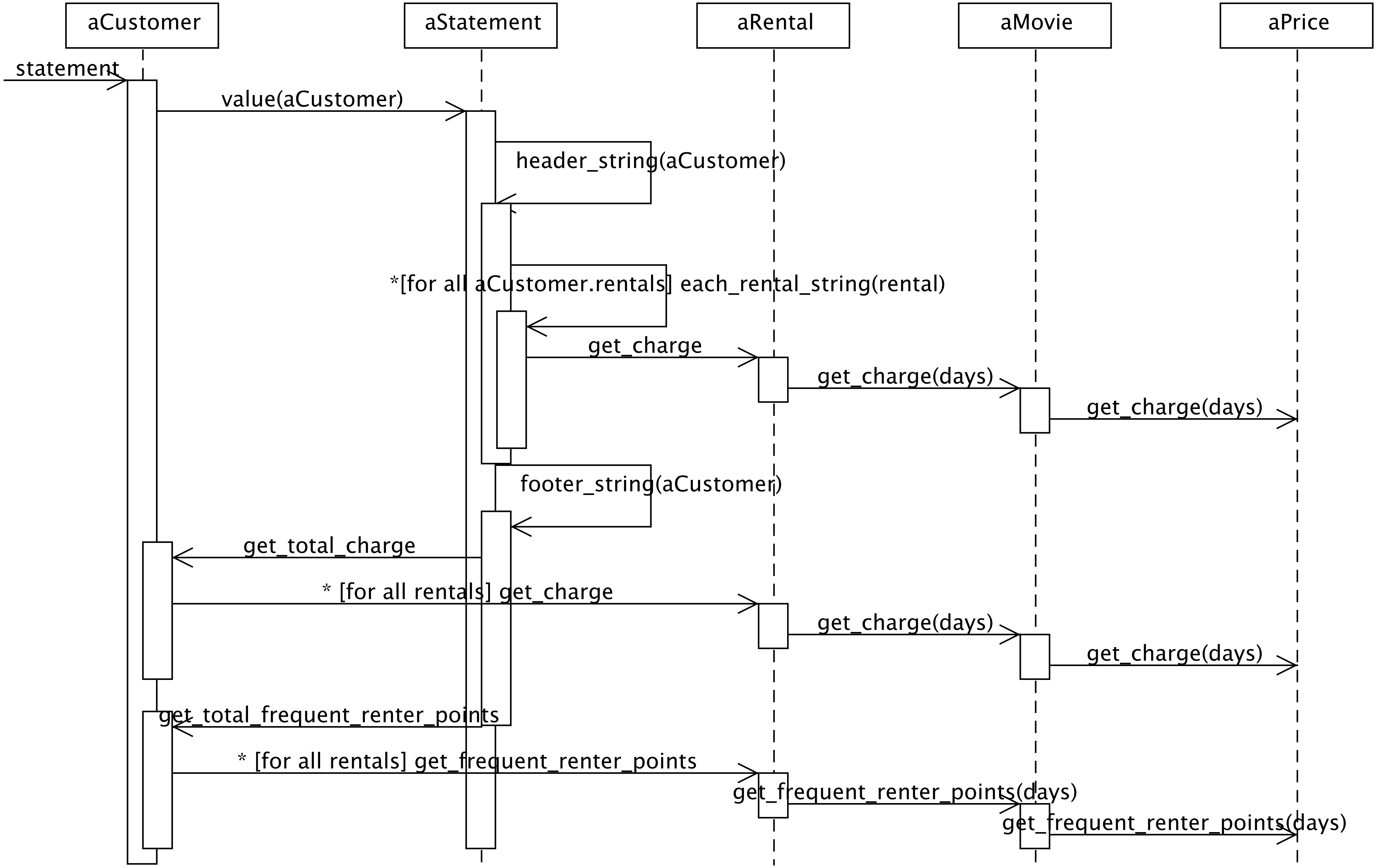
Customer
name: string
statement()
html_statement()
get_total_charge()
get_total_frequent_renter_points()

Statement
value(customer: Customer)
header_string(customer: Customer)
each_rental_string(rental: Renter)
footer_string(customer: Customer)

HtmlStatement
header_string(customer: Customer)
each_rental_string(rental: Renter)
footer_string(customer: Customer)

TextStatement
header_string(customer: Customer)
each_rental_string(rental: Renter)
footer_string(customer: Customer)

Sequence Diagram


aCustomer
aStatement
aRental
aMovie
aPrice
statement
value(aCustomer)
header_string(aCustomer)
*[for all aCustomer.rentals] each_rental_string(rental)
get_charge
get_charge(days)
get_charge(days)
footer_string(aCustomer)
get_total_charge
* [for all rentals] get_charge
get_charge(days)
get_charge(days)
get_total_frequent_renter_points
* [for all rentals] get_frequent_renter_points
get_frequent_renter_points(days)
get_frequent_renter_points(days)

Principles in Refactoring

The Two Hats

- When you use refactoring to develop software, you divide your time between two distinct activities:
    - adding function
        - When you add function, you shouldn't be changing existing code; you are just adding new capabilities.
    - refactoring
        - When you refactor, you make a point of not adding function; you only restructure the code.

Why Should You Refactor?

- Improves the design of software
- makes software easier to understand
- helps you find bugs
- helps you program faster

When Should You Refactor? : The Rule of Three

- Refactor when you add function
    - if you have a hard time to add a new function, you need refactoring
- Refactor when you need to fix a bug
    - if you do get a bug report, you need refactoring because the was not clear enough for you to see there were a bug
- Refactor as you do a code review
    - refactoring also helps the code review have more concrete rseults

When Shouldn’t You Refactor?

- When you should rewrite from scratch instead
- When you closed to a deadline
    - unfinished refactoring as going into debt