Тема 14. Рефакторинг кода

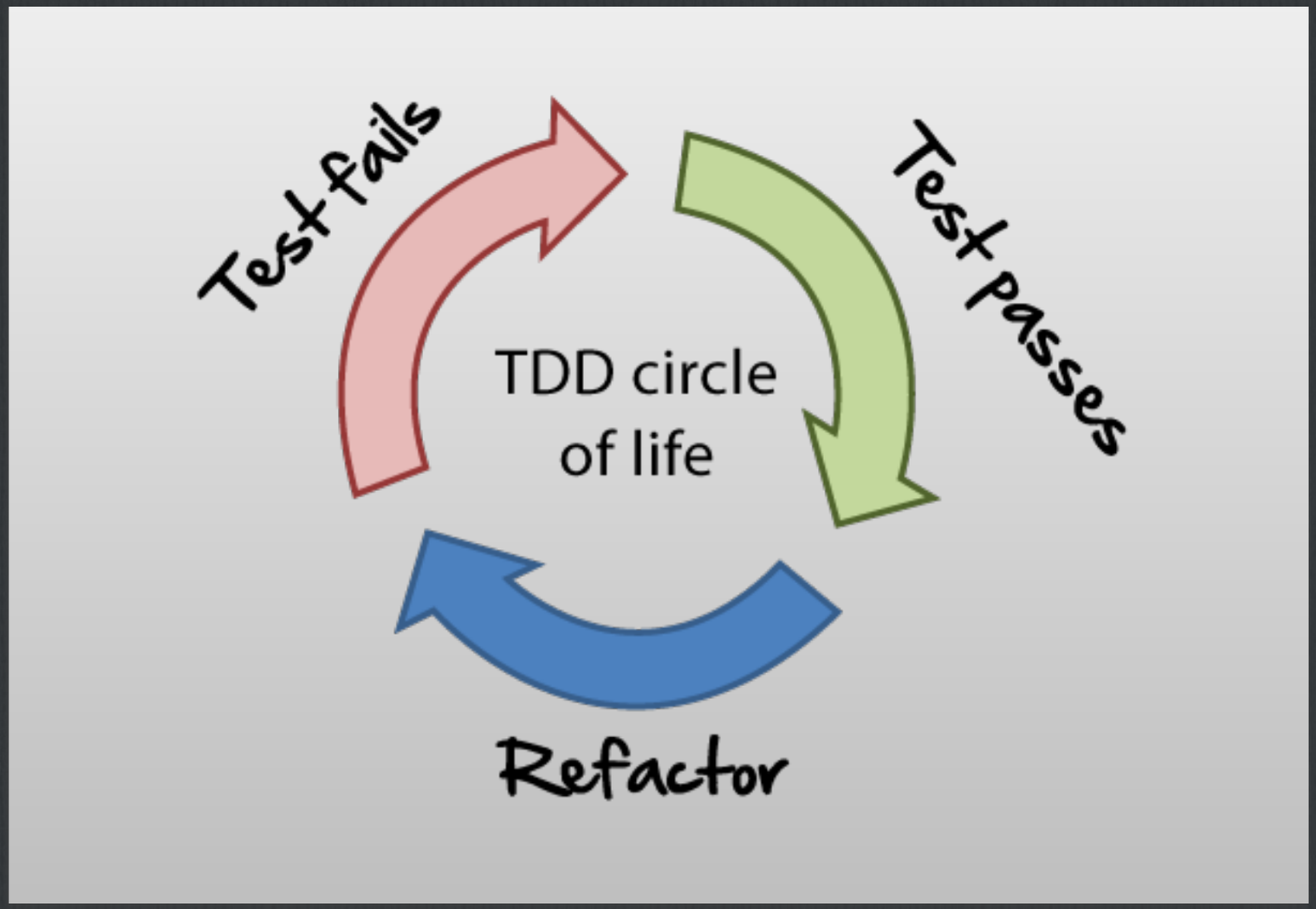
1. Что такое рефакторинг?
2. Пример проведения рефакторинга
3. Рефакторинг и тесты

Определение

- Рефакторинг - процесс такого изменения программной системы, при котором не меняется внешнее поведение кода, а улучшается его внутренняя структура
- Способ приведения кода в порядок
- Улучшение дизайна, после того, как написан код

Code and Fix

Test Driven Development



Рефакторинг -
обязательная часть любого
процесса разработки!

Refactoring

Ref%@ktoring

Литература


РЕФАКТОРИНГ
УЛУЧШЕНИЕ
СУЩЕСТВУЮЩЕГО КОДА
МАРТИН ФАУЛЕР
При участии Кента Бека, Джона Брантта,
Уильяма Опдайка и Дона Роберта
Предисловие Эрика Гаммы
Object Technology International, Inc.
BOOCH
JACOBSON
RUMBAUGH

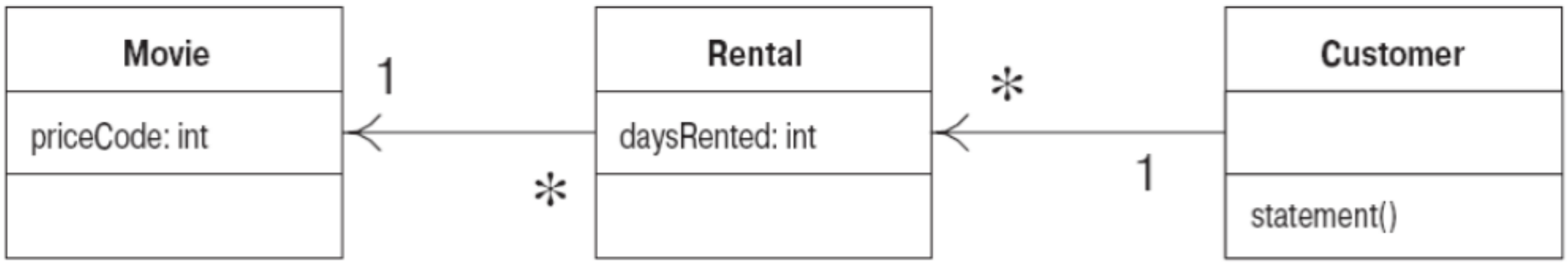

The Addison Wesley Signature Series
РЕФАКТОРИНГ
С ИСПОЛЬЗОВАНИЕМ
ШАБЛОНОВ
Джошуа Кериевски

Пример: Постановка задачи

Расчёт и вывод отчёта об оплате услуг в магазине видеопроката.

Исходные данные:	фильмы, которые брал на прокат клиент, срок, на который клиент брал фильмы.
Результат:	сумма, которую клиент должен оплатить.

Сумма может также зависеть от типа фильма (обычный, детский, новинка), помимо суммы оплаты, за новинки начисляется бонус.



Пример: Основные классы

Класс Movie (данные о фильме)
- _title – название
- _priceCode – коэффициент стоимости
- {CHILDRENS, REGULAR, NEW_RELEASE} – тип фильма
- getPriceCode(), setPriceCode(), getTitle()

Класс Rental (данные о прокате фильма)
- _movie:Movie – фильм
- _daysRented – время проката
- getDaysRented(), getMovie()

Класс Customer (клиент магазина)
- _name – имя клиента
- _rentals:List – список прокатов
- addRental(), getName()

Метод для рефакторинга

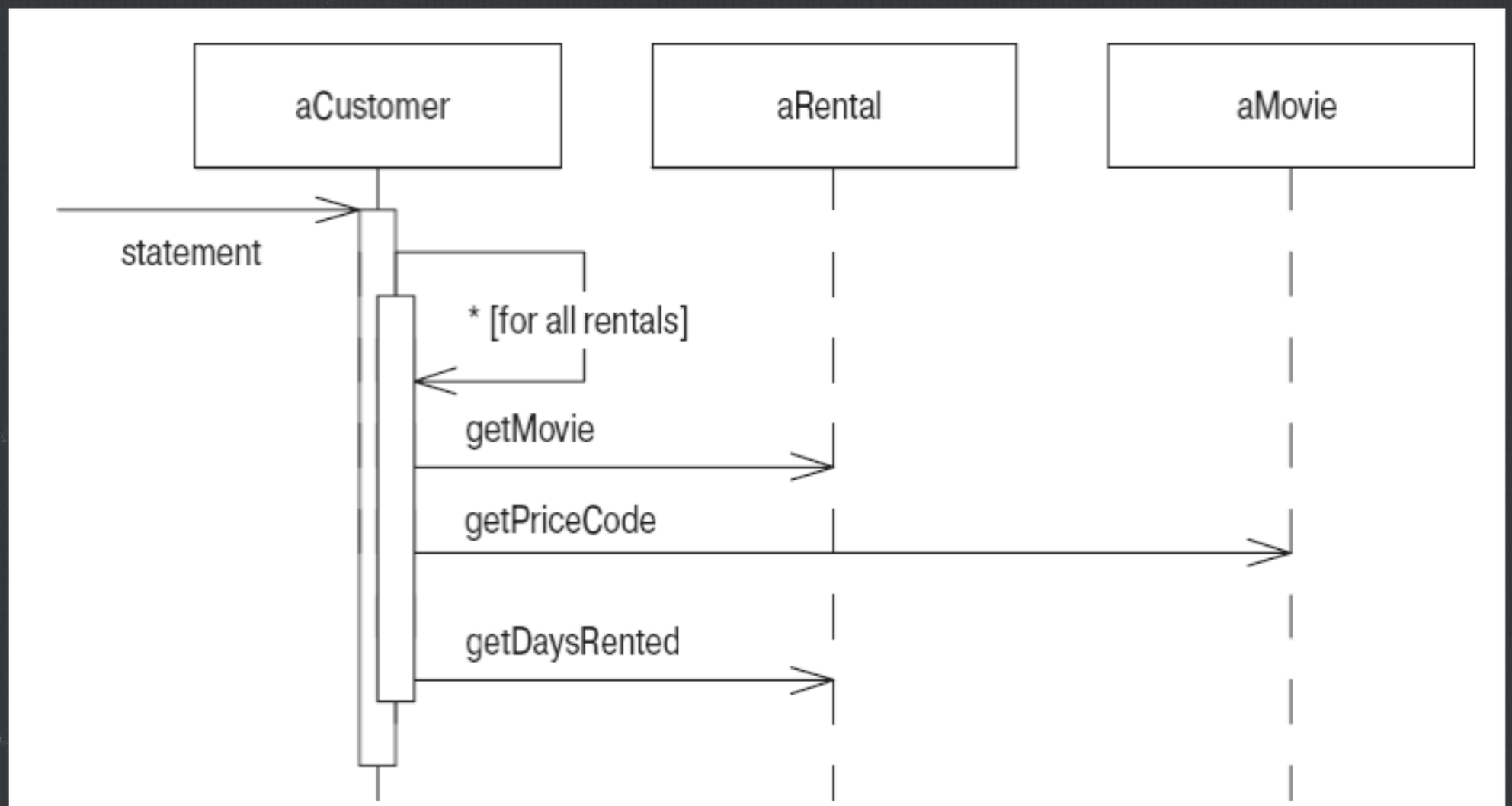
[СМОТРИМ В КОД]

```
//метод для рефакторинга
public String statement() {
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();
    String result = "Учет аренды для: " + getName() + "\n";
    while (rentals.hasMoreElements()) {
        double thisAmount = 0;
        Rental each = (Rental) rentals.nextElement();

        //определить сумму для каждой строки
        switch (each.getMovie().getPriceCode()) {
            case Movie.REGULAR:
                thisAmount += 2;
                if (each.getDaysRented() > 2)
                    thisAmount += (each.getDaysRented() - 2) * 1.5;
                break;
            case Movie.NEW_RELEASE:
                thisAmount += each.getDaysRented() * 3;
                break;
            case Movie.CHILDRENS:
                thisAmount += 1.5;
                if (each.getDaysRented() > 3)
                    thisAmount += (each.getDaysRented() - 3) * 1.5;
                break;
    }
```

```
        // добавить очки для активного арендатора
        frequentRenterPoints++;
        // бонус за аренду новинки на два дня
        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
                each.getDaysRented() > 1) {
            frequentRenterPoints++;
        }
        //показать результаты для этой аренды
        result += "\t" + each.getMovie().getTitle() + "\t" +
                String.valueOf(thisAmount) + "\n";
        totalAmount += thisAmount;
    }
    //добавить нижний колонтитул
    result += "Сумма задолженности составляет: " +
            String.valueOf(totalAmount) + "\n";
    result += "Вы заработали " + String.valueOf(frequentRenterPoints) +
            " очков за активность";
    return result;
  }
}
```

Диаграмма последовательности



Замечания

- ☐ программа НЕ объектно-ориентирована
- ☐ метод statement слишком громоздкий
- ☐ может понадобиться метод генерации отчёта, например, в HTML – htmlStatement
- ☐ может измениться алгоритм подсчёта оплаты
- ☐ может измениться система начисления бонусов

Первый шаг рефакторинга - Тесты!

Легко написать набор тестов:

- ☐ если у нас есть ХОРОШИЕ требования (бизнес-правила)
- ☐ если у нас есть баги

Ритм рефакторинга

- ☐ тестирование
- ☐ малые изменения
- ☐ тестирование
- ☐ малые изменения
- ☐ ...

Самопроверяющиеся тесты

- [ ] сообщения об успехе (совпадение результата с ожидаемым
- [ ] сообщения-списки ошибок

Пишем тесты (обратная инженерия бизнес-правил)

- [ ] для обычного фильма первые два дня стоят $2, каждый следующий день - $1.5
- [ ] новинка стоит $3 в день
- [ ] детский фильм - $1.5 за первые три дня, и $1.5 - за каждый следующий

Рефакторинг 1:
Декомпозиция метода statement

«Выделение метода» Extract Method

1. Определить переменные с локальной областью видимости.
2. Переименовать переменные.

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

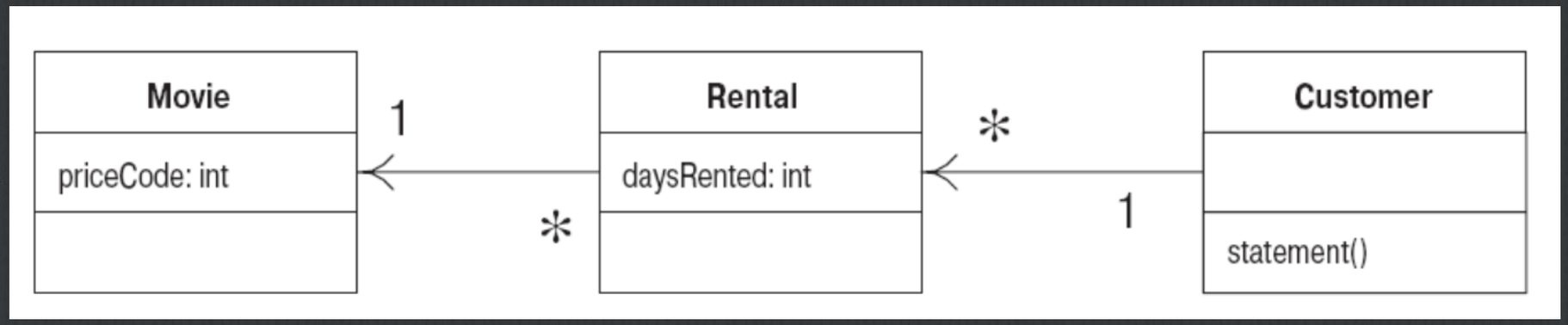
Рефакторинг 1: Декомпозиция метода statement

Метод amountFor() использует данные класса Rental и не использует данные класса Customer.

«Перемещение метода» Move Method

1. Скопировать перемещаемый код в другой класс.
2. Настроить ссылки.

Диаграмма классов после перемещения метода



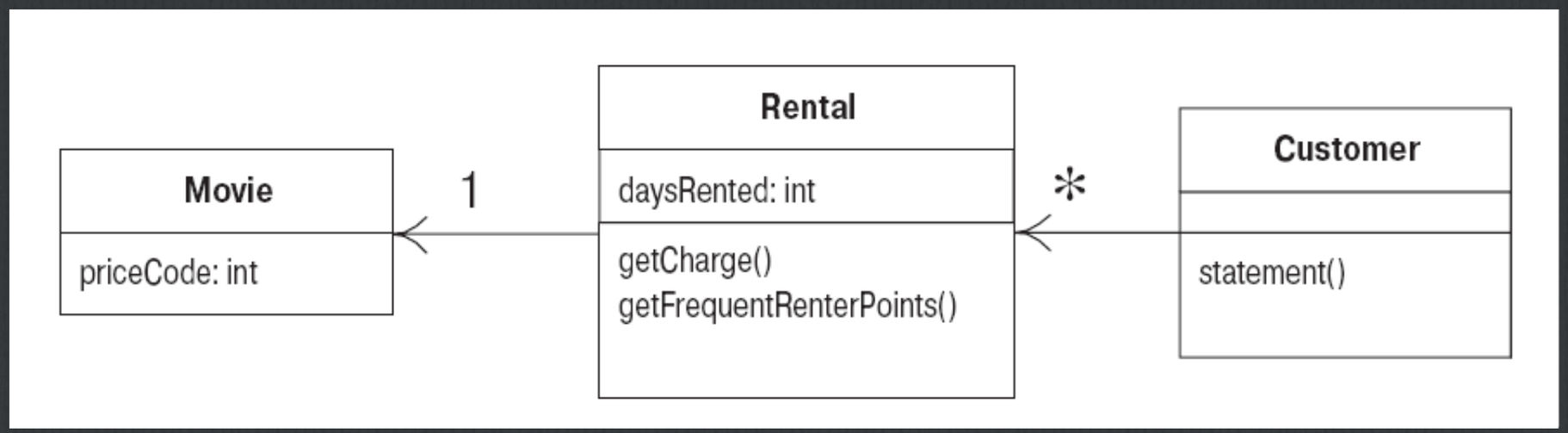


Диаграмма последовательности после выделения методов

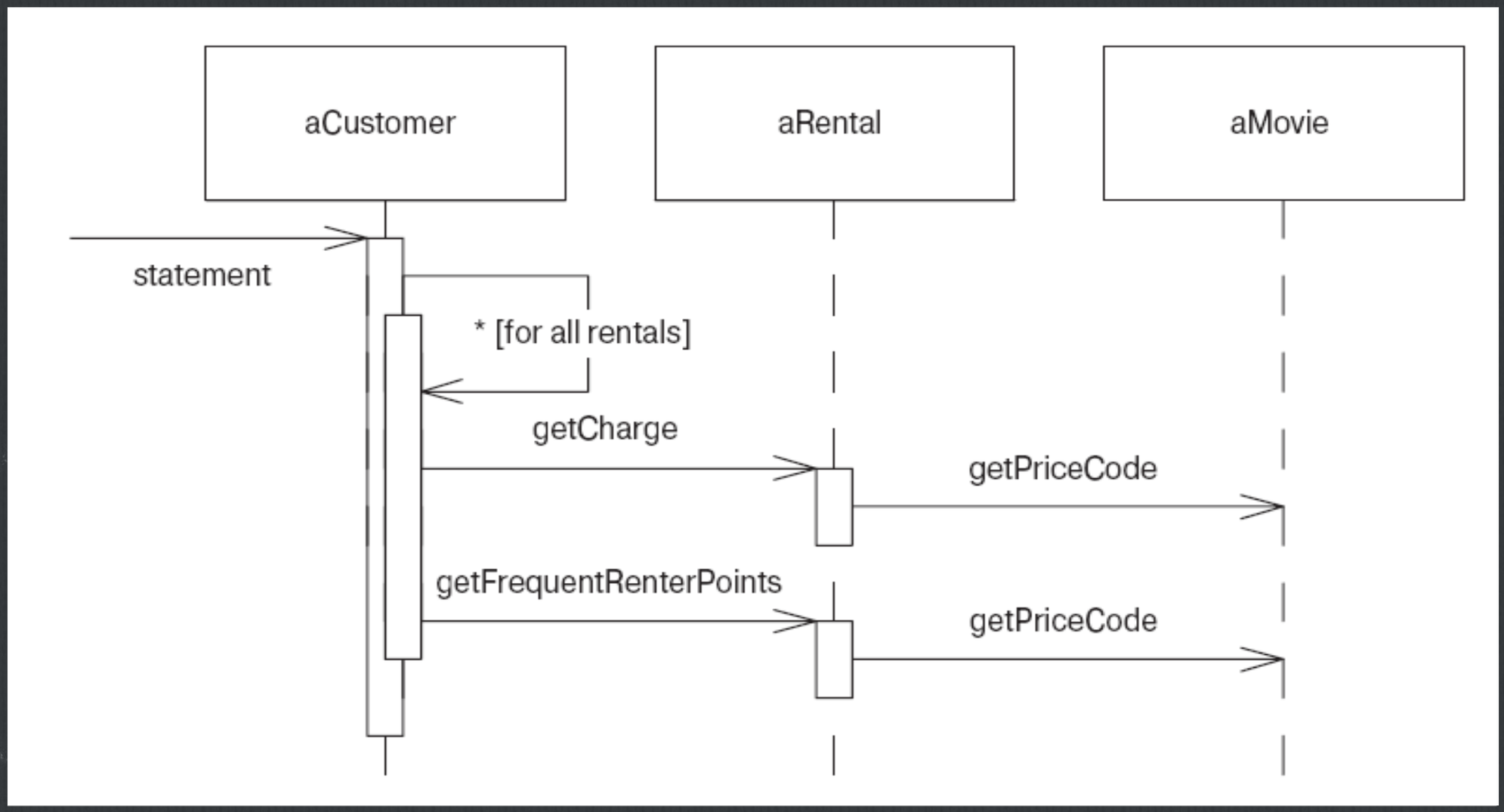
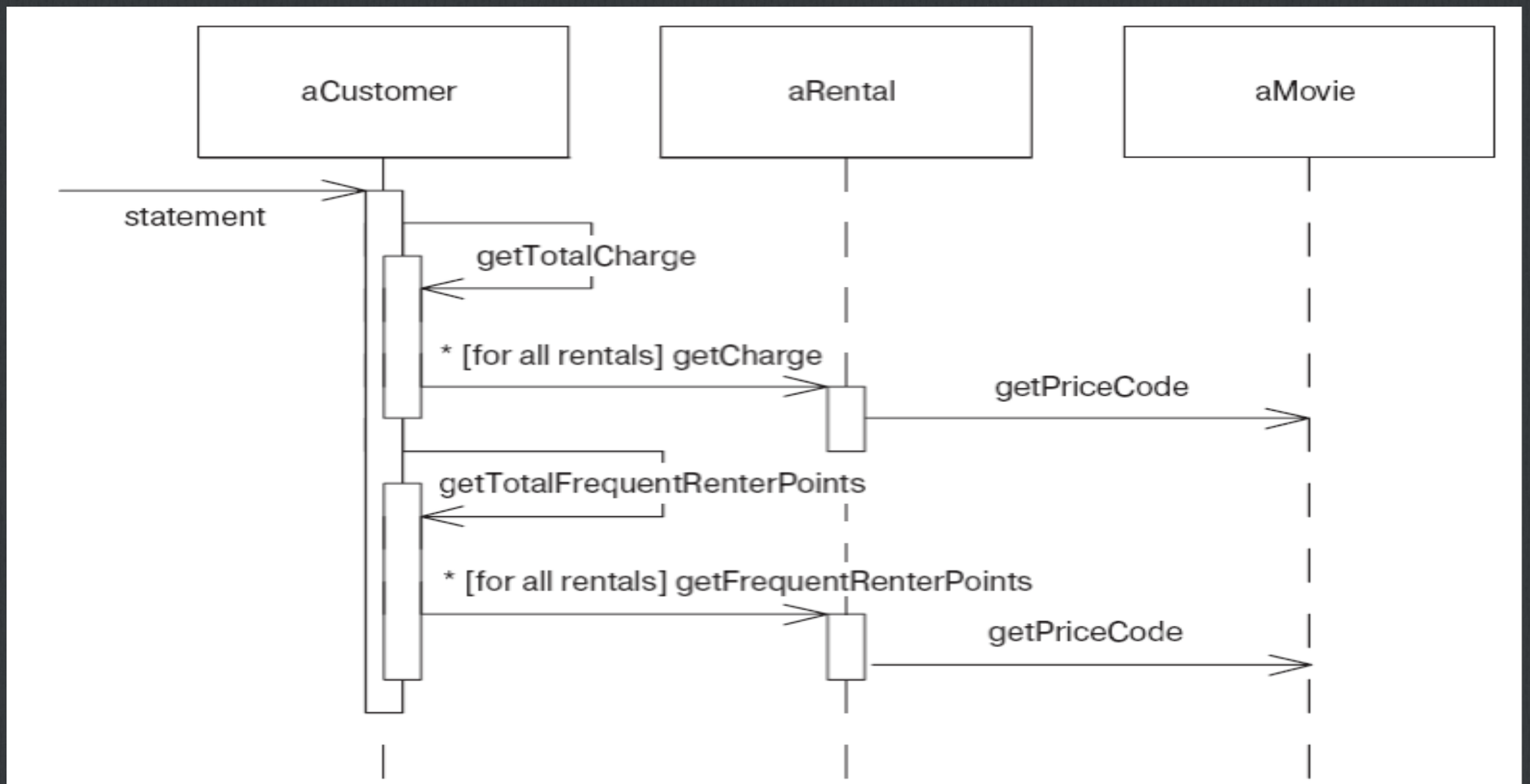
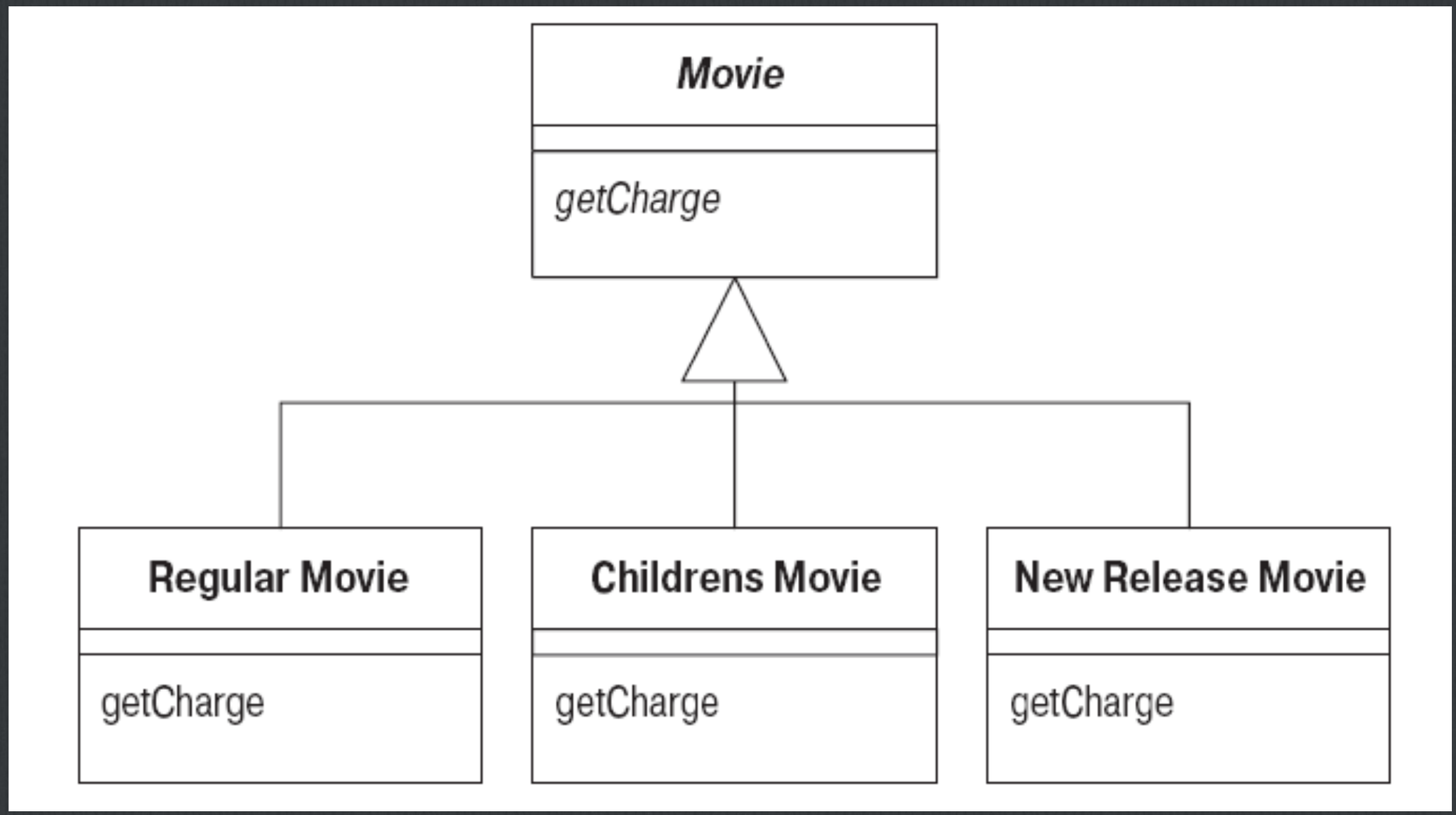


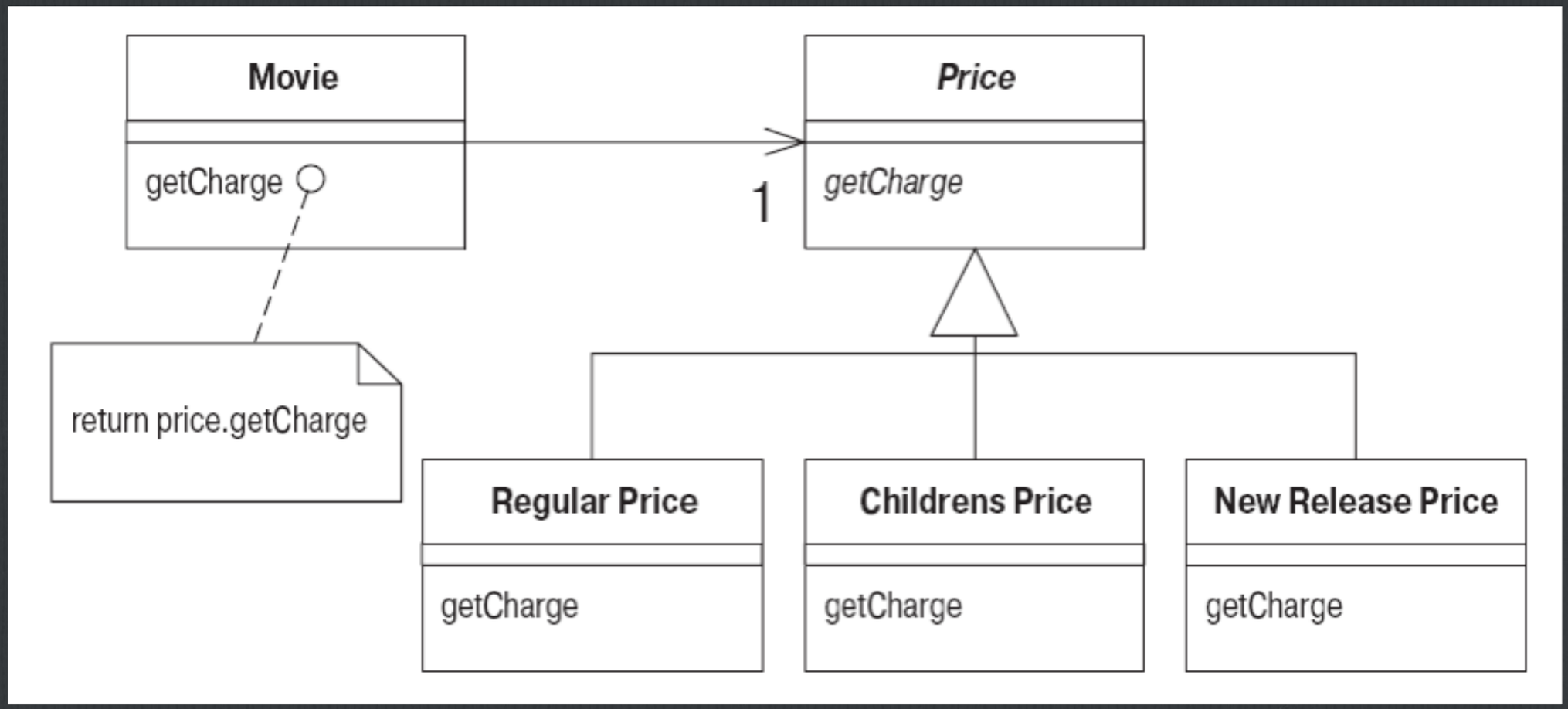
Диаграмма последовательности после удаления временных переменных



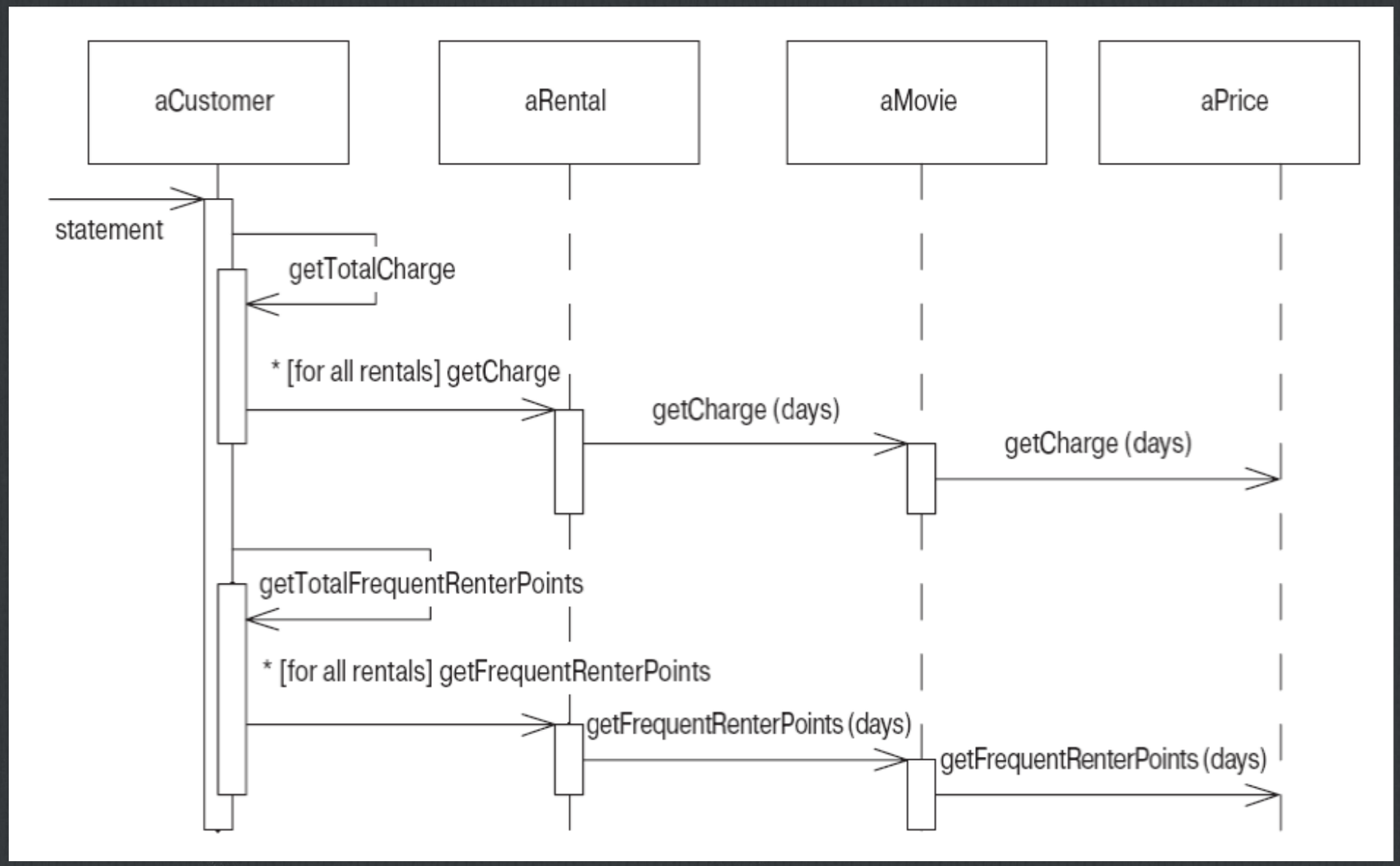
Рефакторинг 2: Замена условной логики наследованием



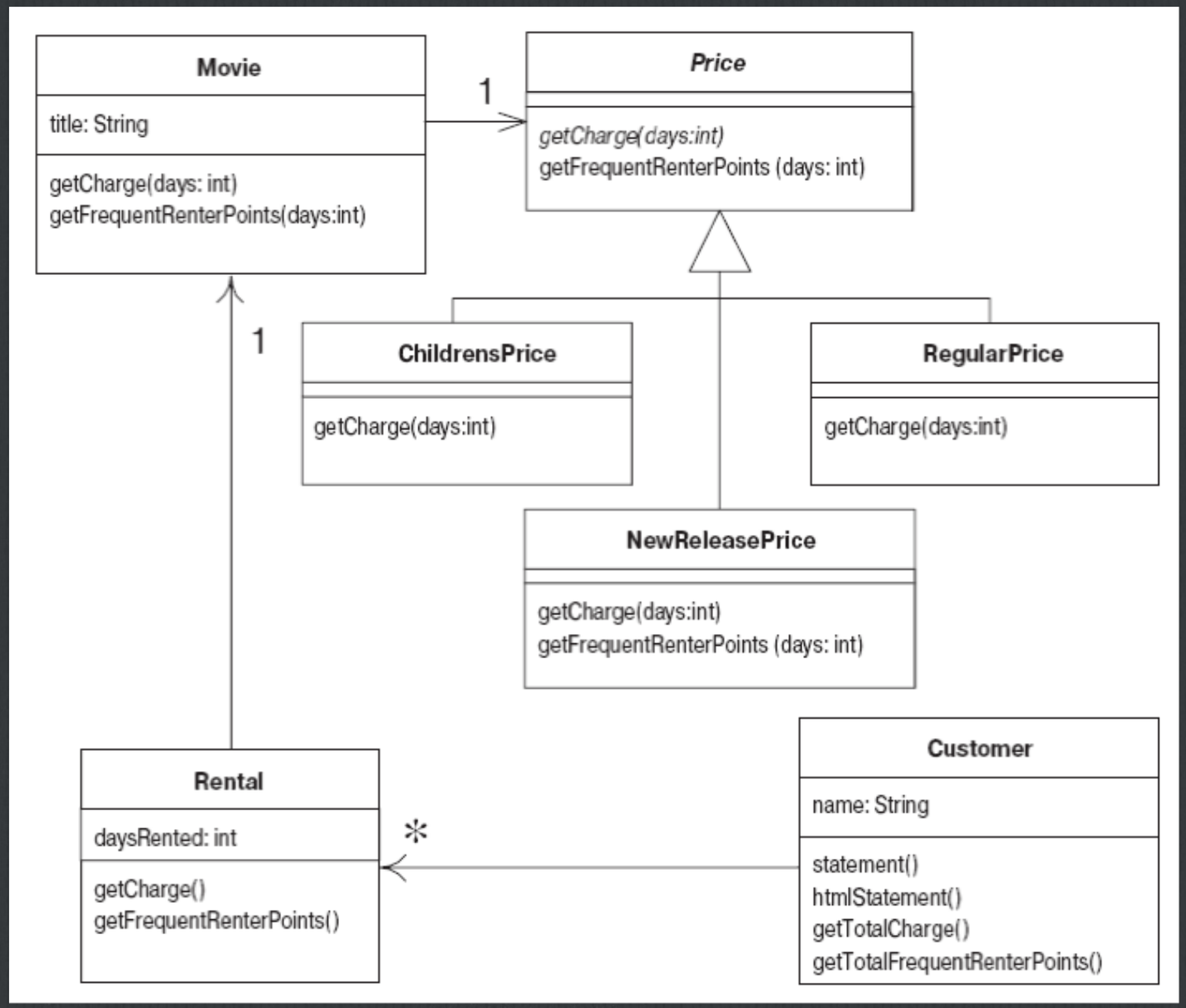
Паттерн проектирования State



Результат: Диаграмма последовательности



Результат: Диаграмма классов



Ценность рефакторинга

- ☐ Рефакторинг улучшает композицию ПО
- ☐ Рефакторинг облегчает понимание ПО
- ☐ Рефакторинг помогает найти ошибки
- ☐ Рефакторинг позволяет быстрее писать программы

Эволюция ПО

- улучшается или ухудшается качество после изменений
- изменения на стадии реализации или сопровождения

(Разумные) Причины выполнения рефакторинга

- ☐ код повторяется
- ☐ метод слишком большой
- ☐ цикл слишком велик, много вложений
- ☐ класс имеет плохую связность
- ☐ интерфейс класса слишком размыт
- ☐ метод принимает слишком много параметров

(Разумные) Причины выполнения рефакторинга

- ☐ при простых изменениях надо менять несколько классов
- ☐ нужно параллельно изменять несколько иерархий
- ☐ нужно параллельно изменять несколько блоков CASE
- ☐ родственные элементы данных не организован в классы
- ☐ метод использует больше элементов чужого класса
- ☐ метод имеет неудачное имя

(Разумные) Причины выполнения рефакторинга

- ☐ поля классов открыты
- ☐ потомки не испольхуют методы предков
- ☐ в коде много комментариев
- ☐ код содержит глобальные переменные
- ☐ перед/после вызова метода нужно вызывать что-то ещё
- ☐ программа содержит код, который потом понадобиться

(Разумные) Причины выполнения рефакторинга

- ☐ бродячие данные
- ☐ перегрузка элементарного типа
- ☐ объект-посредник ничего не делает
- ☐ у класса слишком много функций
- ☐ у класса слишком мало функций
- ☐ один класс слишком много знает о другом

Когда не нужно

- ☐ много багов
- ☐ поджимают сроки
- ☐ проще написать код с нуля

Когда уместно

- [ ] при добавлении нового функционала
- [ ] при исправлении ошибки
- [ ] при проведении ревью

Уровни рефакторинга

- ☐ Уровень данных
- ☐ Уровень отдельных операторов
- ☐ Уровень отдельных методов
- ☐ Уровень реализации классов
- ☐ Уровень интерфейсов классов
- ☐ Уровень системы