Тема 15. Шаблоны рефакторинга

1. Применение
2. Каталог
3. Инструменты

Признаки плохого кода

- ☐ Дублирование кода
- ☐ Длинный метод
- ☐ Большой класс
- ☐ Длинный список параметров

Признаки плохого кода

- ☐ Расходящиеся модификации
- ☐ “Стрельба дробью”
- ☐ “Завистливые” функции
- ☐ Группы данных
- ☐ “Одержимость” элементарными типами

Признаки плохого кода

- ☐ SWITCH
- ☐ Параллельные иерархии наследования
- ☐ Ленивый класс
- ☐ Теоретическая общность
- ☐ Временное поле

Признаки плохого кода

- ☐ Цепочки сообщений
- ☐ Посредник
- ☐ “Неуместная близость”
- ☐ Альтернативные классы с разными интерфейсами

Признаки плохого кода

- ☐ Неполнота библиотечного класса
- ☐ Классы данных
- ☐ Отказ от наследства
- ☐ Комментарии

Признаки плохого кода

- ☐ Сложность условий
- ☐ “Неприличная демонстрация”
- ☐ “Расплывшееся” решение
- ☐ “Комбинаторный взрыв”
- ☐ “Оригинальное” решение

Структура шаблона рефакторинга

- ☐ Название (name)
- ☐ Краткая сводка (summary)
- ☐ Мотивировка (motivation)
- ☐ Техника (mechanics)
- ☐ Примеры (examples)
- ☐ *Разновидности (variations)

Группы шаблонов рефакторинга

- ☐ составление методов
- ☐ перемещение функций между объектами
- ☐ организация данных
- ☐ упрощение условных выражений
- ☐ упрощение вызовов методов
- ☐ решение задач обобщения

Рефакторинг и паттерны

- ☐ простой рефакторинг
- ☐ рефакторинг к паттерну
- ☐ рефакторинг по направлению к паттерну
- ☐ рефакторинг с отказом от паттерна

Составление методов

Выделение метода (Extract Method)

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

    // вывод деталей
    System.out.println ("name:       " + _name);
    System.out.println ("amount      " + amount);
}
```

```
void printOwing(double amount) {
    printBanner();
    printDetails(amount);
}

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

Summary:

Преобразовать фрагмент кода в метод, название которого поясняет его назначение.

Мотивация

1. Короткие методы с осмысленными именами понятнее и удобнее.

2. Такие методы могут использоваться другими методами.

Техника

1. Создать новый метод и назвать его соответственно назначению.
2. Скопировать код, подлежащий выделению.
3. Найти в извлечённом коде все обращения к локальным переменным исходного метода – локальные переменные и параметры выделенного метода.
4. Найти временные переменные внутри выделенного кода – временные переменные выделенного метода .
5. Если переменные модифицируются – использовать вызов других методов или расщепление переменной (шаблоны Replace Temp with Query и Split Temporary Variable).
6. Заменить в исходном методе код вызовом метода.
7. Выполнить компиляцию.
8. Протестировать.

Встраивание метода (Inline Method)

```
int getRating() {
    return (moreThanFiveLateDeliveries()) ? 2 : 1;
}
boolean moreThanFiveLateDeliveries() {
    return _numberOfLateDeliveries > 5;
}
```

```
int getRating() {
    return (_numberOfLateDeliveries > 5) ? 2 : 1;
}
```

Summary:
Поместить тело метода в код, который его вызывает и удалить метод.

Мотивация:
1. Удаление излишней косвенности.
2. Упрощение структуры кода.

Замена временной переменной вызовом метода (Replace Temp with Query)

```
double basePrice = _quantity * _itemPrice
if (basePrice > 1000)
    return basePrice * 0.95;
else
    return basePrice * 0.98;
```

```
if (basePrice() > 1000)
    return basePrice() * 0.95;
else
    return basePrice() * 0.98;


...
  double basePrice() {
      return _quantity * _itemPrice;
}
```

Summary:
Преобразовать выражение в метод. Заменить все ссылки на временные переменные вызовом метода. Новый метод может быть использован в других методах.

Мотивация и техника

Мотивация:

1. Упростить доступ к данным для других методов.
2. Применение в других рефакторингах.

Техника:

1. Найти простую переменную с одним присваиванием.
2. Выделить правую часть присваивания в метод.
3. Скомпилировать.
4. Протестировать.

Как мы код не пишем

Расщепление временной переменной

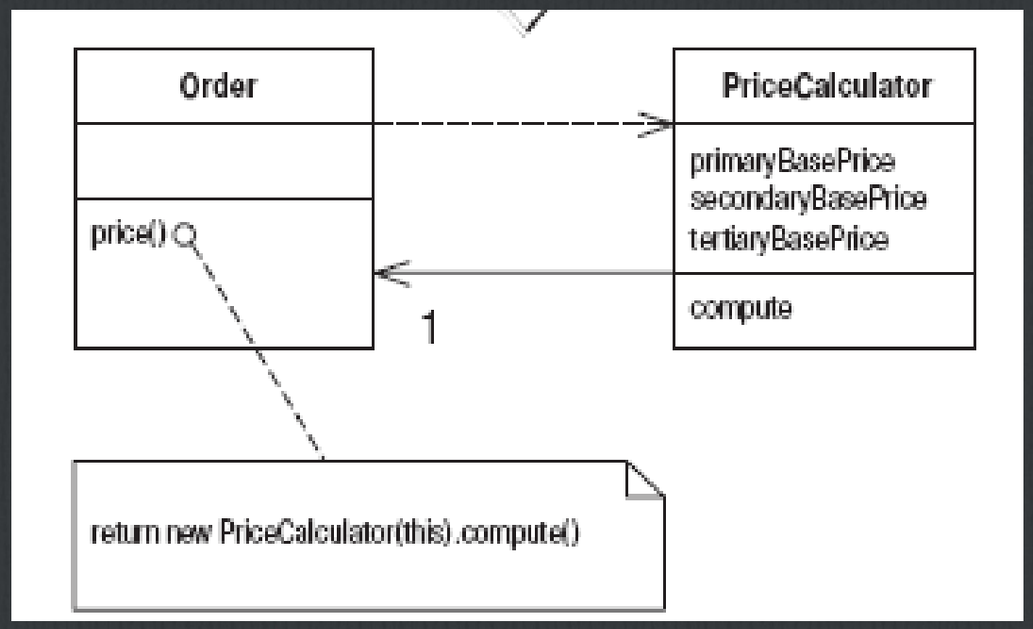
```
double temp = 2 * (_height + _width);
System.out.println (temp);
temp = _height * _width;
System.out.println (temp);
```

Удаление присваиваний параметрам

```
int discount (int inputVal, int quantity, int yearToDate) {
    if (inputVal > 50) inputVal -= 2;
```

Замена метода объектом методов (Replace Method with Method Object)

```
class Order...
  double price() {
      double primaryBasePrice;
      double secondaryBasePrice;
      double tertiaryBasePrice;
      // длинные вычисления;
      ...
}
```



Summary:
Преобразовать метод в отдельный объект так, чтобы локальные переменные стали полями этого объекта. После этого разложить метод на несколько методов того же объекта.

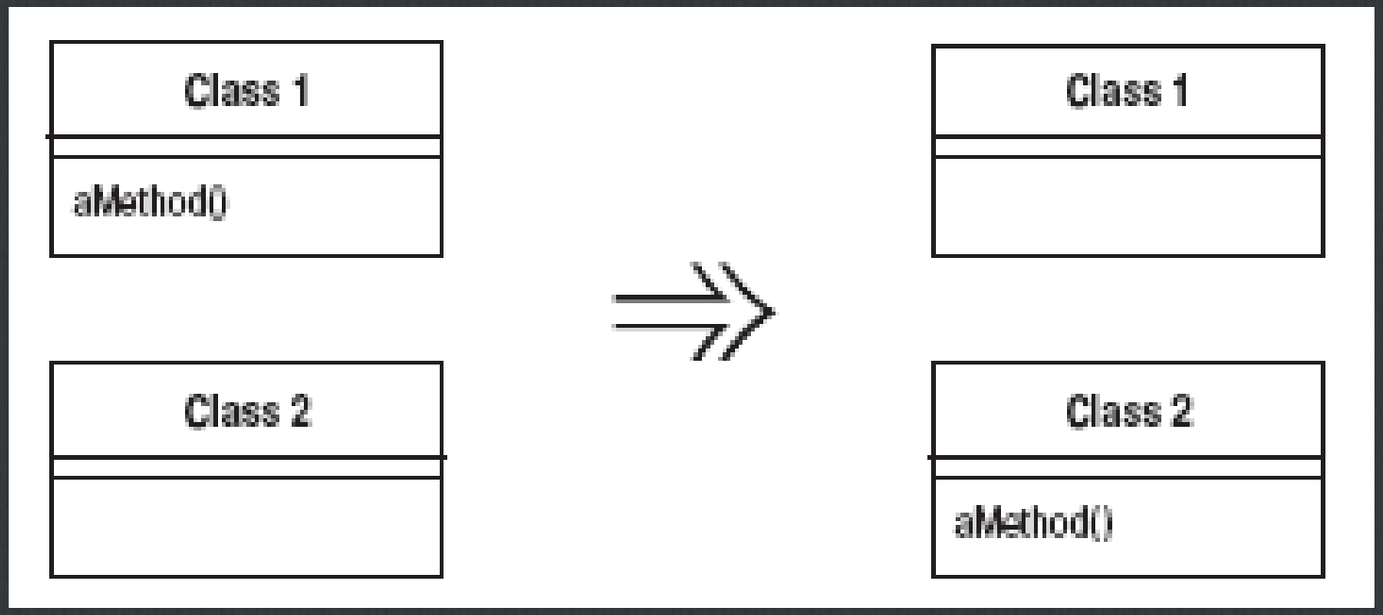
Мотивация и техника

Такая же, как у «Выделения метода» + превращает локальные переменные в поля объекта методов.

1. Создать новый класс и назвать его так же, как метод.
2. Создать в новом классе поле для объекта-владельца исходного метода и поля для всех временных переменных и параметров.
3. Создать конструктор, принимающий исходный объект и все параметры.
4. Скопировать в метод нового класса тело исходного метода.
5. Заменить старый метод созданием нового объекта и вызовом нового метода.
6. Скомпилировать, протестировать.

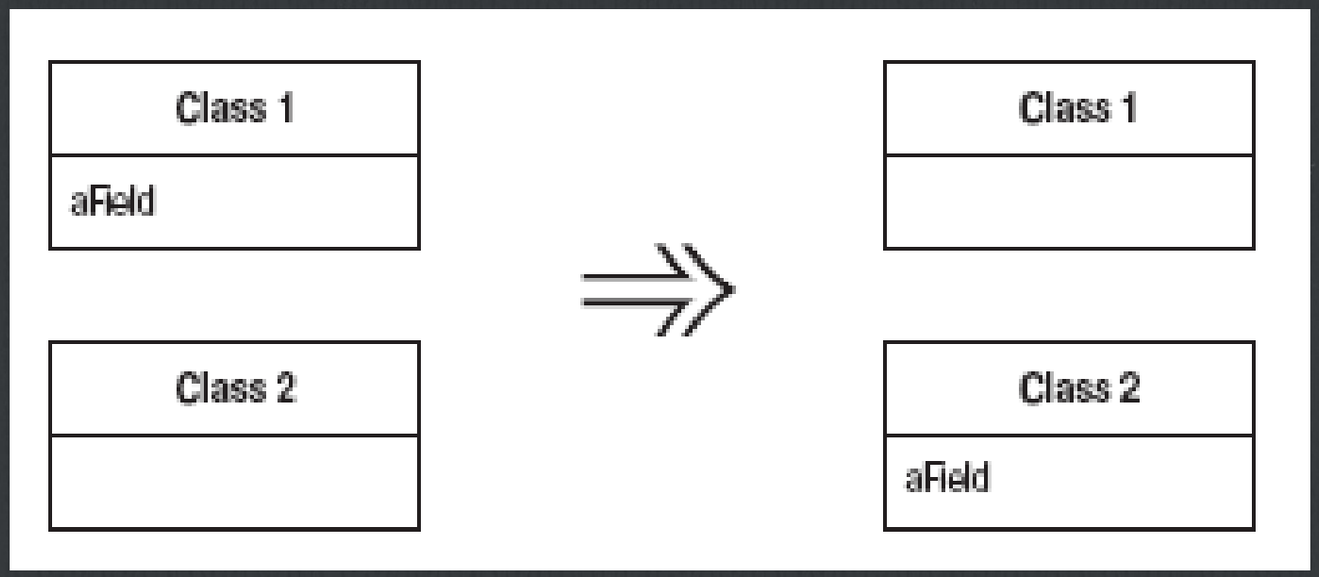
Перемещение графиков функций

Перемещение метода (Move Method)



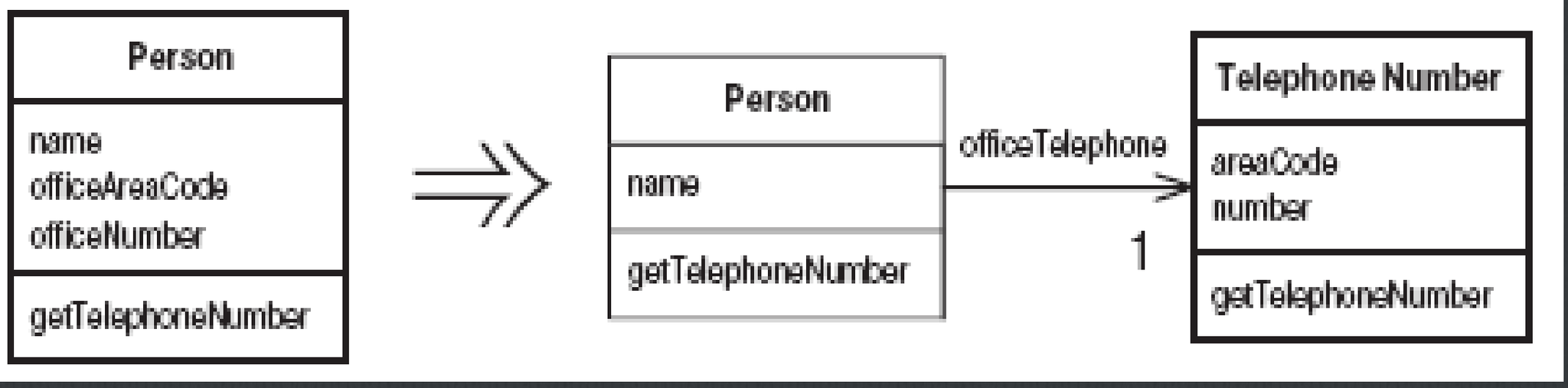
Summary:
Создать новый метод с аналогичным телом в том классе, который чаще всего им используется. Заменить тело прежнего метода простым делегированием или вообще удалить .

Перемещение поля (Move Field)



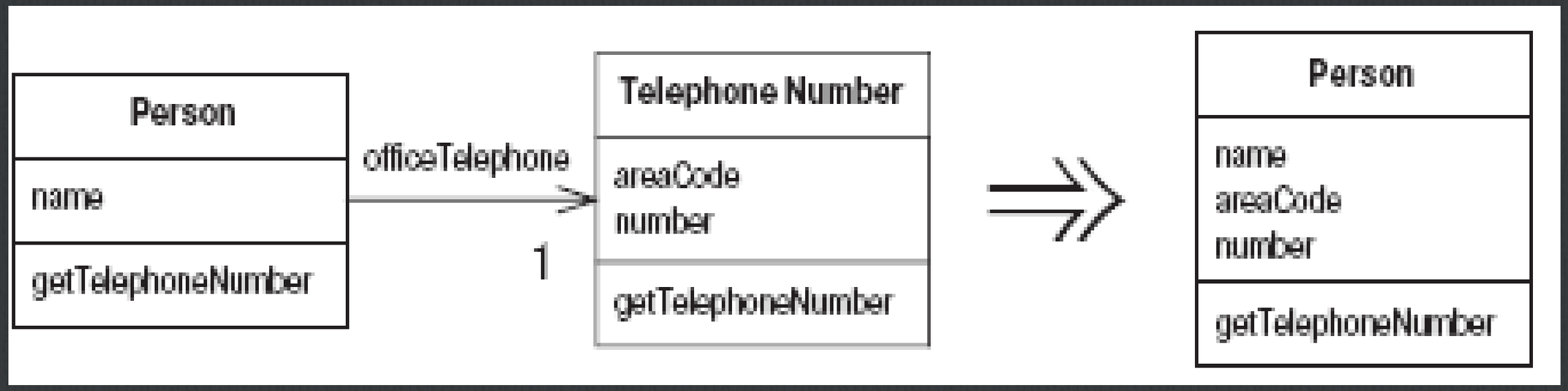
Summary:
Создать в целевом классе новое поле и отредактировать всех его пользователей

Выделение класса (Extract Class)



Summary:
Создать новый класс и переместить соответствующие поля и методы из старого класса в новый.

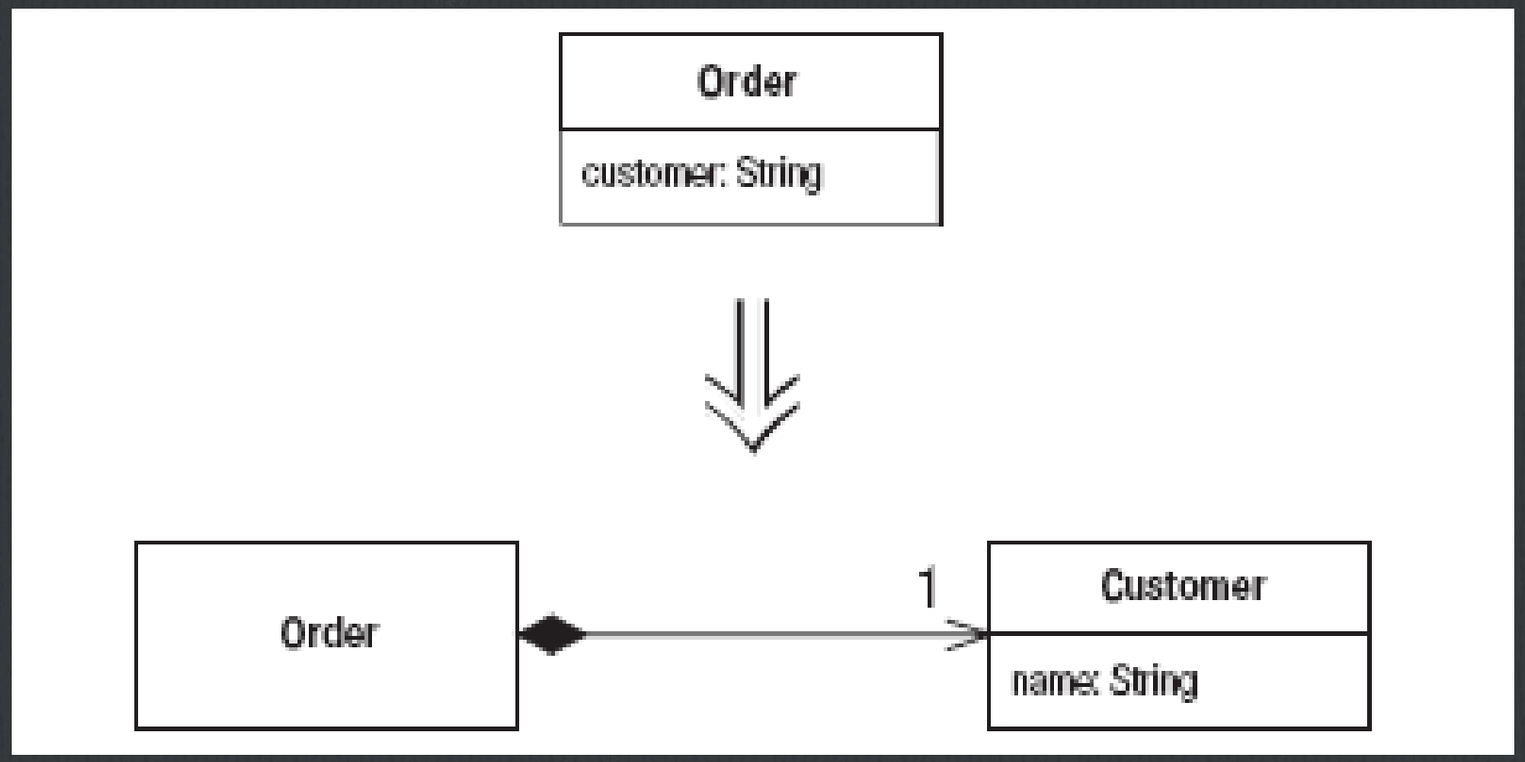
Встраивание класса (Inline Class)



Summary:
Переместить все функции в другой класс и удалить исходный.

Организация данных

Замена значения данных объектом (Replace Data Value with Object)

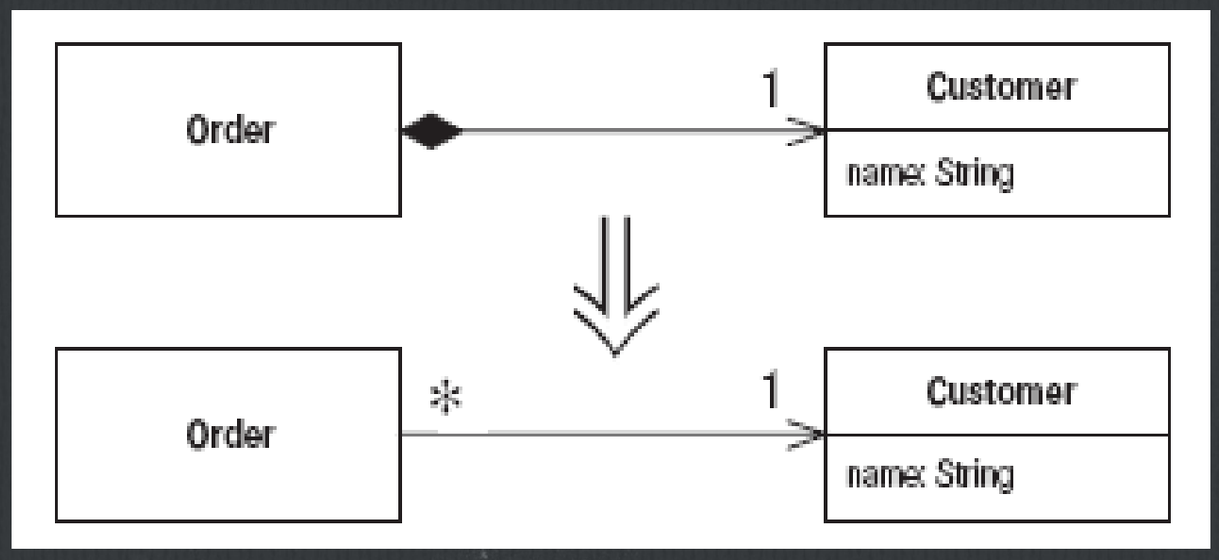


Summary:
Преобразовать элемент данных в объект

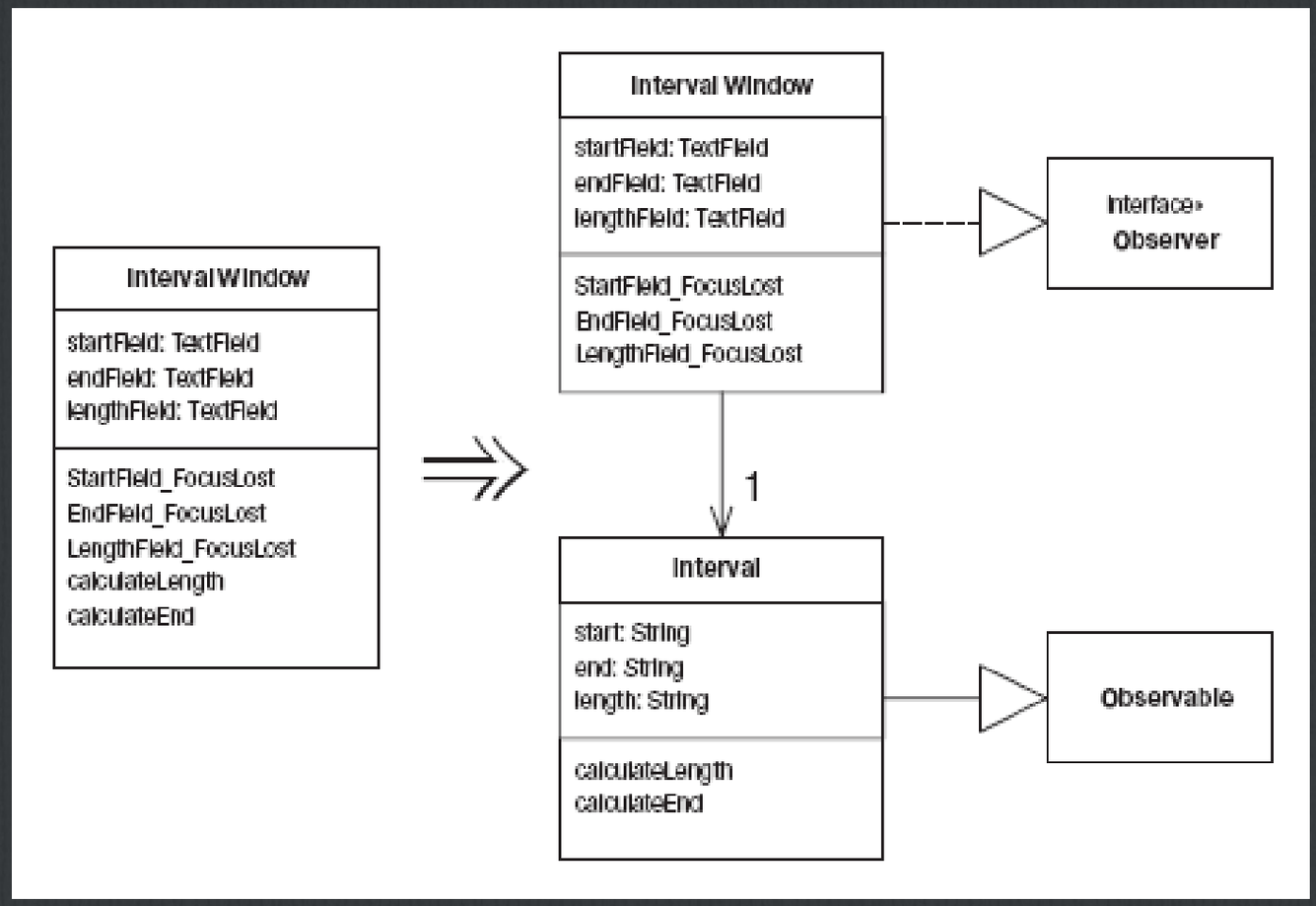
Замена значения данных объектом

Мотивация:
В процессе разработки некоторые, казавшиеся сначала простыми данные, усложняются (например, адрес).

Техника:
Может потребоваться применение шаблона «Замена значения ссылкой»

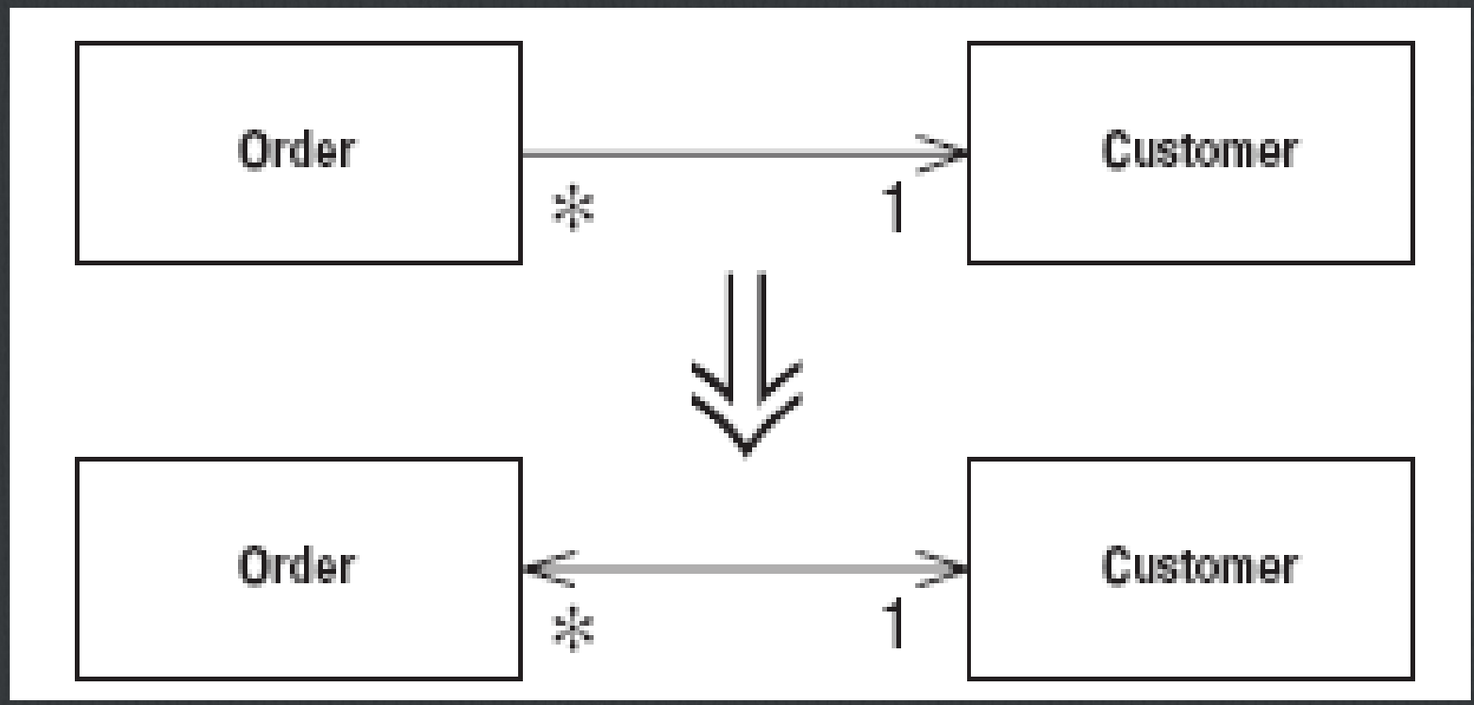


Дублирование видимых данных (Duplicate Observed Data)



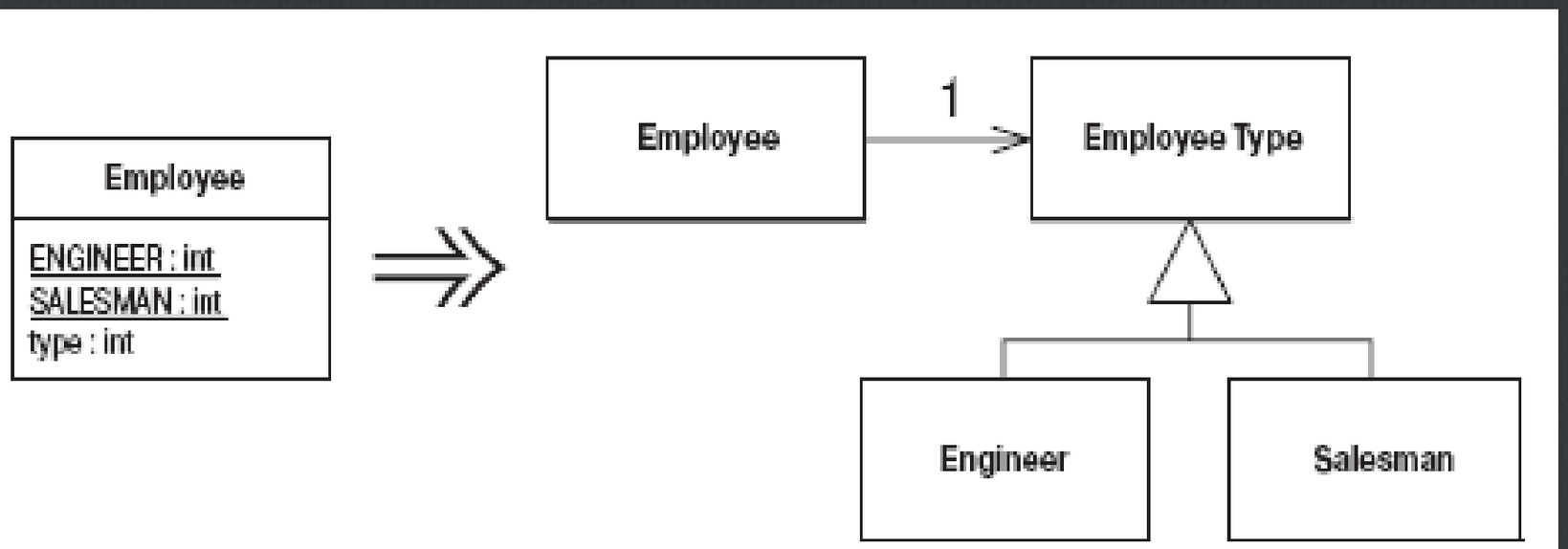
Summary:
Скопировать данные в объект предметной области приложения .
Создать объект-наблюдатель для обеспечения синхронности данных.

Замена однонаправленной связи двунаправленной (Change Unidirectional Association to Bidirectional)



Summary:
Добавить обратные указатели и изменить модификаторы, чтобы они обновляли оба набора.

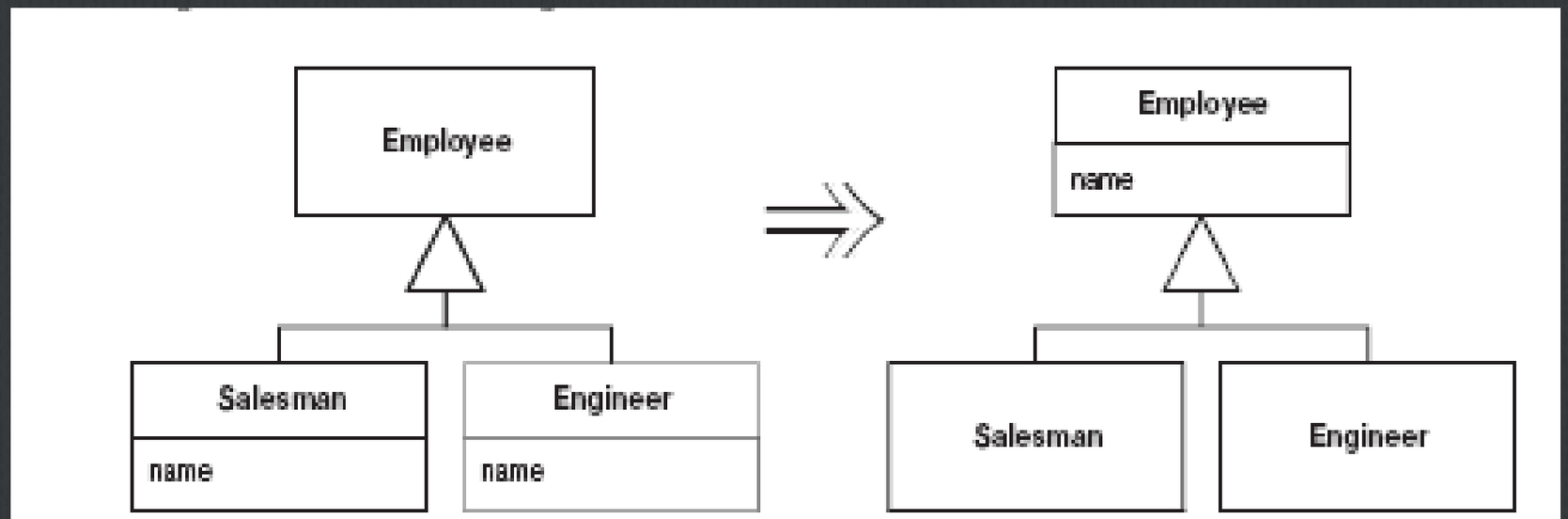
Замена кода типа состоянием/ стратегией (Replace Type Code with State/Strategy)



Summary:
Заменить код типа объектом состояния.

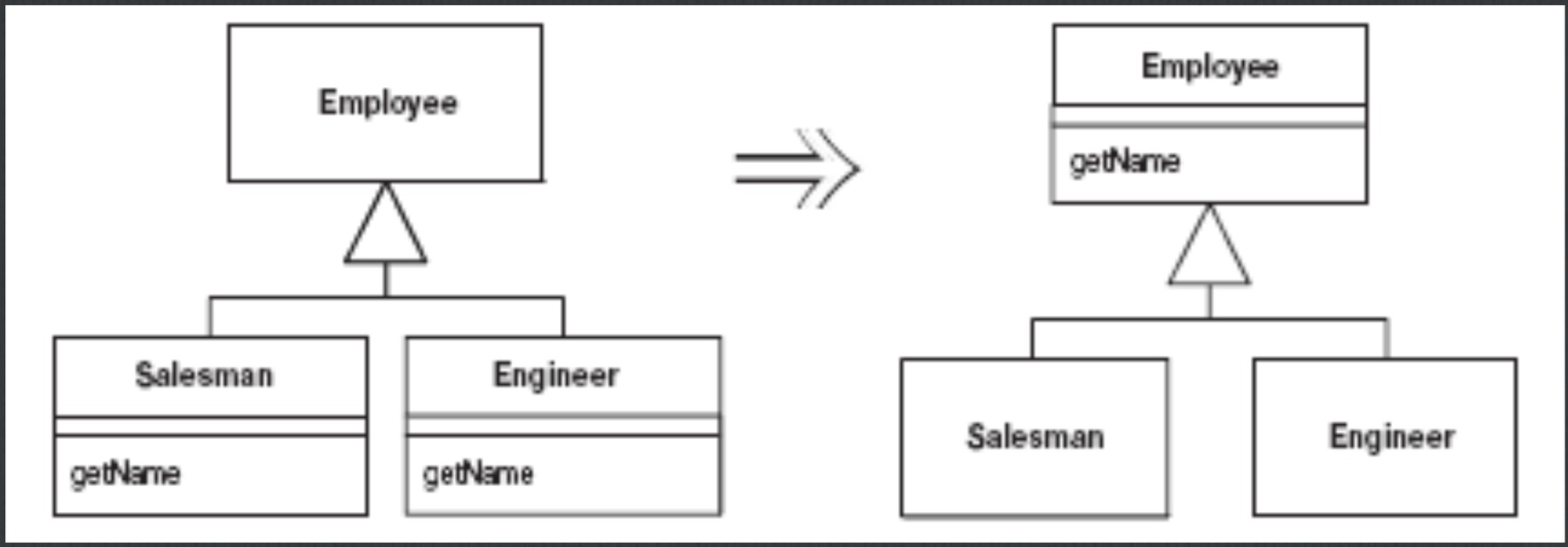
Решение задач обобщения

Подъём поля (Pull Up Field)



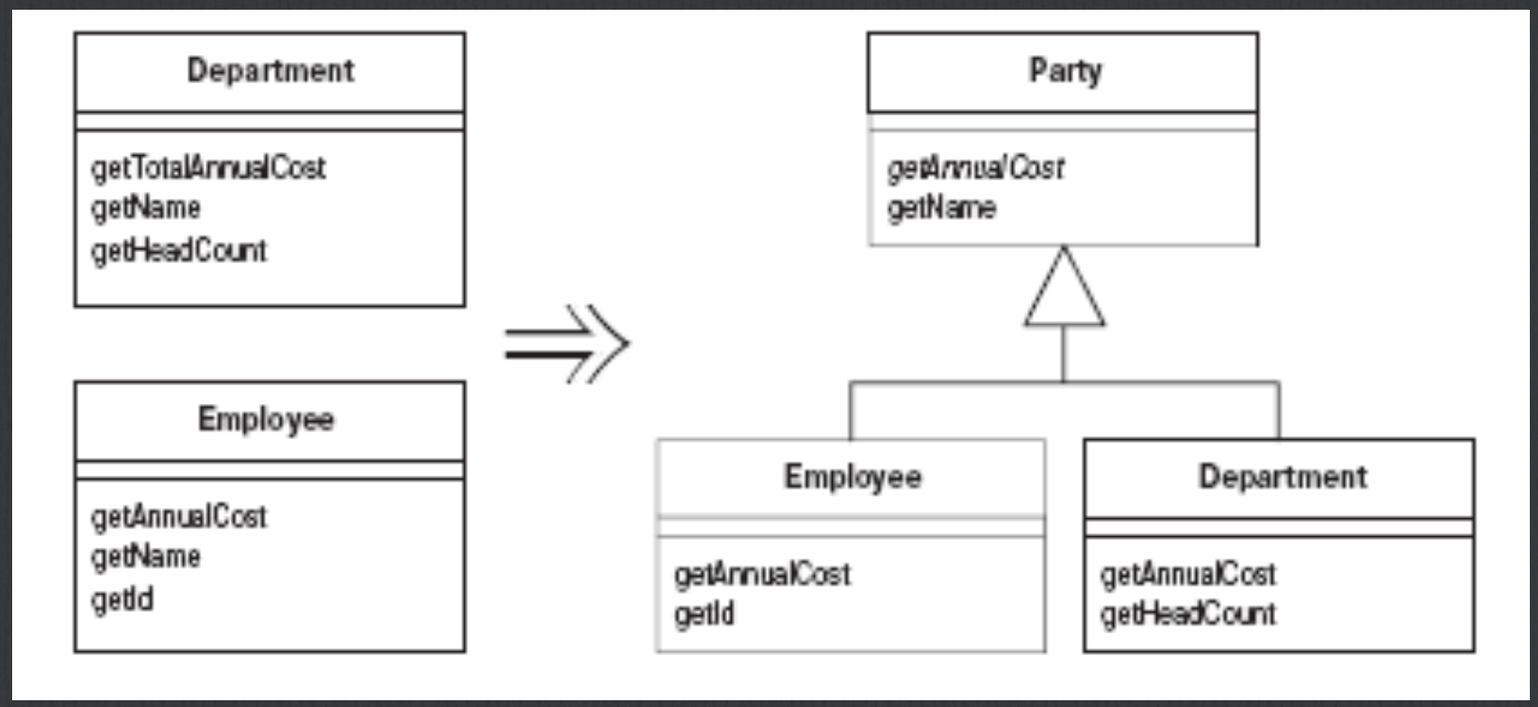
Summary:
Переместить поле в родительский класс, если в двух подклассах есть одинаковое поле.

Подъём метода (Pull Up Method)



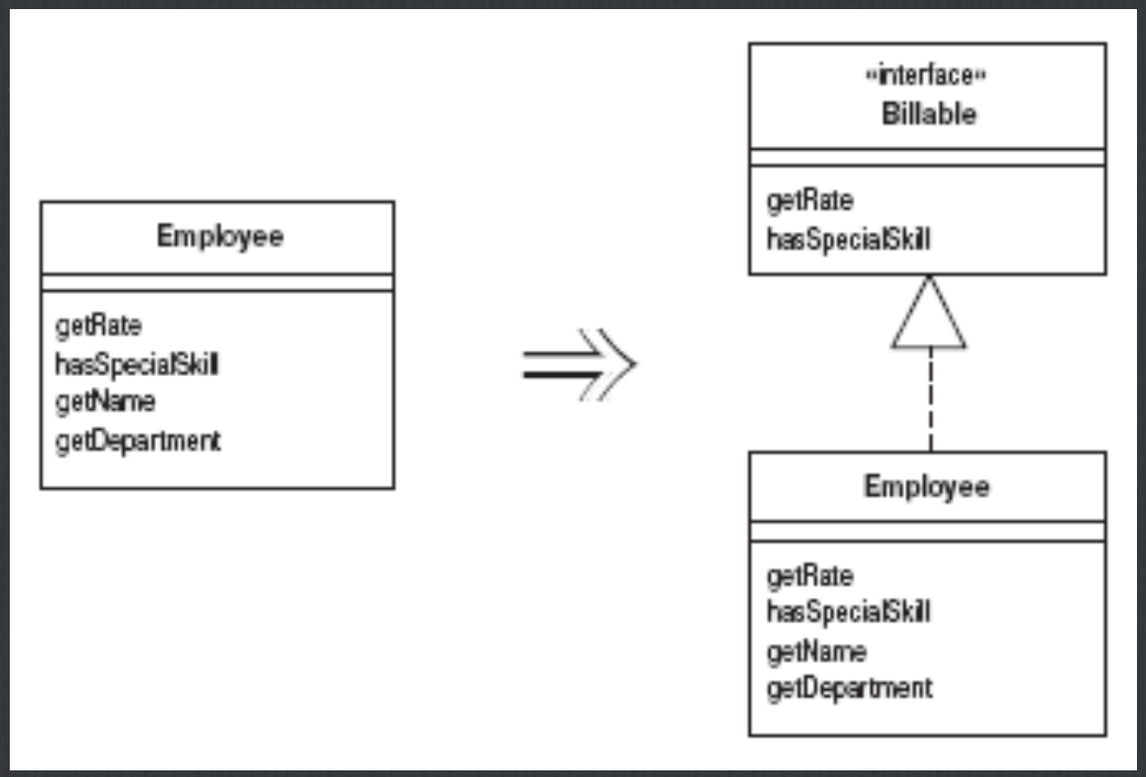
Summary:
Переместить методы с идентичными результатами из подклассов в родительский класс.

Выделение родительского класса (Extract Superclass)



Summary:
Для классов со сходными функциями выделить общий родительский класс и переместить в него общие методы.

Выделение интерфейса (Extract Interface)



Summary:
Подмножество интерфейса классов, являющееся общим выделить в интерфейс.

Упрощение условных выражений

Упрощение условных выражений

if (cond) true else false

- ☐ Декомпозиция условного оператора
- ☐ Консолидация условного выражения, дублирующихся фрагментов
- ☐ Замена вложенных условных операторов граничными

Упрощение вызовов методов

Упрощение вызовов методов

- ☐ Переименование метода
- ☐ Добавление/удаление параметра
- ☐ Параметризация метода
- ☐ Замена параметра явным методом

Крупные рефакторинги

- ☐ Разделение наследования
- ☐ Преобразование процедурного проекта в объектный
- ☐ Отделение предметной области от представления
- ☐ Выделение иерархии

Альтернативная классификация

- ☐ создание объектов
- ☐ упрощение
- ☐ обобщение
- ☐ защита кода
- ☐ накопление информации
- ☐ утилиты

Функциональное программирование

- ☐ Отделение побочных эффектов
- ☐ Рекурсия и хвостовая рекурсия
- ☐ Массовые операции
- ☐ Отложенные вычисления

Инструменты рефакторинга

- ☐ IntellIiJ IDEA
- ☐ Eclipse
- ☐ Visual Studio
- ☐ Visual Assist
- ☐ ReSharper, CodeRush, JustCode
- ☐ DMS Software Reengineering Toolkit