Ращупкин ЕВ КЭ-120  
Практика 2

Рефакторинг кода

Задания типа «По картинке – название». Вам необходимо дать название рефакторинга, который продемонстрирован на изображении.

1. Есть подклассы, которые различаются только в методах, возвращающих константы. Перенести эти методы в суперкласс, сделав их полями, и удалить подклассы. Если подклассы различаются только в методах, возвращающих константы, то нет смысла загромождать структуру классов, а надо объявить поле в суперклассе и добавить аксессор.

**Замена подкласса полем**

1. Есть метод, который возвращает значение и изменяет состояние объекта. Создать два метода – один для возврата значения, другой для изменения состояния. Хорошим тоном является то, что методы, возвращающие значения, вообще не меняют наблюдаемое состояние объекта. То есть они могут изменять разные кэши, призванные ускорять работу, но не несущие смысловой нагрузки.

**Разделение запроса и модификатора**

1. Есть два метода в подклассах, которые выполняют похожие шаги в одинаковом порядке, хотя сами шаги разные. Поместить шаги в методы с одинаковыми сигнатурами так, чтобы исходные методы стали одинаковыми. Затем исходные методы поместить в суперкласс. Как обычно, убирается дублирование кода с помощью наследования – общая структура выносится в суперкласс, частности описываются в подклассах и включаются в работу благодаря полиморфизму.

**Создание шаблонного метода**

1. Используется делегирование, и приходится писать делегирующие методы ко всему интерфейсу. Заменить класс со ссылкой на подкласс. Не надо использовать, если не все методы делегируются, так как интерфейс подкласса должен полностью поддерживать интерфейс суперкласса. Здесь можно использовать [Избавление от промежуточного сервера](http://codingcraft.ru/resources/REFACTORING/RemoveMiddleMan.htm), чтобы клиенты могли сами вызывать делегирующий класс. Можно использовать [Извлечение суперкласса](http://codingcraft.ru/resources/REFACTORING/ExtractSuperclass.htm) для отделения общего интерфейса и наследования от получившегося класса. Таким же образом можно использовать [Извлечение интерфейса](http://codingcraft.ru/resources/REFACTORING/ExtractInterface.htm). Другая ситуация, при которой этот рефакторинг не используется, когда делегат изменяем и разделен между несколькими объектами. Тогда при наследовании каждый подкласс получит свою копию и разделение на этом кончится. При неизменяемом делегате это не является препятствием.

**Замена делегирования наследованием**

1. Есть проверки на значение Null. Заменить значение Null на объект Null. Преимущество полиморфизма состоит в том, что вместо того, чтобы узнавать, какого типа данный объект, а потом пользоваться его методами в зависимости от ответа, можно напрямую вызывать его методы, не заботясь о его типе. Это касается и пустого объекта (Null объекта), который будет корректно обрабатывать вызовы в отсутствие реального объекта.

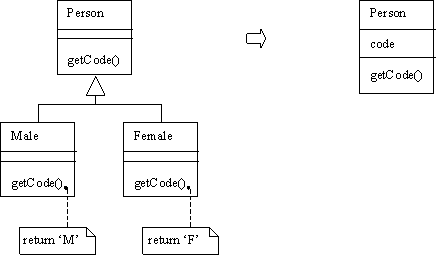
**Введение объекта Null**

1. Класс производит слишком простое делегирование. Заставить клиента обращаться к делегирующему классу напрямую. Цена за промежуточный сервер – необходимость каждый раз добавлять делегирующий метод на сервер при необходимости клиента использовать новый метод на делегирующем объекте. При частой смене методов на делегирующем объекте это неудобно.

**Удаление посредника**

Задания типа «По картинке – код». Вам необходимо написать кусочек кода, демонстрирующий заданный картинкой запах, и кусочек кода, демонстрирующий заданный картинкой рефакторинг.

1)



До

class Person {

String getCode() {

return "";

}

}

class Male extends Person {

@Override

String getCode() {

return "M";

}

}

class Female extends Person {

@Override

String getCode() {

return "F";

}

}

После

class Person {

private final String code;

public Person(String code) {

this.code = code;

}

public String getCode() {

return code;

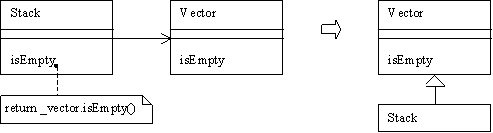
}

}

Person male = new Person("M");

Person female = new Person("F");

4)



До

class Vector {

public boolean isEmpty() {

// something

return true;

}

}

class Stack {

private Vector vector;

public Stack(Vector vector) {

this.vector = vector;

}

public boolean isEmpty() {

return vector.isEmpty();

}

}

После

class Vector {

public boolean isEmpty() {

// something

return true;

}

}

class Stack {

private Vector vector;

public Stack(Vector vector) {

this.vector = vector;

}

}