Лабораторная работа № 2

Рефакторинг программного кода. Перемещение функций между объектами

1. Цель работы

Исследовать эффективность перемещения функций между объектами при рефакторинге программного кода. Получить практические навыков применения приемов рефакторинга объектно-ориентированных программ.

2. Общие положения

2.1. Обзор методов перемещения функций между объектами

Решение о том, где разместить выполняемые функции, является одним из наиболее фундаментальных решений, принимаемых при проектировании объектов. Достоинство рефакторинга заключается в том, что он позволяет изменить решение.

Часто такие проблемы решаются просто с помощью «Перемещения метода» *(MoveMethod)* или «Перемещения поля» *(Move Field)*. Если надо выполнить обе операции, то предпочтительнее начать с «Перемещения поля» *(Move Field)*.

Часто классы перегружены функциями. Тогда применяется «Выделение класса» *(Extract Class)*, чтобы разделить эти функции на части. Если некоторый класс имеет слишком мало обязанностей, с помощью «Встраивания класса» *(Inline Class)* его следует присоединить к другому классу. Если функции класса на самом деле выполняются другим классом, часто удобно скрыть этот факт с помощью «Сокрытия делегирования» *(Hide Delegate)*. Иногда сокрытие класса, которому делегируются функции, приводит к постоянным изменениям в интерфейсе класса, и тогда следует воспользоваться «Удалением посредника» *(Remove Middle Man)*.

Два последних рефакторинга, относящихся к этому разделу, «Введение внешнего метода» *(Introduce Foreign Method)* и «Введение локального расширения» *(Introduce Local Extension)*, представляют собой особые случаи. Их следует использовать только тогда, когда недоступен исходный код класса, но переместить функции в класс, который нет возможности модифицировать, тем не менее, надо. Если таких методов всего один-два, применяется «Введение внешнего метода» *(Introduce Foreign Method)*, если же методов больше, то используется «Введение локального расширения» *(Introduce Local Extension)*.

2.2. Приемы рефакторинга

2.2.1. Перемещение метода (Move Method)

Метод чаще использует функции другого класса (или используется ими), а не того, в котором он определен – в данное время или, возможно, в будущем.

Создайте новый метод с аналогичным телом в том классе, который чаще всего им используется. Замените тело прежнего метода простым делегированием или удалите его вообще.

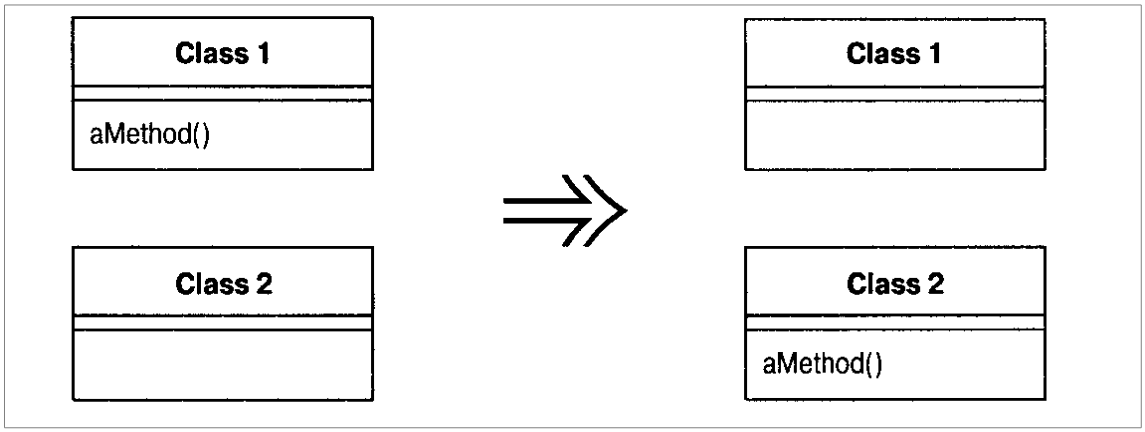

Рисунок 2.1 – Перемещение метода

Мотивация

Перемещение методов – это насущный хлеб рефакторинга. Методы перемещаются, если в классах сосредоточено слишком много функций или когда классы слишком плотно взаимодействуют друг с другом и слишком тесно связаны. Перемещая методы, можно сделать классы проще и добиться более четкой реализации функций.

Обычно просматриваются методы класса с целью обнаружить такой, который чаще обращается к другому объекту, чем к тому, в котором сам располагается. Это полезно делать после перемещения каких-либо полей. Найдя подходящий для перемещения метод, нужно рассмотреть, какие методы вызывают его, какими методами вызывается он сам, и найти переопределяющие его методы в иерархии классов. Следует определить, стоит ли продолжить работу, взяв за основу объект, с которым данный метод взаимодействует теснее всего.

Принять решение не всегда просто. Если нет уверенности в необходимости перемещения данного метода, лучше перейти к рассмотрению других методов. Часто принять решение об их перемещении проще. Фактически особой разницы нет. Если принять решение трудно, то, вероятно, оно не столь уж важно.

Техника

- Изучите все функции, используемые исходным методом, которые определены в исходном классе, и определите, не следует ли их также переместить.

Если некоторая функция используется только тем методом, который вы собираетесь переместить, ее тоже вполне можно переместить. Если эта функция используется другими методами, посмотрите, нельзя ли и их переместить. Иногда проще переместить сразу группу методов, чем перемещать их по одному.

- Проверьте, нет ли в подклассах и родительских классах исходного класса других объявлений метода.

Если есть другие объявления, перемещение может оказаться невозможным, пока полиморфизм также не будет отражен в целевом классе.

- Объявите метод в целевом классе. Можете выбрать для него другое имя, более оправданное для целевого класса.
- Скопируйте код из исходного метода в целевой. Приспособьте метод для работы в новом окружении.

Если методу нужен его исходный объект, необходимо определить способ ссылки на него из целевого метода. Если в целевом классе нет соответствующего механизма, передайте новому методу ссылку на исходный объект в качестве параметра.

Если метод содержит обработчики исключительных ситуаций, определите, какому из классов логичнее обрабатывать исключительные ситуации. Если эту функцию следует выполнять исходному классу, оставьте обработчики в нем.

- Выполните компиляцию целевого класса.
- Определите способ ссылки на нужный целевой объект из исходного.

Поле или метод, представляющие целевой объект, могут уже существовать. Если нет, посмотрите, трудно ли создать для этого метод. При неудаче надо создать в исходном объекте новое поле, в котором будет храниться ссылка на целевой объект. Такая модификация может стать постоянной или сохраниться до тех пор, пока рефакторинг не позволит удалить этот объект.

- Сделайте из исходного метода делегирующий метод.
- Выполните компиляцию и тестирование.
- Определите, следует ли удалить исходный метод или сохранить его как делегирующий свои функции.

Проще оставить исходный метод как делегирующий, если есть много ссылок на него.

- Если исходный метод удаляется, замените все обращения к нему обращениями к созданному методу.

Выполнять компиляцию и тестирование можно после каждой ссылки, хотя обычно проще заменить все ссылки сразу путем поиска и замены.

- Выполните компиляцию и тестирование.

2.2.2. Перемещение поля (Move Field)

Поле используется или будет использоваться другим классом чаще, чем классом, в котором оно определено.

Создайте в целевом классе новое поле и отредактируйте всех его пользователей.

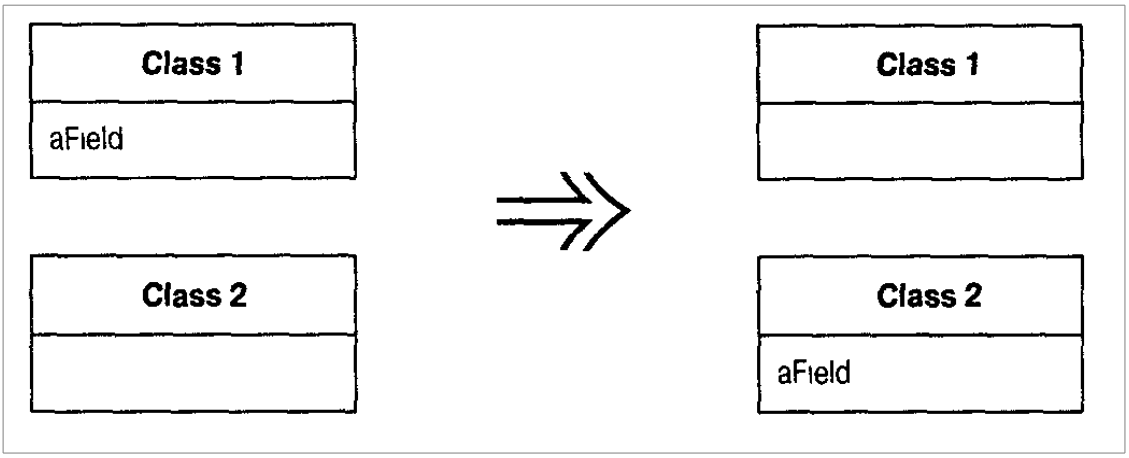

Рисунок 2.2 – Перемещение поля

Мотивация

Перемещение состояний и поведения между классами составляет самую суть рефакторинга. По мере разработки системы выясняется необходимость в новых классах и перемещении функций между ними. Разумное и правильное проектное решение через неделю может оказаться неправильным. Но проблема не в этом, а в том, чтобы не оставить это без внимания.

Возможность перемещения поля следует рассмотреть, если его использует больше методов в другом классе, чем в своем собственном. Использование может быть косвенным, через методы доступа. Можно принять решение о перемещении методов, что зависит от интерфейса. Но если представляется разумным оставить методы на своем месте, следует перемещать поле.

Другим основанием для перемещения поля может быть осуществление «Выделения класса» *(Extract Class)*. В этом случае сначала перемещаются поля, а затем методы.

Техника

- Если поле открытое, выполните «Инкапсуляцию поля» *(Encapsulate Field)*.

Если вы собираетесь переместить методы, часто обращающиеся к полю, или есть много методов, обращающихся к полю, может оказаться полезным воспользоваться «Самоинкапсуляцией поля» (Self Encapsulate Field).

- Выполните компиляцию и тестирование.
- Создайте в целевом классе поле с методами для чтения и установки значений.
- Скомпилируйте целевой класс.

2
2

- Определите способ ссылки на целевой объект из исходного.

Целевой класс может быть получен через уже имеющиеся поля или методы. Если нет, посмотрите, трудно ли создать для этого метод. При неудаче следует создать в исходном объекте новое поле, в котором будет храниться ссылка на целевой объект. Такая модификация может стать постоянной или сохраниться до тех пор, пока рефакторинг не позволит удалить этот объект.

- Удалите поле из исходного класса.

- Замените все ссылки на исходное поле обращениями к соответствующему методу в целевом классе.

Чтение переменной замените обращением к методу получения значения в целевом объекте; для присваивания переменной замените ссылку обращением к методу установки значения в целевом объекте.

Если поле не является закрытым, поищите ссылки на него во всех подклассах исходного класса.

- Выполните компиляцию и тестирование.

2.2.3. Выделение класса (Extract Class)

Некоторый класс выполняет работу, которую следует поделить между двумя классами.

Создайте новый класс и переместите соответствующие поля и методы, из старого класса в новый.

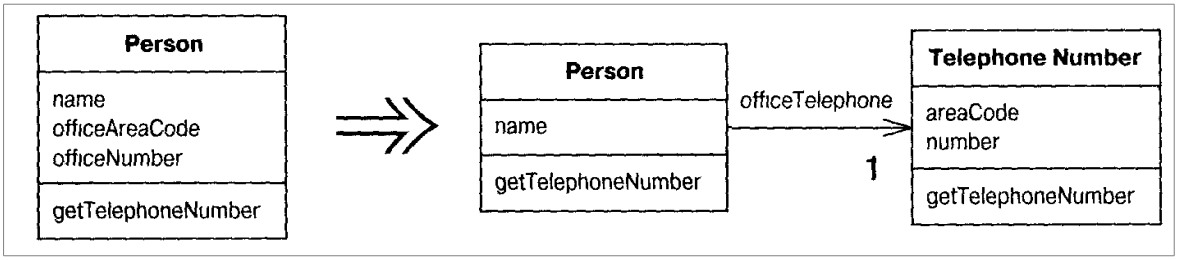

Рисунок 2.3 – Выделение класса

Мотивация

Известно, что класс должен представлять собой ясно очерченную абстракцию, выполнять несколько отчетливых обязанностей. На практике классы подвержены разрастанию. По мере того, как функции растут и плодятся, класс становится слишком сложным.

Получается класс с множеством методов и кучей данных, который слишком велик для понимания. Нужно рассмотреть возможность разделить его на части. Хорошим признаком является сочетание подмножества данных с подмножеством методов. Другой хороший признак – наличие подмножеств данных, которые обычно совместно изменяются или находятся в особой зависимости друг от друга. Полезно задать себе вопрос о том, что произойдет,

если удалить часть данных или метод. Какие другие данные или методы станут бессмысленны?

Одним из признаков, часто проявляющихся в дальнейшем во время разработки, служит характер создания подтипов класса. Может оказаться, что выделение подтипов оказывает воздействие лишь на некоторые функции или что для некоторых функций выделение подтипов производится иначе, чем для других.

Техника

- Определите, как будут разделены обязанности класса.
- Создайте новый класс, выражающий отделяемые обязанности.

Если обязанности прежнего класса перестают соответствовать его названию, переименуйте его.

- Организуйте ссылку из старого класса в новый.

Может потребоваться двусторонняя ссылка, но не создавайте обратную ссылку, пока это не станет необходимо.

- Примените «Перемещение поля» *(Move Field)* ко всем полям, которые желательно переместить.
- После каждого перемещения выполните компиляцию и тестирование.
- Примените «Перемещение метода» *(Move Method)* ко всем методам, перемещаемым из старого класса в новый. Начните с методов более низкого уровня (вызываемых, а не вызывающих) и наращивайте их до более высокого уровня.
- После каждого перемещения выполняйте компиляцию и тестирование.
- Пересмотрите интерфейсы каждого класса и сократите их.

Создав двустороннюю ссылку, посмотрите, нельзя ли превратить ее в одностороннюю.

- Определите, должен ли новый класс быть выставлен наружу. Если да, то решите, как это должно быть сделано – в виде объекта ссылки или объекта с неизменяемым значением.

«Выделение класса» часто используется для повышения живучести параллельной программы, поскольку позволяет устанавливать раздельные блокировки для двух получаемых классов. Если не требуется блокировать оба объекта, то и необязательно делать это.

2.2.4. Встраивание класса (Inline Class)

Класс выполняет слишком мало функций.
Переместите все функции в другой класс и удалите исходный.

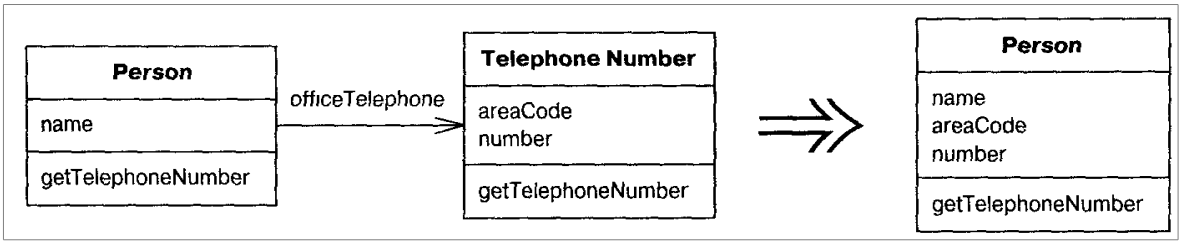

Рисунок 2.4 – Встраивание класса

Мотивация

«Встраивание класса» *(Inline Class)* противоположно «Выделению класса» *(Extract Class)*. Следует обратиться к этой операции, если от класса становится мало пользы и его надо убрать. Часто это происходит в результате рефакторинга, оставившего в классе мало функций.

В этом случае следует вставить данный класс в другой, выбрав для этого такой класс, который чаще всего его использует.

Техника

- Объявите открытый протокол исходного класса в классе, который его поглотит. Делегируйте все эти методы исходному классу.

Если для методов исходного класса имеет смысл отдельный интерфейс, выполните перед встраиванием «Выделение интерфейса» (Extract Interface)

- Перенесите все ссылки из исходного класса в поглощающий класс.

Объявите исходный класс закрытым, чтобы удалить ссылки из за пределов пакета. Поменяйте также имя исходного класса, чтобы компилятор перехватил повисшие ссылки на исходный класс.

- Выполните компиляцию и тестирование.

- С помощью «Перемещения метода» *(Move Method)* и «Перемещения поля» *(Move Field)* перемещайте функции одну за другой из исходного класса, пока в нем ничего не останется.

2.2.5. Сокрытие делегирования (Hide Delegate)

Клиент обращается к делегируемому классу объекта.
Создайте на сервере методы, скрывающие делегирование.

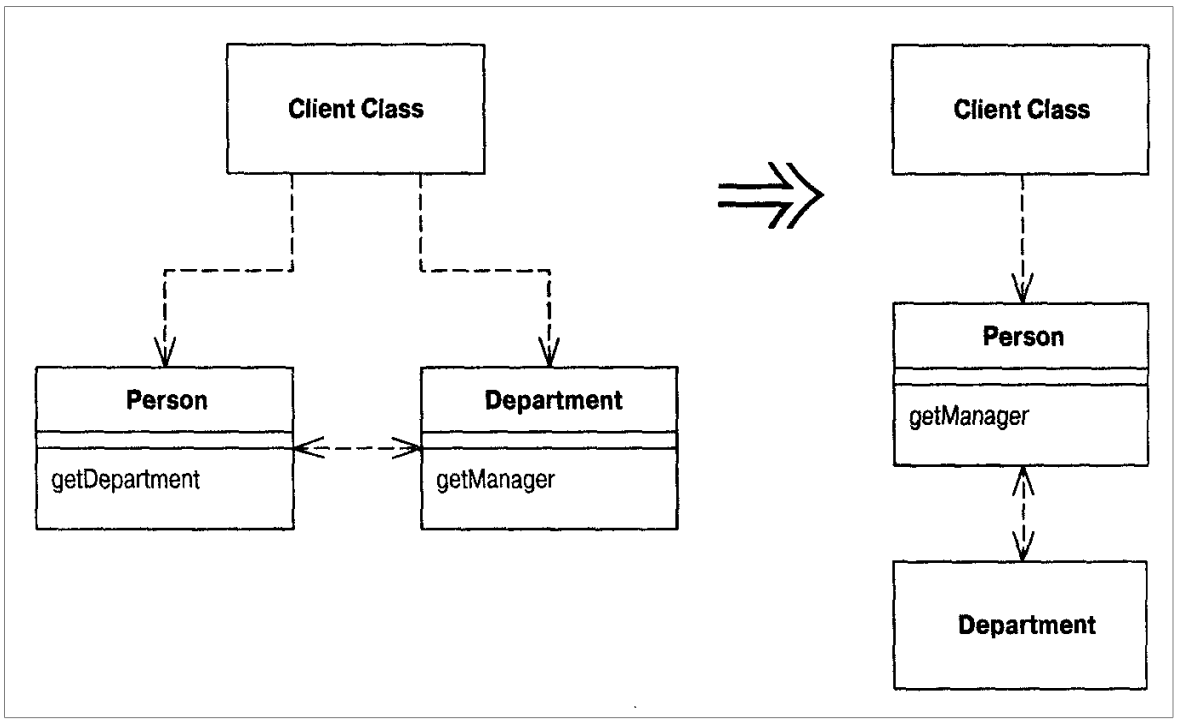

Рисунок 2.5 – Сокрытие делегирования

Мотивация

Одним из ключевых свойств объектов является инкапсуляция. Инкапсуляция означает, что объектам приходится меньше знать о других частях системы. В результате при модификации других частей об этом требуется сообщить меньшему числу объектов, что упрощает внесение изменений.

Всякий, кто занимался объектами, знает, что поля следует скрывать, несмотря на то, что Java позволяет делать поля открытыми. По мере роста искушенности в объектах появляется понимание того, что инкапсулировать можно более широкий круг вещей.

Если клиент вызывает метод, определенный над одним из полей объекта-сервера, ему должен быть известен соответствующий делегированный объект. Если изменяется делегированный объект, может потребоваться модификация клиента. От этой зависимости можно избавиться, поместив в сервер простой делегирующий метод, который скрывает делегирование (рисунок 2.6).

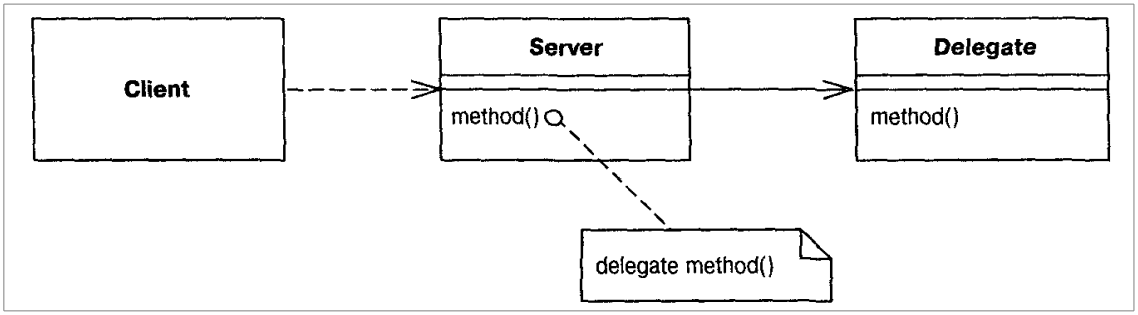

Рисунок 2.6 – Простое делегирование

Тогда изменения ограничиваются сервером и не распространяются на клиента.

Может оказаться полезным применить «Выделение класса» *(Extract Class)* к некоторым или всем клиентам сервера. Если скрыть делегирование от всех клиентов, можно убрать всякое упоминание о нем из интерфейса сервера.

Техника

- Для каждого метода класса-делегата создайте простой делегирующий метод сервера.
- Модифицируйте клиента так, чтобы он обращался к серверу.

Если клиент и сервер находятся в разных пакетах, рассмотрите возможность ограничения доступа к методу делегата областью видимости пакета.

- После настройки каждого метода выполните компиляцию и тестирование.
- Если доступ к делегату больше не нужен никаким клиентам, уберите из сервера метод доступа к делегату.
- Выполните компиляцию и тестирование.

2.2.6. Удаление посредника (Remove Middle Man)

Класс слишком занят простым делегированием.
Заставьте клиента обращаться к делегату непосредственно.

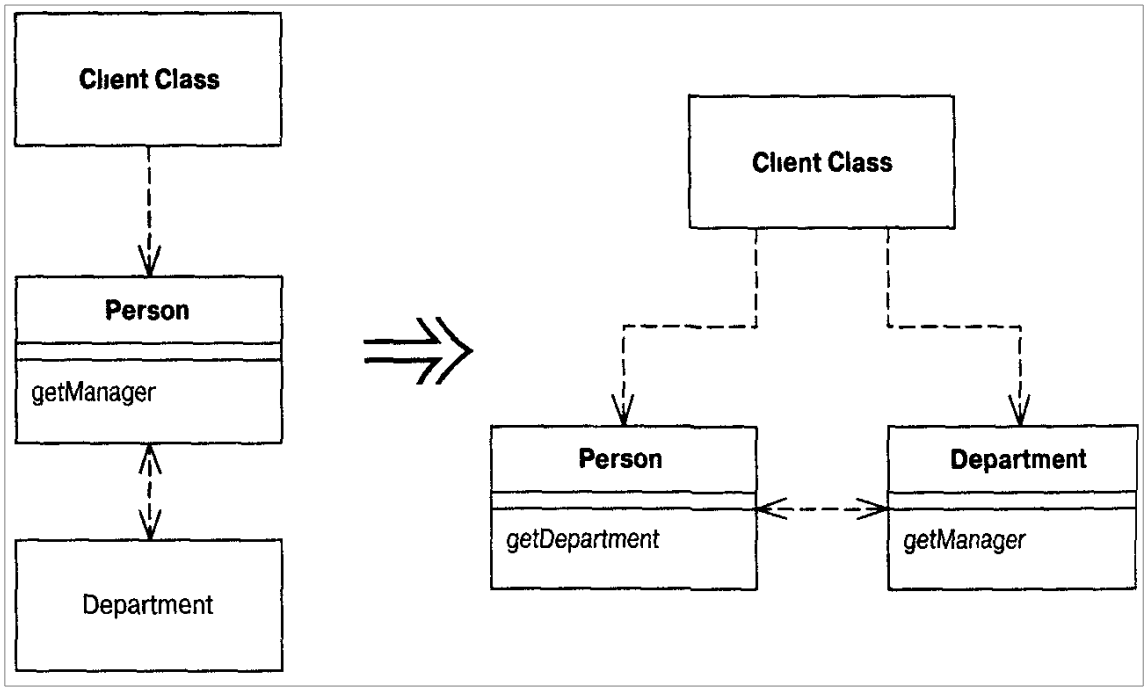

Рисунок 2.7 – Удаление посредника

Мотивация

В мотивировке «Сокрытия делегирования» *(Hide Delegate)* были отмечены преимущества инкапсуляции применения делегируемого объекта. Однако есть и неудобства, связанные с тем, что при желании клиента использовать новую функцию делегата необходимо добавить в сервер простой делегирующий метод. Добавление достаточно большого количества функций оказывается утомительным. Класс сервера становится просто посредником, и может настать момент, когда клиенту лучше непосредственно обращаться к делегату.

Сказать точно, какой должна быть мера сокрытия делегирования, трудно. К счастью, это не столь важно благодаря наличию «Сокрытия делегирования» *(Hide Delegate)* и «Удаления посредника» *(Remove Middle Man).* Можно осуществлять настройку системы по мере надобности. По мере развития системы меняется и отношение к тому, что должно быть скрыто. Инкапсуляция, удовлетворявшая полгода назад, может оказаться неудобной в настоящий момент. Рефакторинг позволяет в этом случае просто внести необходимые исправления.

Техника

- Создайте метод доступа к делегату.
- Для каждого случая использования клиентом метода делегата удалите этот метод с сервера и замените его вызов в клиенте вызовом метода делегата.
- После обработки каждого метода выполняйте компиляцию и тестирование.

2.2.7. Введение внешнего метода (Introduce Foreign Method)

Необходимо ввести в сервер дополнительный метод, но отсутствует возможность модификации класса.

Создайте в классе клиента метод, которому в качестве первого аргумента передается класс сервера.

Мотивация

Ситуация достаточно распространенная. Есть прекрасный класс с отличными сервисами. Затем оказывается, что нужен еще один сервис, но класс его не предоставляет. Если есть возможность модифицировать исходный код, вводится новый метод. Если такой возможности нет, приходится обходными путями программировать отсутствующий метод в клиенте.

Если клиентский класс использует этот метод единственный раз, то дополнительное кодирование не представляет больших проблем и, возможно, даже не требовалось для исходного класса. Однако если метод используется многократно, приходится повторять кодирование снова и снова. Повторение кода – корень всех зол в программах, поэтому повторяющийся код необходимо

выделить в отдельный метод. При проведении этого рефакторинга можно явно известить о том, что этот метод в действительности должен находиться в исходном классе, сделав его внешним методом.

Если обнаруживается, что для класса сервера создается много внешних методов или что многим классам требуется один и тот же внешний метод, то следует применить другой рефакторинг – «Введение локального расширения» *(Introduce Local Extension)*.

Следует помнить, что внешние методы являются искусственным приемом. По возможности следует перемещать методы туда, где им надлежит находиться.

Техника

- Создайте в классе клиента метод, выполняющий нужные вам действия.

Создаваемый метод не должен обращаться к каким-либо характеристикам клиентского класса. Если ему требуется какое-то значение, передайте его в качестве параметра.

- Сделайте первым параметром метода экземпляр класса сервера.
- В комментарии к методу отметьте, что это внешний метод, который должен располагаться на сервере.

Благодаря этому вы сможете позднее, если появится возможность переместить метод, найти внешние методы с помощью текстового поиска.

2.2.8. Введение локального расширения (Introduce Local Extension)

Используемый класс сервера требуется дополнить несколькими методами, но класс недоступен для модификации.

Создайте новый класс с необходимыми дополнительными методами. Сделайте его подклассом или оболочкой для исходного класса.

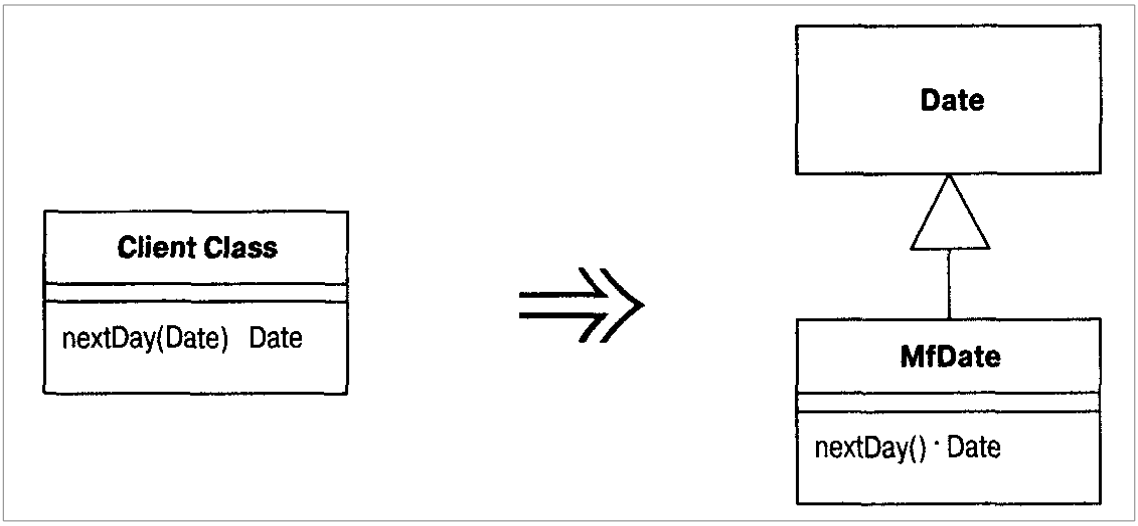

Рисунок 2.8 – Введение локального расширения

Мотивация

К сожалению, создатели классов не могут предоставить все необходимые методы. Если есть возможность модифицировать исходный код, то часто лучше всего добавить в него новые методы. Однако иногда исходный код нельзя модифицировать. Если нужны лишь один-два новых метода, можно применить «Введение внешнего метода» *(Introduce Foreign Method)*. Однако если их больше, они выходят из-под контроля, поэтому необходимо объединить их, выбрав для этого подходящее место. Очевидным способом является стандартная объектно-ориентированная технология создания подклассов и оболочек. В таких ситуациях подкласс или оболочку называют локальным расширением.

Локальное расширение представляет собой отдельный класс, но выделенный в подтип класса, расширением которого он является. Это означает, что он умеет делать то же самое, что и исходный класс, но при этом имеет дополнительные функции. Вместо работы с исходным классом следует создать экземпляр локального расширения и пользоваться им.

При использовании локального расширения поддерживается принцип упаковки методов и данных в виде правильно сформированных блоков. Если же продолжить размещение в других классах кода, который должен располагаться в расширении, это приведет к усложнению других классов и затруднению повторного использования этих методов.

Если стоит вопрос выбора между подклассом и оболочкой, следует учитывать, что использование подкласса связано с меньшим объемом работы. Самое большое препятствие на пути использования подклассов заключается в том, что они должны применяться на этапе создания объектов. Если есть возможность управлять процессом создания, проблем не возникает. Они появляются, если локальное расширение необходимо применять позднее. При работе с подклассами приходится создавать новый объект данного подкласса. Если есть другие объекты, ссылающиеся на старый объект, то появляются два объекта, содержащие данные оригинала. Если оригинал неизменяемый, то проблем не возникает, т. к. можно благополучно воспользоваться копией. Однако если оригинал может изменяться, то возникает проблема, поскольку изменения одного объекта не отражаются в другом. В этом случае надо применить оболочку, тогда изменения, осуществляемые через локальное расширение, воздействуют на исходный объект, и наоборот.

Техника

- Создайте класс расширения в виде подкласса или оболочки оригинала.
- Добавьте к расширению конвертирующие конструкторы.

Конструктор принимает в качестве аргумента оригинал. В варианте с подклассом вызывается соответствующий конструктор родительского класса; в варианте с оболочкой аргумент присваивается полю для делегирования.

- Поместите в расширение новые функции.
- В нужных местах замените оригинал расширением.
- Если есть внешние методы, определенные для этого класса, переместите их в расширение.

3. Порядок выполнения работы

3.1. Выбрать фрагмент программного кода для рефакторинга.
3.2. Выполнить рефакторинг программного кода, применив не менее 7 приемов, рассмотренных в разделе 2.2.
3.3. Составить отчет, содержащий подробное описание каждого модифицированного фрагмента программы и описание использованного метода рефакторинга.

4. Содержание отчета

4.1. Цель работы.
4.2. Постановка задачи.
4.3. Анализ первоначального варианта программного кода.
4.4. Результаты рефакторинга.
4.5. Выводы по работе.

5. Контрольные вопросы

5.1. Какие задачи решает перемещение функций между объектами?
5.2. Какие приемы относятся к перемещению функций между объектами?