Лабораторная работа № 1

Рефакторинг программного кода. Составление методов

1. Цель работы

Исследовать эффективность составления методов при рефакторинга программного кода. Получить практические навыки применения приемов рефакторинга методов.

2. Общие положения

2.1. Основные идеи рефакторинга программного кода

Рефакторинг представляет собой процесс такого изменения программной системы, при котором не меняется внешнее поведение кода, но улучшается его внутренняя структура. Это способ систематического приведения кода в порядок, при котором шансы появления новых ошибок минимальны. В сущности, при проведении рефакторинга кода улучшается его дизайн уже после того, как он написан.

Рефакторинг – это изменение во внутренней структуре программного обеспечения, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения.

Основными результатами проведения рефакторинга являются:

- улучшение композиции программного обеспечения;
- облегчение понимания программного кода;
- упрощение поиска ошибок;
- ускорение разработки программ.

2.2. Составление методов

Важную роль при проведении рефакторинга играет составление методов, которые правильно структурируют программный код. Почти всегда проблемы возникают из-за слишком длинных методов, часто содержащих массу информации, погребенной под сложной логикой, которую они обычно в себе заключают. Основным типом рефакторинга здесь служит «Выделение метода» *(Extract Method)*, в результате которого фрагмент кода превращается в отдельный метод. «Встраивание метода» *(Inline Method)*, по существу, является противоположной процедурой: вызов метода заменяется при этом кодом, содержащимся в теле. «Встраивание метода» может использоваться после проведения нескольких выделений, когда видно, что какие-то из полученных методов больше не выполняют свою долю работы или требуется реорганизовать способ разделения кода на методы.

Самая большая проблема «Выделения метода» связана с обработкой локальных переменных и, прежде всего, с наличием временных переменных. Работая над методом, с помощью «Замены временной переменной вызовом

метода» *(Replace Temp with Query)* следует избавиться от возможно большего количества временных переменных. Если временная переменная используется для разных целей, сначала следует применить «Расщепление временной переменной» *(Split Temporary Variable),* чтобы облегчить последующую ее замену.

Однако иногда временные переменные оказываются слишком сложными для замены. Тогда требуется «Замена метода объектом метода»

(Replace Method with Method Object). Это позволяет разложить даже самый запутанный метод, но ценой введения нового класса для выполнения задачи.

С параметрами меньше проблем, чем с временными переменными, при условии, что им не присваиваются значения. В противном случае требуется выполнить «Удаление присваиваний параметрам» *(Remove Assignments to Parameters).*

Когда метод разложен, значительно легче понять, как он работает. Иногда обнаруживается возможность улучшения алгоритма, позволяющая сделать его более понятным. Тогда применяется «Замещение алгоритма» *(Substitute Algorithm).*

2.2.1. Выделение метода (Extract Method)

Есть фрагмент кода, который можно сгруппировать. Преобразуйте фрагмент кода в метод, название которого объясняет его назначение.

Мотивация

«Выделение метода» – один из наиболее часто используемых приемов рефакторинга. Если есть метод, кажущийся слишком длинным, или код, требующий комментариев, объясняющих его назначение, то этот фрагмент кода следует преобразовать в отдельный метод.

Использование коротких методов с осмысленными именами является предпочтительным по ряду причин. Во-первых, если выделен мелкий метод, повышается вероятность его использования другими методами. Во-вторых, методы более высокого уровня начинают выглядеть как ряд комментариев. Замена методов тоже упрощается, когда они мелко структурированы.

Маленькие методы действительно полезны, если выбирать для них хорошие имена. Если выделение метода делает код более понятным, следует выполнить его, даже если имя метода окажется длиннее, чем выделенный код.

Техника

- Создайте новый метод и назовите его соответственно назначению метода (тому, что, а не как он делает). Когда предполагаемый к выделению код очень простой, например, если он выводит отдельное сообщение или вызывает одну функцию, следует выделять его, если имя нового метода лучше раскрывает

назначение кода. Если вы не можете придумать более содержательное имя, не выделяйте код.

- Скопируйте код, подлежащий выделению, из исходного метода в создаваемый.

- Найдите в извлеченном коде все обращения к переменным, имеющим локальную область видимости для исходного метода. Ими являются локальные переменные и параметры метода.

- Найдите временные переменные, которые используются только внутри этого выделенного кода. Если такие переменные есть, объявите их как временные переменные в создаваемом методе.

- Посмотрите, модифицирует ли выделенный код какие-либо из этих переменных с локальной областью видимости. Если модифицируется одна переменная, попробуйте превратить выделенный код в вызов другого метода, результат которого присваивается этой переменной. Если это затруднительно или таких переменных несколько, в существующем виде выделить метод нельзя. Попробуйте сначала выполнить «Расщепление временной переменной» *(Split Temporary Variable)*, а затем снова выделить метод. Временные переменные можно ликвидировать с помощью «Замены временных переменных вызовом методов» *(Replace Temp with Query)*.

- Передайте в создаваемый метод в качестве параметров переменные с локальной областью видимости, чтение которых осуществляется в выделенном коде.

- Справившись со всеми локальными переменными, выполните компиляцию.

- Замените в исходном методе выделенный код вызовом созданного метода. Если какие-либо временные переменные перемещены в созданный метод, найдите их объявления вне выделенного кода. Если таковые имеются, можно их удалить.

- Выполните компиляцию и тестирование.

Использование локальных переменных

Локальные переменные – параметры, передаваемые в исходный метод, и временных переменных, объявленных в исходном методе, создают дополнительные трудности при выделении метода. Локальные переменные действуют только в исходном методе, поэтому при «Выделении метода» с ними связана дополнительная работа. В некоторых случаях они даже вообще не позволят выполнить рефакторинг.

Проще всего, когда локальные переменные читаются, но не изменяются. В этом случае можно просто передавать их в качестве параметров. Такой способ может быть использован с любым числом локальных переменных.

То же применимо, если локальная переменная является объектом и вызывается метод, модифицирующий переменную. В этом случае также можно передать объект в качестве параметра. Другие меры потребуются, только если действительно выполняется присваивание локальной переменной.

Присваивание нового значения локальной переменной

Сложности возникают, когда локальным переменным присваиваются новые значения. В данном случае мы говорим только о временных переменных. Увидев присваивание параметру, нужно сразу применить «Удаление присваиваний параметрам» *(Remove Assignments to Parametrs)*.

Есть два случая присваивания временным переменным. В простейшем случае временная переменная используется лишь внутри выделенного кода. Если это так, временную переменную можно переместить в выделенный код. Другой случай – использование переменной вне этого кода. В этом случае необходимо обеспечить возврат выделенным кодом модифицированного значения переменной. Можно проиллюстрировать это на следующем методе:

```
void printOwing()
{
    Enumeration e = _orders elements();
    double outstanding = 0.0;

    printBanner();

    // расчет задолженности
    while (e hasMoreElements())
    {
    Order each = (Order) e nextElement(),
    outstanding += each getAmountO,
    }
    printDetails(outstanding),
}
```

Выделим в отдельный метод расчет:

```
void printOwing()
{
    printBanner()
    double outstanding = getOutstanding(),
    printDetails(outstanding),
}

double getOutstanding()
{
    Enumeration e = _orders elements(),
    double outstanding =0.0,
    while (e hasMoreElements())
    {
    Order each = (Order) e nextElement() outstanding += each
getAmount(),
    }
    return outstanding,
}
```

Переменная перечисления используется только в выделяемом коде, поэтому можно целиком перенести ее в новый метод. Переменная `outstanding` используется в обоих местах, поэтому выделенный метод должен ее возвратить. После компиляции и тестирования, выполненных вслед

за выделением, возвращаемое значение переименовывается в соответствии с обычными соглашениями:

```
double getOutstanding()
{
    Enumeration e = _orders elements(),
    double result =0.0
    while (e hasMoreElements())
    {
    Order each = (Order) e nextElement(),
    result += each getAmount(),
    }
    return result
}
```

В данном случае переменная outstanding инициализируется только очевидным начальным значением, поэтому достаточно инициализировать ее в выделяемом методе. Если с переменной выполняются какие-либо сложные действия, нужно передать ее последнее значение в качестве параметра. Исходный код для этого случая может выглядеть так:

```
void printOwing(double previousAmount)
{
    Enumeration e = _orders elements()
    double outstanding = previousAmount * 1.2,
    printBanner(),

    // расчет задолженности
    while (e hasMoreElements())
    {
    Order each = (Order) e nextElement(), outstanding +=
each getAmount(),
    }
    printDetails( outstanding),
}
```

В данном случае выделение будет выглядеть так:

```
void printOwing(double previousAmount)
{
    double outstanding = previousAnount * 1.2,
    printBanner()
    outstanding = getOutstanding(outstandmg)
    printDetails(outstanding)
}

double getOutstanding(double initialValue)
{
    double result = initialValue
    Enumeration e = _orders elements()
    while (e hasMoreElements())
```

```
1
0     {
    Order each = (Order) e nextElement()
    result += each getAmount()
    }
    return result
}
```

После выполнения компиляции и тестирования сделаем более понятным способ инициализации переменной `outstanding`:

```
void printOwing(double previousAmount)
{
    printBanner()
    double outstanding=getOutstanding(previousAmount*1.2)
printDetails(outstanding)
}
```

Может возникнуть вопрос, что произойдет, если надо возвратить не одну, а несколько переменных.

Здесь есть ряд вариантов. Обычно лучше всего выбрать для выделения другой код. Предпочтительно, чтобы метод возвращал одно значение, поэтому следует попытаться организовать возврат разных значений разными методами.

Часто временных переменных так много, что выделять методы становится очень трудно. В таких случаях можно сократить число временных переменных с помощью «Замены временной переменной вызовом метода» *(Replace Temp with Query)*. Если и это не помогает, можно прибегнуть к «Замене метода объектом методов» *(Replace Method with Method Object)*. Для последнего рефакторинга безразлично, сколько имеется временных переменных и какие действия с ними выполняются.

2.2.2. Встраивание метода (Inline Method)

Если тело метода столь же понятно, как и его название, можно поместить тело метода в код, который его вызывает, и удалить метод.

Мотивация

Рефакторинг предполагает, что следует использовать короткие методы с названиями, отражающими их назначение, что приводит к получению более понятного и легкого для чтения кода. Но иногда встречаются методы, тела которых столь же прозрачны, как названия, либо изначально, либо становятся таковыми в результате рефакторинга. В этом случае необходимо избавиться от метода. Косвенность может быть полезной, но излишняя косвенность раздражает.

Еще один случай для применения «Встраивания метода» возникает, если есть группа методов, структура которых представляется неудачной. Можно

встроить их все в один большой метод, а затем выделить методы иным способом.

«Встраиванию метода» целесообразно использовать, когда в коде слишком много косвенности и оказывается, что каждый метод просто выполняет делегирование другому методу, и во всем этом делегировании можно просто заблудиться. В таких случаях косвенность в какой-то мере оправданна, но не целиком. Путем встраивания удается собрать полезное и удалить все остальное.

Техника

- Убедитесь, что метод не является полиморфным. Следует избегать встраивания, если есть подклассы, перегружающие метод; они не смогут перегрузить отсутствующий метод.
- Найдите все вызовы метода.
- Замените каждый вызов телом метода.
- Выполните компиляцию и тестирование.
- Удалите объявление метода.

2.2.3. Встраивание временной переменной (Inline Temp)

Имеется временная переменная, которой один раз присваивается простое выражение, и эта переменная мешает проведению других рефакторингов. Следует заменить этим выражением все ссылки на данную переменную.

Мотивация

Чаще всего встраивание переменной производится в ходе «Замены временной переменной вызовом метода» *(Replace Temp with Query)*, поэтому подлинную мотивировку следует искать там. Встраивание временной переменной выполняется самостоятельно только тогда, когда обнаруживается временная переменная, которой присваивается значение, возвращаемое вызовом метода. Часто эта переменная безвредна, и можно оставить ее в покое. Но если она мешает другим рефакторингам, например, «Выделению метода», ее надо встроить.

Техника

- Объявите временную переменную с ключевым словом final, если это еще не сделано, и скомпилируйте код. Так вы убедитесь, что значение этой переменной присваивается действительно один раз.
- Найдите все ссылки на временную переменную и замените их правой частью присваивания.
- Выполняйте компиляцию и тестирование после каждой модификации.
- Удалите объявление и присваивание для данной переменной.
- Выполните компиляцию и тестирование.

1
2

2.2.4. Замена временной переменной вызовом метода (Replace Temp with Query)

Временная переменная используется для хранения значения выражения. В этом случае следует преобразовать выражение в метод, заменить все ссылки на временную переменную вызовом метода. Новый метод может быть использован в других методах.

Мотивация

Проблема с этими переменными в том, что они временные и локальные. Поскольку они видны лишь в контексте метода, в котором используются, временные переменные ведут к увеличению размеров методов, потому что только так можно до них добраться. После замены временной переменной методом запроса получить содержащиеся в ней данные может любой метод класса. Это существенно содействует получению качественного кода для класса.

«Замена временной переменной вызовом метода» часто представляет собой необходимый шаг перед «Выделением метода». Локальные переменные затрудняют выделение, поэтому следует заменить как можно больше переменных вызовами методов.

Простыми случаями данного рефакторинга являются такие, в которых присваивание временным переменным осуществляется однократно, и те, в которых выражение, участвующее в присваивании, свободно от побочных эффектов. Остальные ситуации сложнее, но разрешимы. Облегчить положение могут предварительное «Расщепление временной переменной» *(Split Temporary Variable)* и «Разделение запроса и модификатора» *(Separate Query from Modifier)*. Если временная переменная служит для накопления результата (например, при суммировании в цикле), соответствующая логика должна быть воспроизведена в методе запроса.

Техника

- Найти простую переменную, для которой присваивание выполняется один раз. Если установка временной переменной производится несколько раз, попробуйте воспользоваться «Расщеплением временной переменной» *(Split Temporary Variable)*.
- Объявите временную переменную с ключевым словом final.
- Скомпилируйте код. Это гарантирует, что присваивание временной переменной выполняется только один раз.
- Выделите правую часть присваивания в метод. Сначала пометьте метод как закрытый (private). Позднее для него может быть найдено дополнительное применение, и тогда защиту будет легко ослабить. Выделенный метод должен быть свободен от побочных эффектов, т. е. не должен модифицировать какие-

либо объекты. Если это не так, воспользуйтесь «Разделением запроса и модификатора» *(Separate Query from Modifier)*.

- Выполните компиляцию и тестирование.
- Выполните для этой переменной «Замену временной переменной вызовом метода» *(Replace Temp with Query)*.

Временные переменные часто используются для суммирования данных в циклах. Цикл может быть полностью выделен в метод, что позволит избавиться от нескольких строк отвлекающего внимание кода. Иногда в цикле складываются несколько величин. В таком случае повторите цикл отдельно для каждой временной переменной, чтобы иметь возможность заменить ее вызовом метода. Цикл должен быть очень простым, поэтому дублирование кода не опасно.

В данном случае могут возникнуть опасения по поводу снижения производительности. Оставим их пока в стороне, как и другие связанные с ней проблемы. В девяти случаях из десяти они не существенны. Если производительность важна, то она может быть улучшена на этапе оптимизации. Когда код имеет четкую структуру, часто находятся более мощные оптимизирующие решения, которые без рефакторинга остались бы незамеченными. Если дела пойдут совсем плохо, можно легко вернуться к временным переменным.

2.2.5. Введение поясняющей переменной (Introduce Explaining Variable)

Поместите результат выражения или его части во временную переменную, имя которой поясняет его назначение.

Мотивация

Выражения могут становиться очень сложными и трудными для чтения. В таких ситуациях полезно с помощью временных переменных превратить выражение в нечто, лучше поддающееся управлению. Особую ценность «Введение поясняющей переменной» имеет в условной логике, когда удобно для каждого пункта условия объяснить, что он означает, с помощью временной переменной с хорошо подобранным именем. Другим примером служит длинный алгоритм, в котором каждый шаг можно раскрыть с помощью временной переменной.

«Введение поясняющей переменной» – очень распространенный вид рефакторинга, но в ряде случаев вместо него целесообразно применять «Выделение метода». Временная переменная полезна только в контексте одного метода. Метод же можно использовать всюду в объекте и в других объектах. Однако бывают ситуации, когда локальные переменные затрудняют применение «Выделения метода. В таких случаях следует обратиться к «Введению поясняющей переменной».

Техника

- Объявите локальную переменную с ключевым словом `final` и установите ее значением результат части сложного выражения.
- Замените часть выражения значением временной переменной. Если эта часть повторяется, каждое повторение можно заменять поочередно
- Выполните компиляцию и тестирование.
- Повторите эти действия для других частей выражения.

2.2.6. Расщепление временной переменной (Split Temporary Variable)

Имеется временная переменная, которой неоднократно присваивается значение, но это не переменная цикла и не временная переменная для накопления результата. В этом случае следует создать для каждого присваивания отдельную временную переменную.

Мотивация

Временные переменные создаются с различными целями. Иногда эти цели естественным образом приводят к тому, что временной переменной несколько раз присваивается значение. Переменные управления циклом изменяются при каждом проходе цикла (например, `i` в `for(int i=0; i<10; i++)`). Накопительные временные переменные аккумулируют некоторое значение, получаемое при выполнении метода.

Другие временные переменные часто используются для хранения результата пространного фрагмента кода, чтобы облегчить последующие ссылки на него. Переменным такого рода значение должно присваиваться только один раз. То, что значение присваивается им неоднократно, свидетельствует о выполнении ими в методе нескольких задач. Все переменные, выполняющие несколько функций, должны быть заменены отдельной переменной для каждой из этих функций. Использование одной и той же переменной для решения разных задач очень затрудняет чтение кода.

Техника

- Измените имя временной переменной в ее объявлении и первом присваивании ей значения. Если последующие присваивания имеют вид `i=i+некоторое_выражение`, то это накопительная временная переменная, и ее расщеплять не надо. С накопительными временными переменными обычно производятся такие действия, как сложение, конкатенация строк, вывод в поток или добавление в коллекцию.
- Объявите новую временную переменную с ключевым словом `final`.

- Измените все ссылки на временную переменную вплоть до второго присваивания.
- Объявите временную переменную в месте второго присваивания.
- Выполните компиляцию и тестирование.
- Повторяйте шаги переименования в месте объявления и изменения ссылок вплоть до очередного присваивания.

2.2.7. Удаление присваиваний параметрам (Remove Assignments to Parameters)

Если код выполняет присваивание параметру, то вместо этого следует воспользоваться присваиванием временной переменной.

Мотивация

Если в качестве значения параметра передается объект с именем `foo`. то присваивание параметру означает, что *`foo`* изменится и станет указывать на другой объект. Выполнение каких-то операций над переданным объектом не является проблемой. Но не следует допускать изменений `foo`, превращающих его в ссылку на совершенно другой объект.

Техника

- Создайте для параметра временную переменную.
- Замените все обращения к параметру, осуществляемые после присваивания, временной переменной.
- Измените присваивание так, чтобы оно производилось для временной переменной.
- Выполните компиляцию и тестирование.

2.2.8. Замена метода объектом методов (Replace Method with Method Object)

Есть длинный метод, в котором локальные переменные используются таким образом, что это не дает применить «Выделение метода». В этом случае можно преобразовать метод в отдельный объект так, чтобы локальные переменные стали полями этого объекта. После этого можно разложить данный метод на несколько методов того же объекта.

Мотивация

Путем выделения в отдельные методы частей большого метода можно сделать код значительно более понятным. Декомпозицию метода затрудняет наличие локальных переменных. Когда они присутствуют в изобилии,

декомпозиция может оказаться сложной задачей. «Замена временной переменной вызовом метода» *(Replace Temp with Query)* может облегчить ее, но иногда необходимое расщепление метода все же оказывается невозможным. В этом случае следует применить «Замену метода объектом методов».

«Замена метода объектом методов» превращает локальные переменные в поля объекта методов. Затем к новому объекту применяется «Выделение метода» *(Extract Method)*, создающее новые методы, на которые распадается первоначальный метод.

Техника

- Создайте новый класс и назовите его так же, как метод.
- Создайте в новом классе поле с модификатором `final` для объекта-владельца исходного метода (исходного объекта) и поля для всех временных переменных и параметров метода.
- Создайте для нового класса конструктор, принимающий исходный объект и все параметры.
- Создайте в новом классе метод с именем «`compute()`» (вычислить).
- Скопируйте в `compute()` тело исходного метода. Для вызовов методов исходного объекта используйте поле исходного объекта.
- Выполните компиляцию.
- Замените старый метод таким, который создает новый объект и вызывает `compute()`.

Теперь, поскольку все локальные переменные стали полями, можно беспрепятственно разложить метод, не нуждаясь при этом в передаче каких-либо параметров.

2.2.9. Замещение алгоритма (Substitute Algorithm)

Желательно заменить алгоритм более понятным. Для этого можно заменить тело метода новым алгоритмом.

Мотивация

Если обнаруживается более понятный способ сделать что-либо, следует заменить сложный способ простым. Рефакторинг позволяет разлагать сложные вещи на более простые части, но иногда наступает такой момент, когда надо взять алгоритм целиком и заменить его чем-либо более простым. Это происходит, когда вы ближе знакомитесь с задачей и обнаруживаете, что можно решить ее более простым способом. Иногда с этим сталкиваешься, начав использовать библиотеку, в которой есть функции, дублирующие имеющийся код.

Иногда, когда желательно изменить алгоритм, чтобы он решал несколько иную задачу, легче сначала заменить его таким кодом, в котором потом проще произвести необходимые изменения.

Перед выполнением этого приема следует убедиться, что дальнейшая декомпозиция метода уже невозможна. Замену большого и сложного алгоритма выполнить очень трудно; только после его упрощения замена становится осуществимой.

Техника

- Подготовьте свой вариант алгоритма. Добейтесь, чтобы он компилировался.
- Прогоните новый алгоритм через свои тесты. Если результаты такие же, как и для старого, на этом следует остановиться.
- Если результаты отличаются, используйте старый алгоритм для сравнения при тестировании и отладке. Выполните каждый контрольный пример со старым и новым алгоритмами и следите за результатами. Это поможет увидеть, с какими контрольными примерами возникают проблемы и какого рода эти проблемы.

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. В чем заключается цель рефакторинга программного кода?
5.2. Какие основные результаты дает рефакторинг?
5.3. Как рефакторинг влияет на производительность программы?
5.4. Какие задачи решает составление методов?
5.5. Какие приемы относятся к составлению методов?