Рефакторинг: методы

Тема 1.1.5. Оптимизация и рефакторинг кода

Техники рефакторинга

- **Техники рефакторинга** описывают конкретные методы борьбы с грязным кодом. Большинство рефакторингов имеют как достоинства, так и недостатки. Поэтому любой рефакторинг должен быть мотивирован и обдуман.

Выделяют группы приемов рефакторинга:

1) *Составление методов*
2) *Перемещение функций между объектами*
3) *Организация данных*
4) *Упрощение условных выражений*
5) *Упрощение вызовов методов*
6) *Решение задач обобщения*

Методы рефакторинга

1. Составление методов

Значительная часть рефакторинга посвящается правильному составлению методов. В большинстве случаев, причиной являются слишком длинные метод что прячет логику выполнения и делают метод крайне сложным для понимания, а значит и изменения. Рефакторинги этой группы призваны уменьшить сложность внутри метода, убрать дублирование кода и облегчить последующую работу с ним.

- *Извлечение метода*
- *Встраивание метода*
- *Извлечение переменной*
- *Встраивание переменной*
- *Замена переменной вызовом метода*
- *Расщепление переменной*
- *Удаление присваиваний параметрам*
- *Замена метода объектом методов*
- *Замена алгоритма*

Методы рефакторинга

2. Перемещение функций между объектами

Если вы разместили функциональность по классам не самым удачным образом — это еще не повод отчаиваться. Рефакторинги этой группы показывают как безопасно перемещать функциональность из одних классов в другие, создавать новые классы, а также скрывать детали реализации из публичного доступа.

- *Перемещение метода*
- *Перемещение поля*
- *Извлечение класса*
- *Встраивание класса*
- *Сокрытие делегирования*
- *Удаление посредника*
- *Введение внешнего метода*
- *Введение локального расширения*

Методы рефакторинга

3. Организация данных

Рефакторинги этой группы призваны облегчить работу с данными, заменив работу с примитивными типами богатыми функциональностью классами. Кроме того, важным моментом является уменьшение связанности между классами, что улучшает переносимость классов и шансы их повторного использования.

- *Замена значения ссылкой*
- *Замена ссылки значением*
- *Дублирование видимых данных*
- *Самоинкапсуляция поля*
- *Замена простого поля объектом*
- *Замена поля-массива объектом*
- *Замена однонаправленной связи двунаправленной*
- *Замена двунаправленной связи однонаправленной*
- *Инкапсуляция поля*
- *Инкапсуляция коллекции*
- *Замена магического числа символьной константой*
- *Замена кодирования типа классом*
- *Замена кодирования типа подклассами*
- *Замена кодирования типа состоянием/стратегией*
- *Замена подкласса полями*

Методы рефакторинга

4. Упрощение условных выражений

Логика условного выполнения имеет тенденцию становиться сложной, поэтому ряд рефакторингов направлен на то, чтобы упростить ее.

- *Объединение условных операторов*
- *Объединение дублирующихся фрагментов в условных операторах*
- *Разбиение условного оператора*
- *Замена условного оператора полиморфизмом*
- *Удаление управляющего флага*
- *Замена вложенных условных операторов граничным оператором*
- *Введение Null-объекта*
- *Введение проверки утверждения*

Методы рефакторинга

5. Упрощение вызовов методов

Эти рефакторинги делают вызовы методов проще и яснее для понимания, что, в свою очередь, упрощает интерфейсы взаимодействия между классами.

- *Добавление параметра*
- *Удаление параметра*
- *Переименование метода*
- *Разделение запроса и модификатора*
- *Параметризация метода*
- *Замена параметров объектом*
- *Передача всего объекта*
- *Удаление сеттера*
- *Замена параметра набором специализированных методов*
- *Замена параметра вызовом метода*
- *Сокрытие метода*
- *Замена конструктора фабричным методом*
- *Замена кода ошибки исключением*
- *Замена исключения проверкой условия*

Методы рефакторинга

6. Решение задач обобщения

Обобщение порождает собственную группу рефакторингов, в основном связанных с перемещением функциональности по иерархии наследования классов, создания новых классов и интерфейсов, а также замены наследования делегированием и наоборот.

- *Подъём поля*
- *Подъём метода*
- *Подъём тела конструктора*
- *Спуск поля*
- *Спуск метода*
- *Извлечение подкласса*
- *Извлечение суперкласса*
- *Извлечение интерфейса*
- *Свёртывание иерархии*
- *Создание шаблонного метода*
- *Замена наследования делегированием*
- *Замена делегирования наследованием*

Крупные рефакторинги

1) **«Разделение наследования»** (Tease Apart Inheritance) имеет дело с запутанной иерархией наследования, в которой различные варианты представляются объединенными вместе так, что это сбивает с толку.
2) **«Преобразование процедурного проекта в объекты»** (Convert Procedu& ral Design to Object) содействует решению классической проблемы преобразования процедурного кода. Множество программистов пользуется объектно-ориентированными языками, не имея действительного представления об объектах, поэтому данный вид рефакторинга приходится часто выполнять.
3) Если вы увидите код, написанный с классическим двухзвенным подходом к интерфейсам пользователя и базам данных, то вам понадобится **«Отделение предметной области от представления»** (Separate Domain from Presentation), чтобы разделить бизнес-логику и код интерфейса пользователя. Опытные объектно-ориентированные разработчики поняли, что такое разделение жизненно важно для долго живущих и благополучных систем.
4) **«Выделение иерархии»** (Extract Hierarchy) упрощает чрезмерно сложные классы путем превращения их в группы подклассов

Составление методов

1. Выделение (извлечение) метода
Проблема: есть фрагмент кода, который можно сгруппировать.
Решение: Выделите участок кода в новый метод (или функцию) и вызовите этот метод вместо старого кода.
2. Встраивание метода
Проблема: Стоит использовать в том случае, когда тело метода очевиднее самого метода.
Решение: Замените вызовы метода его содержимым и удалите сам метод.
3. Введение поясняющей (извлечение) переменной
Проблема: есть сложное для понимания выражение.
Решение: Поместите результат выражения или его части в отдельные переменные, поясняющие суть выражения.
4. Встраивание временной переменной
Проблема: есть временная переменная, которой присваивается результат простого выражения (и больше ничего).
Решение: Замените обращения к переменной этим выражением.
5. Замена временной переменной вызовом метода
Проблема: результат какого-то выражения помещается в локальную переменную, чтобы использовать её далее в коде.
Решение: Выделите все выражение в отдельный метод и возвращайте результат из него. Замените использование вашей переменной вызовом метода. Новый метод может быть использован и в других методах.

6. Расщепление временной переменной
Проблема: есть локальная переменная, которая используется для хранения разных промежуточных значений внутри метода (за исключением переменных циклов).
Решение: Используйте разные переменные для разных значений. Каждая переменная должна отвечать только за одну определённую вещь.
7. Удаление присваиваний параметрам
Проблема: Параметру метода присваивается какое-то значение.
Решение: Вместо параметра воспользуйтесь новой локальной переменной.
8. Замена метода объектом методов
Проблема: есть длинный метод, в котором локальные переменные так сильно переплетены, что это делает невозможным применение «извлечения метода».
Решение: Преобразуйте метод в отдельный класс так, чтобы локальные переменные стали полями этого класса. После этого можно без труда разделить метод на части.
9. Замена алгоритма
Проблема: хотите заменить существующий алгоритм другим?
Решение: Замените тело метода, реализующего старый алгоритм, новым алгоритмом.

Перемещение функций между объектами

1. Перемещение метода
Проблема: Метод используется в другом классе больше, чем в собственном.
Решение: Создайте новый метод в классе, который использует его больше других, и перенесите туда код из старого метода. Код оригинального метода превратите в обращение к новому методу в другом классе либо уберите его вообще.
2. Перемещение поля
Проблема: Поле используется в другом классе больше, чем в собственном.
Решение: Создайте поле в новом классе и перенаправьте к нему всех пользователей старого поля.
3. Выделение (извлечение) класса
Проблема: Один класс работает за двоих.
Решение: Создайте новый класс, переместите в него поля и методы, отвечающие за определённую функциональность.
4. Встраивание класса
Проблема: Класс почти ничего не делает (или выполняет слишком мало функций), ни за что не отвечает, и никакой ответственности для этого класса не планируется.
Решение: Переместите все функции из описанного класса в другой и удалите исходный.

5. Сокрытие делегирования

Проблема: Клиент получает объект B из поля или метода объекта A. Затем клиент вызывает какой-то метод объекта B.

Решение: Создайте новый метод в классе A, который бы делегировал вызов объекту B. Таким образом, клиент перестанет знать о классе B и зависеть от него.

6. Удаление посредника

Проблема: Класс имеет слишком много методов, которые просто делегируют работу другим объектам.

Решение: Удалите эти методы и заставьте клиента вызывать конечные методы напрямую.

7. Введение внешнего метода

Проблема: Служебный класс не содержит метода, который вам нужен, при этом у вас нет возможности добавить метод в этот класс.

Решение: Добавьте метод в клиентский класс и передавайте в него объект служебного класса в качестве аргумента.

8. Введение локального расширения

Проблема: В служебном классе отсутствуют некоторые методы, которые вам нужны. При этом добавить их в этот класс вы не можете.

Решение: Создайте новый класс, который бы содержал эти методы, и сделайте его наследником служебного класса, либо его обёрткой.

Организация данных

1. Самоинкапсуляция поля
Проблема: используется прямой доступ к приватным полями внутри класса.
Решение: Создайте методы получения и установки (геттер и сеттер) для поля, и пользуйтесь для доступа к полю только ими.
2. Замена простого поля (значения данных) объектом
Проблема: В классе (или группе классов) есть поле простого типа. У этого поля есть своё поведение и связанные данные.
Решение: Создайте новый класс, поместите в него старое поле и его поведения, храните объект этого класса в исходном классе.
3. Замена значения ссылкой
Проблема: Есть много одинаковых экземпляров одного класса, которые можно заменить одним объектом.
Решение: Превратите одинаковые объекты в один объект-ссылку.
4. Замена ссылки значением
Проблема: есть объект-ссылка, который слишком маленький и неизменяемый, чтобы оправдать сложности по управлению его жизненным циклом.
Решение: Превратите его в объект-значение.
5. Замена массива объектом
Проблема: есть массив, в котором хранятся разнотипные данные.
Решение: Замените массив объектом, который будет иметь отдельные поля для каждого элемента.

6. Дублирование видимых данных
Проблема: Данные предметной области программы хранятся в классах, отвечающих за пользовательский интерфейс (GUI).
Решение: Имеет смысл выделить данные предметной области в отдельные классы и, таким образом, обеспечить связь и синхронизацию между классом предметной области и GUI.
7. Замена однонаправленной связи двунаправленной
Проблема: есть два класса, которым нужно использовать функции друг друга, но между ними существует только односторонняя связь.
Решение: Добавьте обратные указатели и измените модификаторы, чтобы они обновляли оба набора.
8. Замена двунаправленной связи однонаправленной
Проблема: есть двухсторонняя связь между классами, но один из классов больше не использует функции другого.
Решение: Уберите неиспользуемую связь.
9. Замена магического числа символьной константой
Проблема: В коде используется число, которое несёт какой-то определённый смысл.
Решение: Замените это число константой с человеко-читаемым названием, объясняющим смысл этого числа.
10. Инкапсуляция поля
Проблема: есть публичное поле.
Решение: Сделайте поле приватным и создайте для него методы доступа.

11. Инкапсуляция коллекции
Проблема: Класс содержит поле-коллекцию и простой геттер и сеттер для работы с этой коллекцией.
Решение: Сделайте возвращаемое геттером значение доступным только для чтения и создайте методы добавления/удаления элементов этой коллекции.
12. Замена кода типа классом
Проблема: У класса есть числовой тип, который не влияет на его поведение.
Решение: Замените число новым классом.
13. Замена кода типа подклассами
Проблема: Имеется неизменяемый код типа, воздействующий на поведение класса.
Решение: Замените код типа подклассами, т.е. для каждого значения закодированного типа, создайте подклассы. А затем, вынесите соответствующие поведения из исходного класса в эти подклассы. Управляющий код замените полиморфизмом.
14. Замена кода типа состоянием/стратегией
Проблема: есть закодированный тип, который влияет на поведение, но вы не можете использовать подклассы, чтобы избавиться от него.
Решение: Замените кодирование типа объектом-состоянием. При необходимости заменить значение поля с кодированием типа, в него подставляется другой объект-состояние.
15. Замена подкласса полями
Проблема: есть подклассы, которые отличаются только методами, возвращающими данные-константы.
Решение: Замените методы полями в родительском классе и удалите подклассы.

Упрощение условных выражений

1. Декомпозиция (разбиение) условного оператора
Проблема: есть сложный условный оператор (if-then/else или switch).
Решение: Выделите в отдельные методы все сложные части оператора: условие, then и else.
2. Консолидация (объединение) условного выражения
Проблема: есть несколько условных операторов, ведущих к одинаковому результату или действию.
Решение: Объедините все условия в одном условном операторе.
3. Консолидация (объединение) дублирующихся условных фрагментов
Проблема: Одинаковый фрагмент кода находится во всех ветках условного оператора.
Решение: Вынесите его за рамки оператора.
4. Удаление управляющего флага
Проблема: есть булевская переменная, которая играет роль управляющего флага для нескольких булевских выражений.
Решение: Используйте break, continue и return вместо этой переменной.
5. Замена вложенных условных операторов граничным оператором
Проблема: есть группа вложенных условных операторов, среди которых сложно выделить нормальный ход выполнения кода.
Решение: Выделите все проверки специальных или граничных случаев выполнения в отдельные условия и поместите их перед основными проверками.

6. Замена условного оператора полиморфизмом

Проблема: есть условный оператор, который, в зависимости от типа или свойств объекта, выполняет различные действия.

Решение: Создайте подклассы, которым соответствуют ветки условного оператора. В них создайте общий метод и переместите в него код из соответствующей ветки условного оператора. Впоследствии замените условный оператор на вызов этого метода. Таким образом, нужная реализация будет выбираться через полиморфизм в зависимости от класса объекта.

7. Введение Null-объекта

Проблема: Из-за того, что некоторые методы возвращают null вместо реальных объектов, в коде присутствует множество проверок на null.

Решение: Вместо null возвращайте Null-объект, который предоставляет поведение по умолчанию.

8. Введение проверки утверждения

Проблема: Корректная работа участка кода предполагает наличие каких-то определённых условий или значений.

Решение: Замените эти предположения конкретными проверками.

Упрощение вызовов методов

1. Переименование метода
Проблема: Название метода не раскрывает суть того, что он делает.
Решение: Измените название метода.
2. Добавление параметра
Проблема: Методу не хватает данных для осуществления каких-то действий.
Решение: Создайте новый параметр, чтобы передать эти данные.
3. Удаление параметра
Проблема: Параметр не используется в теле метода.
Решение: Удалите неиспользуемый параметр.
4. Разделение запроса и модификатора
Проблема: есть метод, который возвращает какое-то значение, но при этом в процессе работы он изменяет что-то внутри объекта.
Решение: Разделите метод на два разных метода. Один - возвращает значение, а второй модифицирует объект.
5. Параметризация метода
Проблема: Несколько методов выполняют похожие действия, которые отличаются только какими-то внутренними значениями, числами или операциями.
Решение: Объедините все эти методы в один с параметром, в который будет передаваться отличающееся значение.

6. Замена параметра явными методами
Проблема: Метод разбит на части, каждая из которых выполняется в зависимости от значения какого-то параметра.
Решение: Извлеките отдельные части метода в собственные методы и вызывайте их вместо оригинального метода.
7. Сохранение (передача) всего объекта
Проблема: Вы получаете несколько значений из объекта, а затем передаёте их в метод как параметры.
Решение: Вместо этого передавайте весь объект.
8. Замена параметра вызовом метода
Проблема: Вызываем метод и передаем его результаты как параметры другого метода. При этом значение параметров могли бы быть получены и внутри вызываемого метода.
Решение: Вместо передачи значения через параметры метода, попробуйте переместить код получения значения внутрь самого метода.
9. Введение граничного объекта (замена параметров объектом)
Проблема: встречается повторяющаяся группа параметров, естественным образом связанных друг с другом.
Решение: Замените эти параметры объектом.
10. Инкапсуляция нисходящего преобразования типа
Проблема: Метод возвращает объект, к которому вызывающий должен применить нисходящее преобразование типа.
Решение: Переместите нисходящее преобразование внутрь метода

11. Удаление метода установки значения (сеттера)
Проблема: Значение поля должно быть установлено только в момент создания и больше никогда не меняться.
Решение: Удалите методы, устанавливающие значение этого поля.
12. Сокрытие метода
Проблема: Метод не используется другими классами, либо используется только внутри своей иерархии классов.
Решение: Сделайте метод приватным или защищённым.
13. Замена конструктора фабричным методом
Проблема: есть сложный конструктор, делающий нечто большее, чем простая установка значений полей объекта.
Решение: Создайте фабричный метод и замените им вызовы конструктора.
14. Замена кода ошибки исключением
Проблема: Метод возвращает определенное значение, которое будет сигнализировать об ошибке.
Решение: Вместо этого следует выбрасывать исключение.
15. Замена исключения проверкой условия
Проблема: Вы выбрасываете исключение там, где можно было бы обойтись простой проверкой условия.
Решение: Замените выбрасывание исключения проверкой этого условия.

Решение задач обобщения

1. Подъём поля
Проблема: Два класса имеют одно и то же поле.
Решение: Переместите поле в родительский класс, убрав его из подклассов.
2. Подъём метода
Проблема: Подклассы имеют методы, которые делают схожую работу.
Решение: В этом случае нужно сделать методы идентичными, а затем переместить их в родительский класс.
3. Подъём тела конструктора
Проблема: Подклассы имеют конструкторы с преимущественно одинаковым кодом.
Решение: Создайте конструктор в родительском классе и вынесите в него общий для подклассов код. Вызывайте конструктор родительского класса в конструкторах подкласса.
4. Спуск метода
Проблема: Поведение, реализованное в родительском классе, используется только одним или несколькими подклассами.
Решение: Переместите это поведение в подклассы.
5. Спуск поля
Проблема: Поле используется только в некоторых подклассах.
Решение: Переместите поле в эти подклассы.
6. Выделение (извлечение) подкласса
Проблема: Класс имеет функции, которые используются только в определённых случаях.
Решение: Создайте подкласс и используйте его в этих случаях.

7. Выделение (извлечение) родительского класса
Проблема: есть два класса с общими полями и методами.
Решение: Создайте для них общий родительский класс и перенесите туда одинаковые поля и методы.
8. Выделение (извлечение) интерфейса
Проблема: Несколько клиентов пользуются одной и той же частью интерфейса класса. Либо в двух классах часть интерфейса оказалась общей.
Решение: Выделите эту общую часть в свой собственный интерфейс.
9. Свёртывание иерархии
Проблема: есть некая иерархия классов, в которой подкласс мало чем отличается от родительского класса .
Решение: Слейте подкласс и родительский класс воедино.
10. Формирование шаблона метода
Проблема: В подклассах реализованы алгоритмы, содержащие похожие шаги и одинаковый порядок выполнения этих шагов.
Решение: Вынесите структуру алгоритма и одинаковые шаги в родительский класс, а в подклассах оставьте реализацию отличающихся шагов.
11. Замена наследования делегированием
Проблема: есть подкласс, который использует только часть методов родительского класса или не хочет наследовать его данные.
Решение: Создайте поле и поместите в него объект родительского класса, делегируйте выполнение методов объекту родительского класса, удалите подклассы.
12. Замена делегирования наследованием
Проблема: Класс содержит множество простых делегирующих методов ко всем методам другого класса.
Решение: Сделайте класс наследником делегата, после чего делегирующие методы потеряют смысл.

Контрольные вопросы:

1) Техники рефакторинга: понятие
2) Опишите группы простых методов рефакторинга: назначение
3) По каждой группе простых методов рефакторинга составьте краткое их описание:

Название метода	Причина использования	Решение проблемы
Выделение (извлечение) метода		
…		

4) Опишите крупные рефакторинги.

Основная:

Фаулер М. Рефакторинг: улучшение существующего кода. – Пер. с англ. – СПб: Символ-Плюс, 2018. – 432 с., ил.