1.

**Рефа́кторинг** (англ. refactoring), или перепроектирование кода, переработка кода, равносильное преобразование алгоритмов — процесс изменения внутренней структуры программы, не затрагивающий её внешнего поведения и имеющий целью облегчить понимание её работы.

Цель оптимизации — улучшение производительности программы, а рефакторинга — улучшение понятности кода. После оптимизации исходный код может стать сложнее для понимания.

2.

Цель рефакторинга — сделать код программы более легким для понимания; без этого рефакторинг нельзя считать успешным.

3.

Признаки, показывающие, что назрела необходимость в рефакторинге:

-Программа работает, но даже небольшие доработки сильно затягиваются из-за того, что каждый раз приходится долго разбираться в коде.

-Разработчик постоянно не может точно сказать, сколько времени ему нужно на выполнение задачи, потому что «там надо вначале разбираться».

-Одинаковые изменения приходится вносить в разные места текста программы.

Такой код нужно срочно рефакторить, иначе он будет тормозить реализацию проекта и затруднять внесение правок.

Вообще рефакторинг нужно проводить постоянно. Делайте его каждый раз, после того как поменяли программу и убедились, что всё работает. Например, если добавили или изменили какую-то функцию, метод, класс или объявили новую переменную.

4.

1) **Мёртвый код**. Переменная, параметр, метод или класс больше не используются: требования к программе изменились, но код не почистили. Мёртвый код может встретиться и в сложной условной конструкции, где какая-то ветка никогда не исполняется из-за ошибки или изменения требований. Такие элементы или участки текста нужно удалить.

2) **Дублирование**. Один и тот же код выполняет одно и то же действие в нескольких местах программы. Вынесите эту часть в отдельную функцию.

3) **Имена переменных, функций или классов не передают их назначение.** Имена должны сообщать, почему элемент кода существует, что он делает и как используется. Если видите, что намерения программиста непонятны без комментария, — рефакторьте.

Примеры корректных имен: totalScore — переменная, означающая итоговый счёт в игре, maxWeight — максимальный вес. Для функций и методов лучше использовать глаголы, например: saveScore () — сохранить счет, setSize () — задать размер, getSpeed () — получить скорость.

4) **Слишком длинные функции и методы.** Оптимальный размер этих элементов — 2-3 десятка строк. Если получается больше, разделите функцию на несколько маленьких и добавьте одну общую. Пусть маленькие выполняют по одной операции, а общая функция их вызывает.

5) **Слишком длинные классы.** То же самое. Оптимальная длина класса — 20–30 строк. Разбейте длинный класс на несколько маленьких и включите их объекты в один общий класс.

6) **Слишком длинный список параметров функции или метода**. Они только запутывают, а не помогают. Если все эти параметры действительно нужны, вынесите их в отдельную структуру или класс с понятным именем, а в функцию передайте ссылку на него.

7) **Много комментариев**. Плохой код часто прикрывается обильными комментариями. Если почувствовали желание пояснить какой-то участок кода, попробуйте сначала его переписать, чтобы и так стал понятным. Бесполезные комментарии загромождают программу, а устаревшие и неактуальные вводят в заблуждение.

5.

Также известен как: Extract Method

*Проблема*

У вас есть фрагмент кода, который можно сгруппировать.

*Решение*

Выделите участок кода в новый метод (или функцию) и вызовите этот метод вместо старого кода.

**Порядок рефакторинга**

1) Создайте новый метод и назовите его так, чтобы название отражало суть того, что будет делать этот метод.

2) Скопируйте беспокоящий вас фрагмент кода в новый метод. Удалите этот фрагмент из старого места и замените вызовом вашего нового метода.

3) Найдите все переменные, которые использовались в этом фрагменте кода. Если они были объявлены внутри этого фрагмента и не используются вне его, просто оставьте их без изменений — они станут локальными переменными нового метода.

4) Если переменные объявлены перед интересующим вас участком кода, значит, их следует передать в параметры вашего нового метода, чтобы использовать значения, которые в них находились ранее. Иногда от таких переменных проще избавиться с помощью замены переменных вызовом метода.

5) Если вы видите, что локальная переменная как-то изменяется в вашем участке кода, это может означать, что её изменённое значение понадобится дальше в основном методе. Проверьте это. Если подозрение подтвердилось, значение этой переменной следует возвратить в основной метод, чтобы ничего не сломать.

6.

**Перемещение метода**

Также известен как: Move Method

*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.

**Перемещение поля**

Также известен как: Move Field

*Проблема*

Поле используется в другом классе больше, чем в собственном.

*Решение*

Создайте поле в новом классе и перенаправьте к нему всех пользователей старого поля.

7.

Самоинкапсуляция поля

Замена простого поля объектом

Замена значения ссылкой

Замена ссылки значением

Замена поля-массива объектом

Дублирование видимых данных

Замена однонаправленной связи двунаправленной.

Замена двунаправленной связи однонаправленной

Замена магического числа символьной константой

Инкапсуляция поля

Инкапсуляция коллекции

Замена кодирования типа классом

Замена кодирования типа подклассами

Замена кодирования типа состоянием/стратегией

Замена подкласса полями

8.

Переименование метода

Добавление параметра

Удаление параметра

Разделение запроса и модификатора

Параметризация метода

Замена параметра набором специализированных методов

Передача всего объекта

Замена параметра вызовом метода

Замена параметров объектом

Удаление сеттера

Сокрытие метода

Замена конструктора фабричным методом

Замена кода ошибки исключением

Замена исключения проверкой условия

9.

Разбиение условного оператора

Объединение условных операторов

Объединение дублирующихся фрагментов в условных операторах

Удаление управляющего флага

Замена вложенных условных операторов граничным оператором

Замена условного оператора полиморфизмом

Проблема: У вас есть условный оператор, который, в зависимости от типа или свойств

Введение Null-объекта

Введение проверки утверждения

10.

Подъём поля

Подъём метода

Подъём тела конструктора

Спуск метода

Спуск поля

Извлечение подкласса

Извлечение суперкласса

Извлечение интерфейса

Свёртывание иерархии

Создание шаблонного метода

Замена наследования делегированием

Замена делегирования наследованием