Паттерны и практики написания кода

Преждевременная оптимизация, Принцип наименьшего удивления, Закон Деметры, Магические числа

избегайте преждевременной оптимизации

Преждевременная оптимизация – это корень всех бед. Не оптимизируйте свой код без веской на то причины

Тони Хоар

разница между рефакторингом и оптимизацией

В 97% случаев в оптимизации нет необходимости. До оптимизации стоит задуматься несколько раз о проведении рефакторинга. В наше время время разработчиков стоит дороже чем покупка нового сервера для расширения ресурсов, потому качество кода первично

→ Рефакторинг и оптимизация схожи: оба они не изменяют исходное поведение программы. После их применения процессы с точки зрения продукта и набора действий пользователя останутся нетронутыми

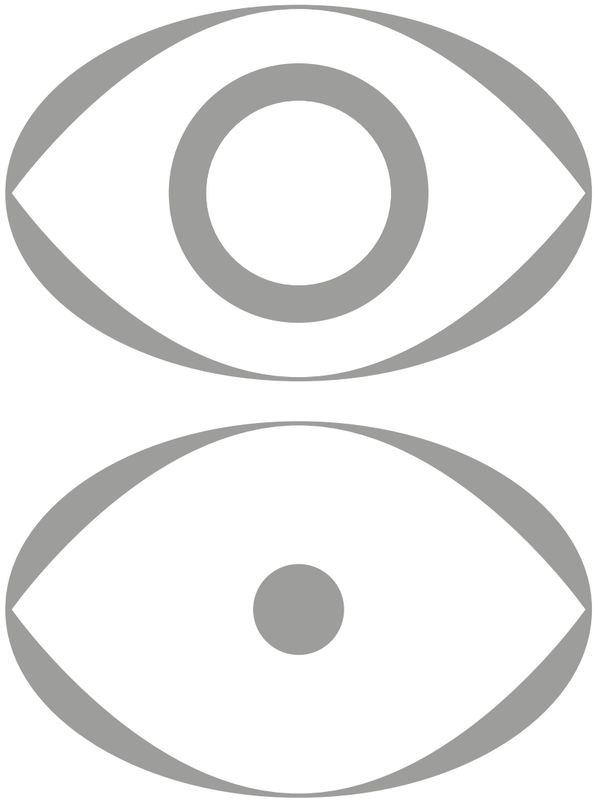
НО

→ Рефакторинг делает код выразительным, читаемым, гибким и легко расширяемым. Он придает коду наглядность, чистоту и ясность – пользуемся рефакторингом постоянно!

→ Оптимизация ускоряет работу программы по памяти или времени, при этом ухудшает понимание кода, его читаемость падает

→ Рефакторинг и оптимизация решают разные проблемы: рефакторинг – читаемость, оптимизация – ресурсы

принцип наименьшего удивления



- Если некоторый элемент, сущность, конструкция и так далее имеет высокую степень удивления то, возможно, необходимо его перепроектировать
- Код должен читаться словно книга. В них чтение происходит сверху вниз и есть перекрёстные ссылки на соседние главы

Пишите так, чтобы никого не удивлять, что весьма очевидно. Константы, методы и свойства должны располагаться на своём уровне абстракции. Избавляйтесь от побочных эффектов. Название методов должны соответствовать тому, что они действительно выполняют

закон Деметры = принцип минимальной информированности

ЦЕЛИ ПРИНЦИПА

- **наложить ограничения на взаимодействие объектов.** Оптимально применять при написании нового кода, а вот переписывание уже существующего может обойтись слишком дорого

- **уменьшить связанность (применить low coupling) за счёт взаимодействия с классами напрямую.** Здесь, в отличие от материалов 1 сезона, закон Деметры имеет четкие правила, рассматриваем его как пример прикладного подхода, который реализует слабую связанность между классами

- **упрощать поддержку и повторное использование классов,** что помогает ещё лучше давать вам оценку качеству кода

ИТОГ

Принцип минимальной информированности формализует ограничения для достижения слабого связывания объектов (low coupling)

правила, которые применяются в Законе Деметры

Из методов объекта должны вызываться те, которые принадлежат:

- самому объекту
- объектам, переданным в параметрах метода
- любому объекту, созданному внутри метода

магические числа

ПРИНЦИП

выносить в константы все числа, которые появляются в программе без объяснений (к ним же относим все строковые значения, например, ‘credit_card’, ‘order_status’)

подробнее можно почитать в книге *Стива Макконелла* **«СОВЕРШЕННЫЙ КОД»**

какие плюсы дает вынос строк и чисел в константы

понимание значения числа за счет именованной константы улучшается. Теперь мы точно знаем, что скрывается за тем или иным числом (например, http код, просто продуктовое значение или внутренний идентификатор ошибки при ответе в межсервисном взаимодействии)

код становится легко читаемым и понятным. Сама константа описывает смысл значения + всегда можно добавить комментарий с развернутым описанием. Важно следить за отсутствием опечаток, т.к. это приводит к необходимости матчинга двух разных вариантов

значения констант легко заменить во всём коде. Так не нужно будет искать и разбираться какое значение относится к одному сценарию использования, а какое не связано с ним

числа 0 и 1 можно не добавлять в константы, если они семантически означают начало отсчёта


avito.tech