# АНТИПАТТЕРНЫ

## BLOATERS (раздувальщики)

### Long method (длинный метод)

Метод содержит слишком большое число строк кода. Длина метода более 10 строк должна начинать вас беспокоить.

Если хочется что-то закомментить, лучше вынести этот кусок в отдельный метод

### Large class (большой класс)

Класс содержит множество полей/методов/строк кода. Часто как следствие того, что класс расфокусирован (low cohesion)

### Primitive obsession (одержимость элементарными типами)

Использование элементарных типов вместо маленьких объектов для небольших задач (например, валюта, диапазоны, специальные строки для телефонных номеров и т.п.)

Использование констант для кодирования какой-то информации (например, константа USER\_ADMIN\_ROLE = 1

Использование строковых констант в качестве названий полей в массивах.

### Long parameter list (длинный список параметров)

Количество параметров метода больше 4

### Data Clumps (группы данных)

Иногда в разных частях кода встречаются одинаковые группы переменных (например, параметры подключения к базе данных).

Решение: группы следует превращать в самостоятельные классы.

## OBJECT-ORIENTATION ABUSERS (нарушители ООП)

### Switch statements (операторы switch)

У вас есть сложный оператор switch или последовательность if-ов.

Решение: каждой ветке можно создать свой метод\класс и вынести туда логику.

### Temporary field (временное поле)

Временные поля – это поля, которые нужны объекту только при определённых обстоятельствах. Только тогда они заполняются какими-то значениями, оставаясь пустыми в остальное время.

Решение — сделать локальные переменные полями нового класса

### Refused bequest (отказ от наследства)

Если подкласc использует лишь малую часть унаследованных методов и свойств суперкласа, это является признаком неправильной иерархии. При этом ненужные методы могут просто не использоваться либо быть переопределёнными и выбрасывать исключения.

Решение: создав второй класс, заменить наследование делегированием, в котором второй класс будет пользоваться методами первого

### Alternative classes with different interfaces (альтернативные классы с разными интерфейсами)

Два класса выполняют одинаковые функции, но имеют разные названия методов — программист мог не заметить, что похожий класс уже существует.

Решение: попробовать расформировать один из классов, если не получается, вынести повторяющиеся методы в суперкласс

## CHANGE PREVENTERS (утяжелители изменений)

### Divergent Change (расходящиеся модификации)

При внесении изменений в класс приходится изменять большое число различных методов. Например, для добавления нового вида товара вам нужно изменить методы поиска, отображения и заказа товаров.

Решение: разделить класс на 2 или больше классов, если у классов есть повторяющееся поведение — сделать общий суперкласс

### Shotgun Surgery (стрельба дробью)

При выполнении любых модификаций приходится вносить множество мелких изменений в большое число классов. Причина — одна обязанность была разделена среди множества классов.

Решение: переместить методы разных классов с одной обязностью в 1 класс. При необходимости

### Parallel Inheritance Hierarchies (параллельниые иерархии наследования)

Всякий раз при создании подкласса какого-то класса приходится создавать ещё один подкласс для другого класса.

Решение: вы можете попытаться устранить дублирования паралельных классов в два этапа. Во-первых, нужно заставить экземпляры одной иерархии ссылаться на экземпляры другой иерархии. Затем следует убрать иерархию в ссылающемся классе c помощью перемещения метода и перемещения поля.

## DISPENSABLES (замусориватели)

### Comments (комментарии)

Метод содержит множество поясняющих комментариев.

Решение: если вы чувствуете, что фрагмент кода будет непонятным без комментария, попробуйте изменить структуру кода так, чтобы любые комментарии стали излишними

### Duplicate Code (дублирование кода)

Два фрагмента кода выглядят почти одинаковыми, могут называться по-разному, но делать одну и ту же задачу.

Решение: в зависимости от того, что дублируется:

* метод в одном классе — вынести метод в отдельный класс и вызывать его
* метод в двух подклассах одного суперкласса — вынести в суперкласс или сделать Template pattern — вынести структуру алгоритма и одинаковые шаги в суперкласс, а в подклассах оставить реализацию отличающихся шагов
* Дублирующийся код в двух разных классах — сделать суперкласс для обоих или извлечь дубляж в отдельный класс и делегировать ему выполнение

### Lazy Class (безработный класс)

На понимание и поддержку классов всегда требуются затраты времени и денег. А потому, если класс не делает достаточно много, чтобы уделять ему достаточно внимания, он должен быть уничтожен. Это может произойти если класс в результате рефакторинга стал очень маленьким.

Решение: слить подкласс и суперкласс воедино, или расформировать класс, перенеся его методы в другие

### Data Class (класс данных)

Классы данных – это классы, которые содержат только поля и простейшие методы для доступа к ним (геттеры и сеттеры). Это просто контейнеры для данных, используемые другими классами. Эти классы не содержат никакой дополнительной функциональности и не могут самостоятельно работать с данными, которыми владеют

Решение: сделать все поля и коллекции не-публичными, сделать доступ к ним только через геттеры и сеттеры. Проверить, можно ли из клиентских классов вынести функциональность в класс данных, возможно они там будут уместнее.

### Dead Code

Переменная, параметр, поле, метод или класс больше не используются (чаще всего потому, что устарели)

### Speculative Generality

Класс, метод, поле или параметр не используются

## COUPLERS (опутыватели связями)

### Feature Envy (функции-завистники)

Метод обращается к данным другого объекта чаще, чем к собственным данным.

Решение: то, что изменяется одновременно, нужно хранить в одном месте. Обычно данные и функции, использующие эти данные, также изменяются вместе, потому после переноса полей в класс данных, надо задуматься так же о переносе туда методов по работе с ними. Если класс использует функции нескольких других классов, нужно определеить в каком классе находится больше всего используемых данных и переместить в него метод вместе с остальными данными

Неуместно если отделение поведения от класса было сделано намеренно (паттерн *стратегия*, *посетитель*)

### Inappropriate Intimacy (неуместная близость)

Один класс использует служебные поля и методы другого класса.

Решение:

* перенести методы в другой класс
* изменить связь с двунаправленной на однонаправленную
* заменить делегирование наследованием

Неуместно, если используется паттерн *State*

### Message Chains (цепочка вызовов)

Вы видите в коде цепочки вызовов вроде такой $a→b()→c()→d()

Решение:

* скрыть делегирование — класса А делегирует работу классу Б, таким образом клиент перестанет знать о существовании класса Б
* посмотреть, для чего используется конечный объект. Возможно его можно вынести в отдельный метод

### Middle Man (делегатор)

Если класс выполняет одно действие – делегирует работу другому классу – стоит задуматься, зачем он вообще существует. Данный запах может быть результатом фанатичной борьбы с *Message Chains.*

Решение: если большую часть методов класс делегирует другому классу, нужно удалить посредника, вызывая методы напрямую

Не стоит трогать:

* если посредник был введен для избавления от нежелательной зависимости между классами
* некоторые паттерны намеренно создают посредников (*Proxy, Decorator*)

## INCOMPLETE LIBRARY CLASS (отсутствие нужного функционала у либы)

Библиотеки через некоторое время перестают удовлетворять требованиям пользователей.

Решение: сделать обертку для библиотеку, в которую дописать собственный функционал (*Decorator*)

## JAVASCRIPT ANTIPATTERNS

* Загрязнять глобальными переменными
* Передача строк вместо функций в setTimeout или setInterval
* Изменять базовый прототип Object
* Встраивать JS inline
* Использовать document.write