ASTON

Solid

astondevs.ru

S - Single responsibility

— принцип единственной ответственности

Модуль должен иметь только одну причину для изменения. Или: модуль должен отвечать только за одну заинтересованную группу.

Таким образом если одна из групп пользователей запросила изменения в работу кого-то модуля, то эти изменения не должны влиять на результаты работы других групп или работы других функций.

Кроме того это как правило это приводит к тому что у каждого класса может быть не более 1 – 2 метода.

Очень важно: данный принцип также подразумевает что вся бизнес логика должна быть собрана со всего проекта в одном месте

O - open-closed

— принцип открытости/закрытости

Программные сущности должны быть открыты для расширения и закрыты для изменения.

Когда вы меняете текущее поведение класса, эти изменения сказываются на всех системах, работающих с данным классом. Если хотите, чтобы класс выполнял больше операций, то идеальный вариант – не заменять старые на новые, а добавлять новые к уже существующим.

Принцип служит для того, чтобы делать поведение класса более разнообразным, не вмешиваясь в текущие операции, которые он выполняет. Благодаря этому вы избегаете ошибок в тех фрагментах кода, где задействован этот класс.

L - liskov substitution

— принцип подстановки Барбары Лисков

Имеет сложное математическое определение, которое можно заменить на: Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.

Это видно на простом примере создания объекта Collection.

List <String> strings = new ArrayList<>();

1) Также очень важно! Реализация в классах наследниках не должна противоречить реализации в базовых классах. В том числе нельзя закрывать методы в дочерних классах.
2) Нельзя вызывать методы не характерные для базовой реализации

I - interface segregation

— принцип разделения интерфейсов

Make fine grained interfaces that are client specific.

Под интерфейсом здесь понимается именно Java интерфейс. Разделение интерфейса облегчает использование и тестирование модулей.

Много интерфейсов лучше чем

D - dependency inversion

— принцип инверсии зависимости

Depend on abstractions, not on concretions.

- Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
- Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Что такое модули верхних уровней? Как определить этот уровень? Как оказалось, все очень просто. Чем ближе модуль к вводу/выводу, тем ниже уровень модуля. Т.е. модули, работающие с BD, интерфейсом пользователя, низкого уровня. А модули, реализующие бизнес-логику — высокого уровня.

KISS

Always Keep It Simple, Stupid (будь проще)

1. Ваши методы должны быть небольшими (40-50 строк).
2. Каждый метод решает одну проблему.
3. При модификации кода в будущем не должно возникнуть трудностей.
4. Система работает лучше всего, если она не усложняется без надобности.
5. Не устанавливайте целую библиотеку ради одной функции из неё.
6. Не делай того, что не просят.
7. Писать код необходимо надежно и «дубово».

YAGNI

You are not gonna need it (Вам это не понадобится)

1. Реализуйте только то, что нужно здесь и сейчас, а не в теории, что оно пригодится в будущем.
2. Подчищайте ненужный код (найдите через Git историю при надобности).
3. Программист не должен добавлять новый функционал, о котором его не просят (благими намерениями без должной проверки вы только добавите багов).

DRY

Don’t Repeat Yourself (Не повторяйся)

1. Избегайте копирования кода.
2. Выносите общую логику.
3. Прежде чем добавлять функционал, проверьте в проекте, может, он уже создан.
4. Константы.

GRASP

Самостоятельное изучение

Литература

- https://habr.com/ru/post/508086/
- https://www.youtube.com/watch?v=O4uhPCEDzSo
- https://www.youtube.com/watch?v=x5OtQiKOG-Q&t=314s
- https://www.youtube.com/watch?v=otrfSgeK3JI