Принципы SOLID

**Вот что входит в принципы SOLID:**

1. **S**ingle Responsibility Principle (Принцип единственной ответственности).
2. **O**pen Closed Principle (Принцип открытости/закрытости).
3. **L**iskov’s Substitution Principle (Принцип подстановки Барбары Лисков).
4. **I**nterface Segregation Principle (Принцип разделения интерфейса).
5. **D**ependency Inversion Principle (Принцип инверсии зависимостей).

Принцип единственной ответственности (SRP)

Данный принцип гласит: **никогда не должно быть больше одной причины изменить класс.**

На каждый объект возлагается одна обязанность, полностью инкапсулированная в класс. Все сервисы класса направлены на обеспечение этой обязанности.

**Проще говоря один класс или же сущность отвечает только за одну задачу.**

Принцип открытости/закрытости (OCP)

Этот принцип емко описывают так: **программные сущности (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменения.**

Это означает, что должна быть возможность изменять внешнее поведение класса, не внося физические изменения в сам класс. Следуя этому принципу, классы разрабатываются так, чтобы для подстройки класса к конкретным условиям применения было достаточно расширить его и переопределить некоторые функции.

Принцип подстановки Барбары Лисков (LSP)

Это вариация принципа открытости/закрытости, о котором говорилось ранее. Его можно описать так: **объекты в программе можно заменить их наследниками без изменения свойств программы.**

Это означает, что класс, разработанный путем расширения на основании базового класса, должен переопределять его методы так, чтобы не нарушалась функциональность с точки зрения клиента. То есть, если разработчик расширяет ваш класс и использует его в приложении, он не должен изменять ожидаемое поведение переопределенных методов.

Принцип разделения интерфейса (ISP)

Характеризуется следующим утверждением**: клиенты не должны быть вынуждены реализовывать методы, которые они не будут использовать.**

Принцип разделения интерфейсов говорит о том, что слишком «толстые» интерфейсы необходимо разделять на более мелкие и специфические, чтобы клиенты мелких интерфейсов знали только о методах, необходимых в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип инверсии зависимостей (DIP)

Этот принцип SOLID в Java описывают так: **зависимости внутри системы строятся на основе абстракций. Модули верхнего уровня не зависят от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций**.

Программное обеспечение нужно разрабатывать так, чтобы различные модули были автономными и соединялись друг с другом с помощью абстракции.