SOLID

Принципы **SOLID** — это набор из пяти рекомендаций, предложенных для создания масштабируемого, легко поддерживаемого и понятного программного обеспечения. Эти принципы включают:

1. **Принцип единственной ответственности (Single Responsibility Principle, SRP):**
    - Этот принцип гласит, что класс должен иметь только одну причину для изменения.
    - Каждый класс должен делать только одну важную вещь. Не пытайтесь упаковать много разных задач в один класс.
    - *Пример:* Разделение класса, отвечающего за операции с файлами, на два класса: один для чтения из файла, другой для записи в файл.
2. **Принцип открытости/закрытости (Open/Closed Principle, OCP):**
    - Классы должны быть открытыми для расширения, но закрытыми для изменения.
    - Когда вы пишете код, пытайтесь сделать его таким, чтобы другие могли добавлять новые функции, не трогая старый код. Не изменяйте существующий код, если необходимо добавить что-то новое.
    - *Пример:* Создание абстрактного класса для отчетов, который можно расширить для создания различных типов отчетов, не изменяя исходный класс.
3. **Принцип подстановки Барбары Лисков (Liskov Substitution Principle, LSP):**
    - Объекты должны быть заменяемыми на экземпляры их подтипов без изменения желаемых свойств программы.
    - Если вы создаете новые классы, чтобы расширить функциональность старых классов, убедитесь, что новые классы могут использоваться точно так же, как и старые, без сломанных вещей.
    - *Пример:* Если класс "Птица" имеет метод "летать", то класс "Пингвин" не должен наследоваться от класса "Птица", так как пингвины не летают.
4. **Принцип разделения интерфейса (Interface Segregation Principle, ISP):**

- Клиенты не должны быть вынуждены зависеть от интерфейсов, которые они не используют.
- Когда вы создаете интерфейсы (способы взаимодействия с объектами), сделайте их небольшими и специфическими, чтобы клиенты могли использовать только то, что им нужно, не нагромождая ничего лишнего.
- *Пример:* Вместо создания одного общего интерфейса для управления принтером, можно создать несколько маленьких интерфейсов для различных операций (печать, сканирование, факс).

5. **Принцип инверсии зависимостей (Dependency Inversion Principle, DIP):**
    - Модули верхних уровней не должны зависеть от модулей нижних уровней, оба типа модулей должны зависеть от абстракций.
    - Вместо того, чтобы классы зависели от деталей реализации, они должны зависеть от абстракций (обобщенных идей), которые могут быть легко заменены другими абстракциями. Это делает ваш код более гибким и менее привязанным к конкретным реализациям.
    - *Пример:* Вместо жесткой привязки к конкретному классу базы данных, код может зависеть от абстрактного интерфейса для доступа к базе данных.

Эти принципы помогают обеспечить высокий уровень модульности, расширяемости и поддерживаемости кода, что делает разработку программного обеспечения более эффективной и менее затратной в долгосрочной перспективе.