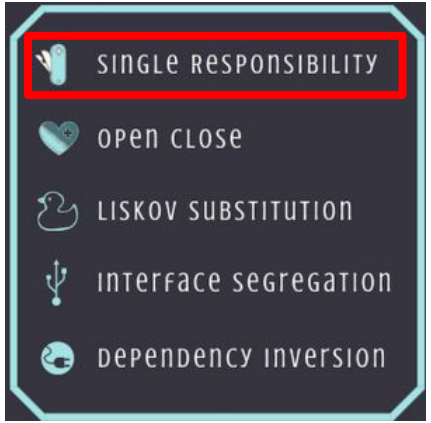
SOLID
Single Responsibility Principle

Upcode Software
Engineer Team

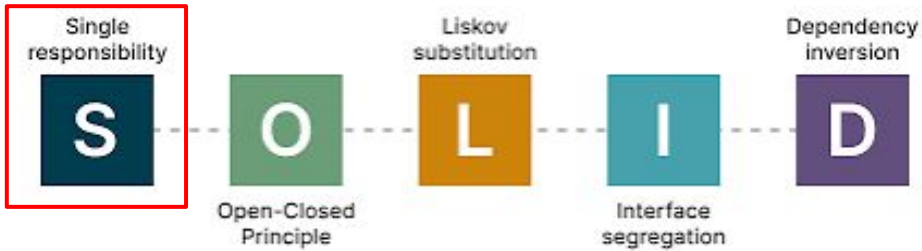
-2023-

CONTENT / КОНТЕНТ

1. **What is SOLID ?** Что такое SOLID ?
    a. Single Responsibility Principle
2. **Why needs to SRP ?** Зачем это НУЖНО ?
3. **How to use SRP ?** Как это использовать ?
4. **Where to use SRP ?** Где это НУЖНО ?
5. **SOURCE code.** Исходный Код.



1. Что такое SOLID ? (What is SOLID) ? (1/n)



SOLID - это аббревиатура, которая обозначает **пять основных принципов объектно-ориентированного программирования (ООП)**. Каждая буква соответствует одному из этих принципов:

1. **S -** Single Responsibility Principle (Принцип единственной ответственности)
2. **O -** Open/Closed Principle (Принцип открытости/закрытости)
3. **L -** Liskov Substitution Principle (Принцип подстановки Барбары Лисков)
4. **I -** Interface Segregation Principle (Принцип разделения интерфейса)
5. **D -** Dependency Inversion Principle (Принцип инверсии зависимостей)

Single Responsibility Principle (SRP) (Принцип единственной ответственности)

- **Single Responsibility Principle** (Принцип единственной ответственности) - это принцип объектно-ориентированного программирования, который гласит, что каждый класс или модуль должен иметь только одну ответственность, то есть выполнять только одну задачу.
- Суть этого принципа заключается в том, что классы или модули, которые занимаются слишком многим, становятся сложными и трудными для понимания, изменения и тестирования.
- Если класс или модуль имеет только одну ответственность, то он будет более гибким и легким для поддержки.

1. (What is SOLID) ? (2/n)
Что такое SOLID ? (2/n)

- Эти принципы были разработаны для обеспечения более гибкого, расширяемого и понятного кода.
- Они помогают разработчикам создавать программное обеспечение, которое легко изменять и расширять, а также понятное для других разработчиков.
- В целом, принцип единственной ответственности помогает создавать более чистый, понятный и гибкий код, что улучшает качество и производительность программного обеспечения.
- В целом, принцип единственной ответственности помогает создавать более чистый, понятный и гибкий код, что улучшает качество и производительность программного обеспечения.

Why needs to SRP ?
Зачем это НУЖНО ?

- Принцип единственной ответственности, как и другие принципы SOLID, был разработан для **облегчения разработки и поддержки программного обеспечения.** Если каждый класс или модуль выполняет только одну задачу, то код становится более понятным, легко изменяемым и тестируемым.
- Принцип единственной ответственности также помогает повысить гибкость и расширяемость системы, так как изменения, связанные с одной ответственностью, не затрагивают другие. Это может сократить время разработки и снизить риски ошибок.

Кроме того, использование принципа единственной ответственности может способствовать повышению качества кода, так как он становится более структурированным и легко читаемым для других разработчиков. Это особенно важно в командной разработке, где различные разработчики могут работать с одним и тем же кодом.

How to use SRP (1/n)?
Как это использовать (1/n)?

- Определите ответственности классов или модулей на начальном этапе проектирования системы. Не пытайтесь заложить сразу все функции в один класс или модуль.

- Разделите функциональность системы на **независимые модули или классы**, каждый из которых будет отвечать за свою задачу.

- При написании кода старайтесь следить за тем, чтобы **каждый класс или модуль выполнял только одну ответственность**, и не пытайтесь добавлять в них новые функции, которые не относятся к этой ответственности.

How to use SRP (2/n)?
Как это использовать (2/n)?

- **Используйте интерфейсы и абстракции** для разделения функциональности между различными классами или модулями. Это позволяет легко заменять один класс другим без изменения кода других частей системы.

- Регулярно проводите **рефакторинг кода**, чтобы убедиться, что каждый класс или модуль по-прежнему выполняет только одну ответственность, и не добавляет новых функций, которые могут привести к его "нагромождению".

- Используйте **логирование и мониторинг** для отслеживания работы системы и быстрого выявления ошибок или проблем.

Where to use SRP (1/n) ?
Где это НУЖНО (1/n) ?

- **Разработка веб-приложений:** веб-приложения обычно состоят из множества компонентов, таких как контроллеры, модели и представления.

- **Разработка мобильных приложений:** мобильные приложения, как и веб-приложения, обычно состоят из множества компонентов. Принцип единственной ответственности помогает создавать более чистый и структурированный код, что облегчает его тестирование и поддержку.

- **Разработка игр:** в игровой разработке могут быть множество компонентов, таких как игровые объекты, искусственный интеллект, физический движок и т. д.

Where to use SRP (2/n) ?
Где это НУЖНО (2/n) ?

- **Разработка многопоточных приложений:** многопоточные приложения могут быть сложными в написании и тестировании.

- **Разработка библиотек и фреймворков:** при разработке библиотек и фреймворков особенно важно следовать принципу единственной ответственности, чтобы пользователи могли легко использовать их функции без необходимости изучения большого количества документации и кода.

- **Разработка системного программного обеспечения:** для создания более стабильных и безопасных системных приложений.

Source Code SRP (1/n) ?
Исходный Код (1/n) ?

- **Customer нарушает принцип единственной ответственности**, так как он отвечает не только за хранение данных, но и за выполнение других функций, таких как сохранение, валидация, отправка электронной почты и вычисление счета.

```
public class Customer {

    private String name;
    private String email;
    private String city;
    private String state;

    public void setName(String name) {
        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public void setZip(String zip) {
        this.zip = zip;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public String getStreet() {
        return street;
    }

    public String getZip() {
        return zip;
    }

    public void save() {
        // save customer to database
    }

    public void validate() {
        // validate customer data
    }

    public void sendEmail() {
        // send email to customer
    }

    public void calculateInvoice() {
        // calculate customer invoice
    }
}
```

Source Code (S) ?
Исходный Код (S) ?

- **Customer и Address отвечают только за хранение данных и не выполняют других функций.**

```java
public class Customer {
    private String name;
    private String email;
    private Address address;

    public void setName(String name) {
        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setAddress(Address address)
{
        this.address = address;
    }

    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public Address getAddress() {
        return address;
    }
}
```

```java
public class Address {
    private String city;
    private String state;

    public void setCity(String city) {
        this.city = city;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getCity() {
        return city;
    }

    public String getState() {
        return state;
    }
}
```

Summary

A class should have just one reason to change. Try to make every class responsible for a single part of the functionality provided by the software, and make that responsibility entirely encapsulated by (you can also say hidden within) the class.

Reference Resources

1. The SOLID Principles of Object-Oriented Programming Explained in Plain English
2. SOLID: The First 5 Principles of Object Oriented Design
3. Dive into Design Patterns

Thank you !

Presented by **TURSUNOV NODIRKHOJA**

AMERIQANO@GMAIL.COM