Принципы SOLID

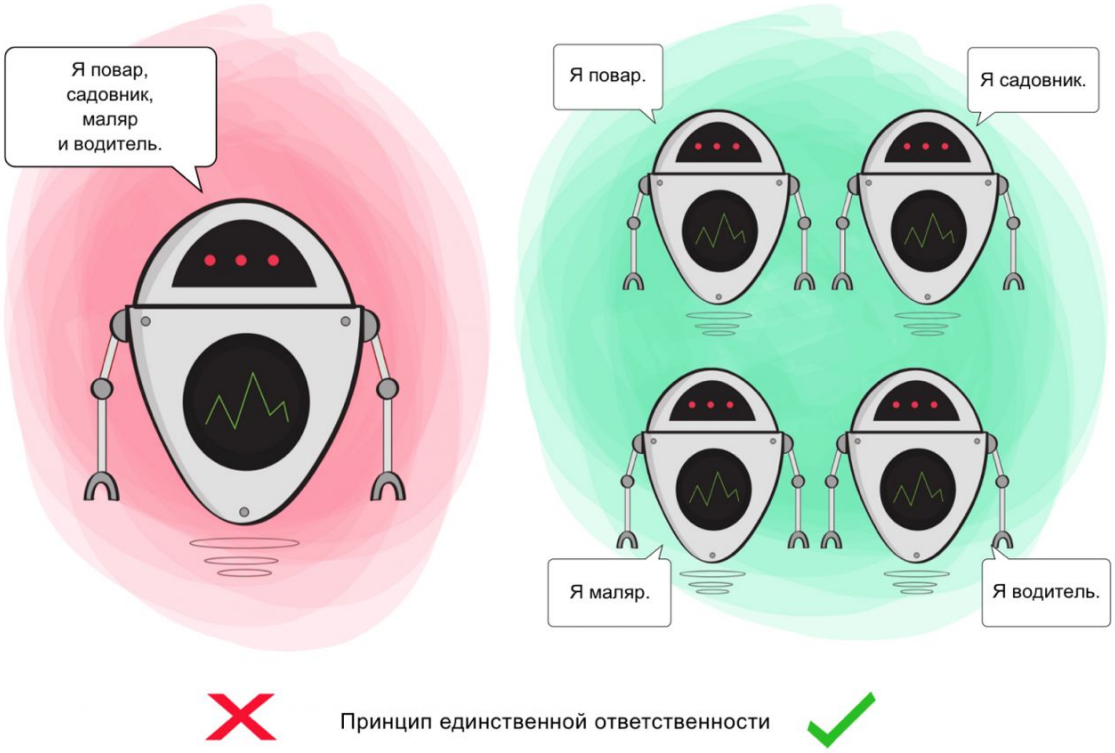
Inspired from a couple of articles from **habr.com**

«СОЗДАВАТЬ ПОНЯТНЫЙ, ЧИТАЕМЫЙ, ТЕСТИРУЕМЫЙ КОД, НАД КОТОРЫМ СМОГУТ СОВМЕСТНО РАБОТАТЬ МНОГИЕ РАЗРАБОТЧИКИ».

Принцип	Смысл
Принцип единственной ответственности	У класса должна быть всего одна причина для изменения.
Принцип открытости/закрытости	Сущности программы (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменений.
Принцип подстановки Барбары Лисков	Объекты в программе должны быть заменяемы экземплярами их подтипов без ущерба корректности работы программы.
Принцип разделения интерфейсов	Ни один клиент не должен зависеть от методов, которые он не использует.
Принцип инверсии зависимостей	Модуль высокого уровня не должен зависеть от модулей низкого уровня. И то, и другое должно зависеть от абстракций. Абстракции не должны зависеть от деталей реализации. Детали реализации должны зависеть от абстракций.

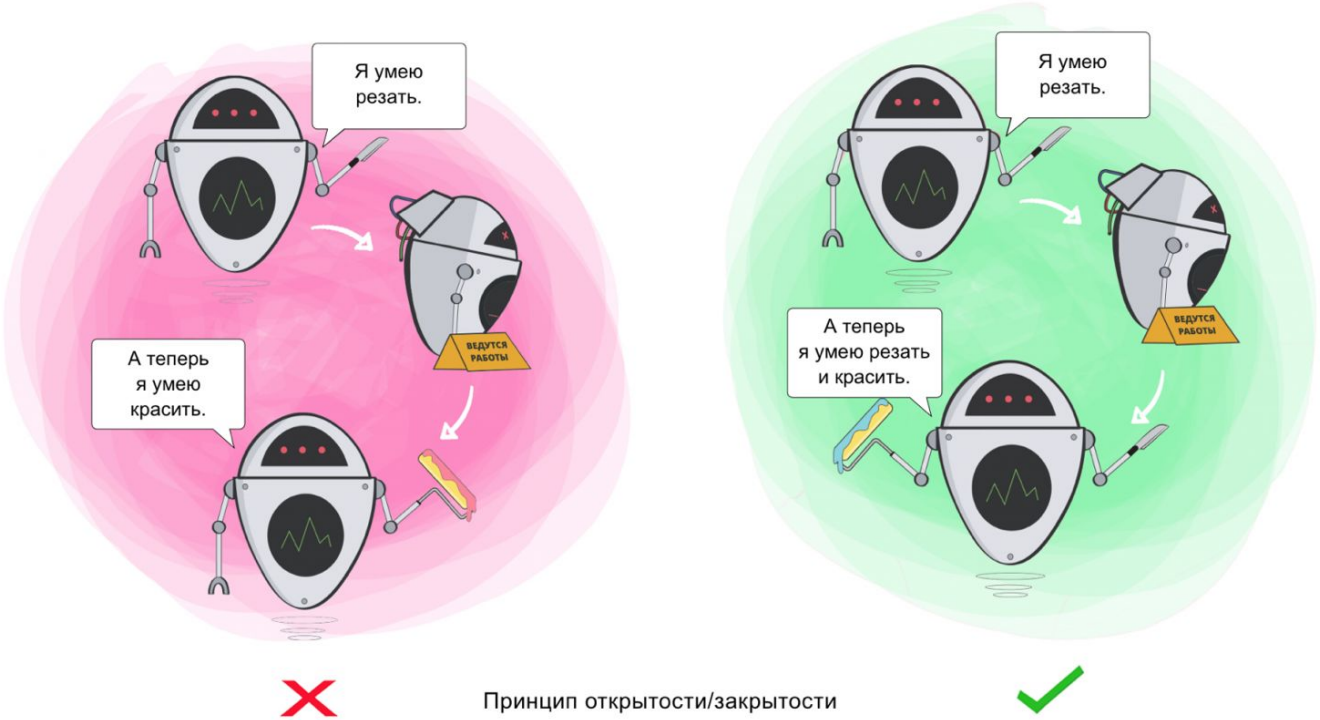
S – Single Responsibility (Принцип единственной ответственности)

> *Каждый класс должен отвечать только за одну операцию.*



O — Open-Closed (Принцип открытости-закрытости)

> *Классы должны быть открыты для расширения, но закрыты для модификации.*



Принцип открытости/закрытости

```cpp
enum class SensorModel {
    Good,
    Better
};

struct DistanceSensor {
    DistanceSensor(SensorModel model) : mModel{model} {}
    int getDistance() {
      switch (mModel) {
        case SensorModel::Good :
          // Business logic for "Good" model
        case SensorModel::Better :
          // Business logic for "Better" model
      }
    }
};
```

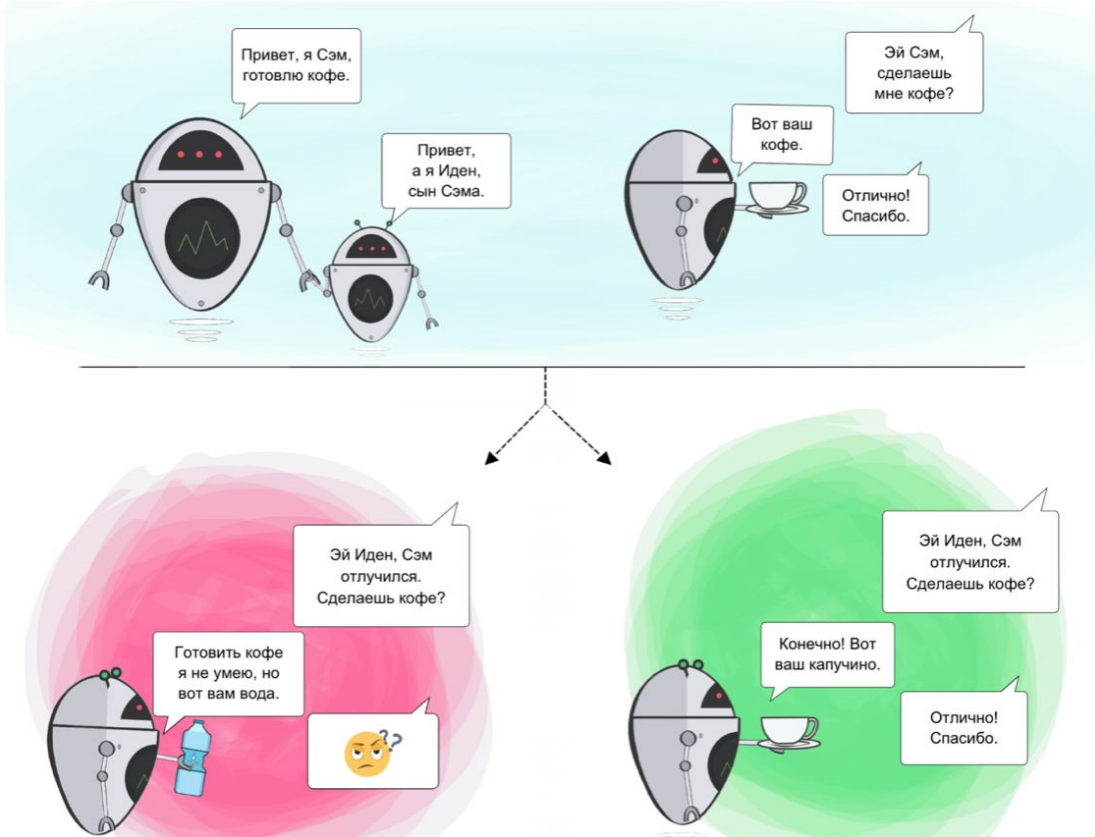
```cpp
struct DistanceSensor {
    virtual ~DistanceSensor() = default;
    virtual int getDistance() = 0;
};

struct GoodDistanceSensor : public DistanceSensor {
    int getDistance() override {
      // Business logic for "Good" model
    }
};

struct BetterDistanceSensor : public DistanceSensor {
    int getDistance() override {
      // Business logic for "Better" model
    }
};
```

L — Liskov Substitution (Принцип подстановки Барбары Лисков)

> *Если П является подтипом Т, то любые объекты типа Т, присутствующие в программе, могут заменяться объектами типа П без негативных последствий для функциональности программы.*



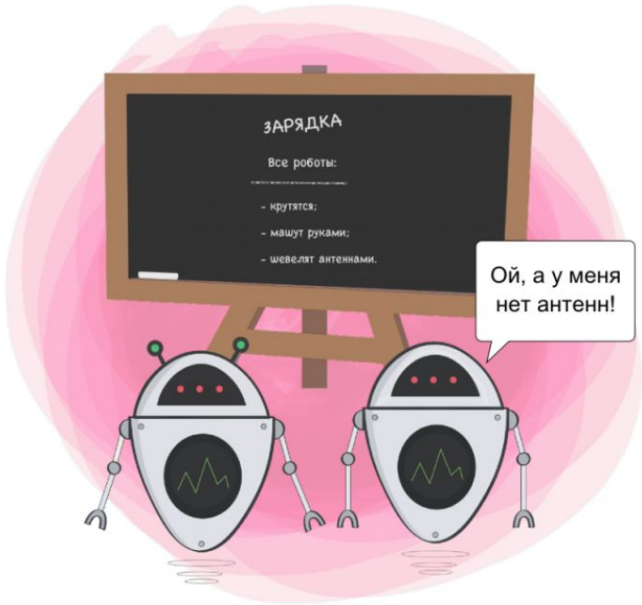
```cpp
struct InertialMeasurementUnit {
    virtual ~InertialMeasurementUnit() = default;
    /**
     * Sets the frequency of measurements
     * @param frequency (in Hertz)
     * @return Whether frequency was valid
     */
    virtual bool setFrequency(double frequency) = 0;
};
struct Gyroscope : public InertialMeasurementUnit {
    // Valid range [0.5, 10]
    bool setFrequency(double frequency) override;
};
struct Accelerometer : public InertialMeasurementUnit {
    // Valid range [0.1, 100]
    bool setFrequency(double frequency) override;
};
```

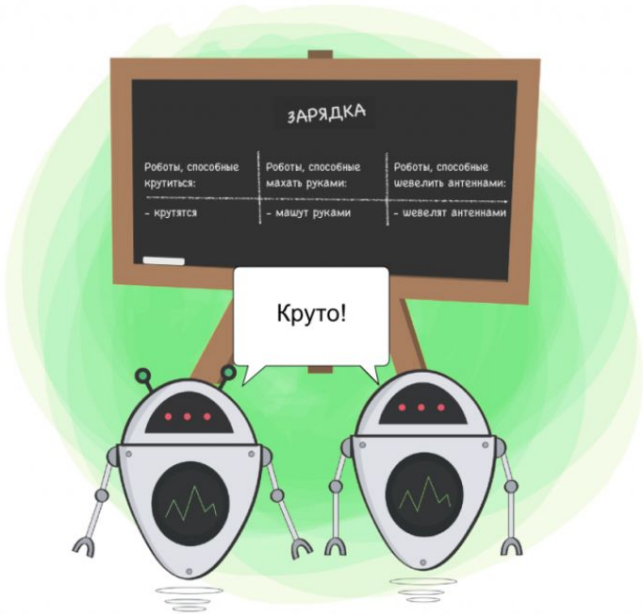
```cpp
struct InertialMeasurementUnit {
    virtual ~InertialMeasurementUnit() = default;
    /**
     * Sets the frequency of measurements
     * @param frequency (in Hertz)
     * @throw std::out_of_range exception if frequency is invalid
     */
    virtual void setFrequency(double frequency) = 0;

    /**
     * Provides the valid measurement range
     * @return <minimum frequency, maximum frequency>
     */
    virtual pair<double, double> getFrequencyRange() const = 0;
};
```

I — Interface Segregation (Принцип разделения интерфейсов)

> *Не следует ставить клиент в зависимость от методов, которые он не использует.*





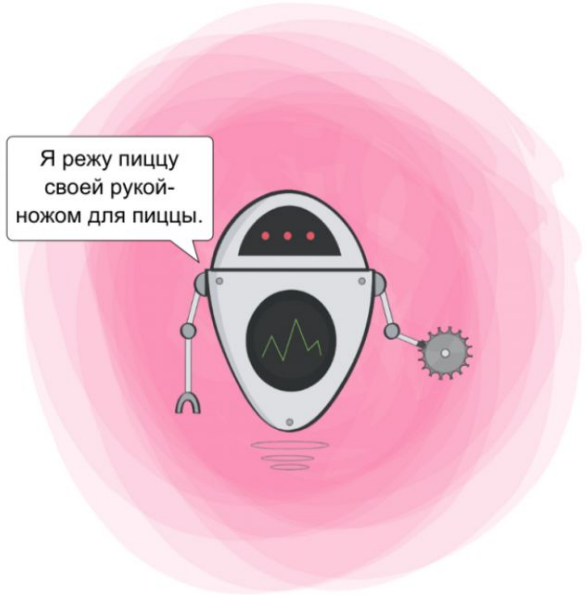
✗ Принцип разделения интерфейсов ✓

```cpp
struct IMachine {
    virtual void print(Document &doc) = 0;
    virtual void fax(Document &doc) = 0;
    virtual void scan(Document &doc) = 0;
};
struct MultiFunctionPrinter : IMachine {      // OK
    void print(Document &doc) override { }
    void fax(Document &doc) override { }
    void scan(Document &doc) override { }
};
struct Scanner : IMachine {                   // Not OK
    void print(Document &doc) override { /* Blank */ }
    void fax(Document &doc) override { /* Blank */ }
    void scan(Document &doc) override {
      // Do scanning ...
    }
};
```

```cpp
/* ------------------------------ Interfaces ---------------------------- */
struct IPrinter {
    virtual void print(Document &doc) = 0;
};
struct IScanner {
    virtual void scan(Document &doc) = 0;
};
/* ---------------------------------------------------------------------- */
struct Printer : IPrinter {
    void print(Document &doc) override;
};
struct Scanner : IScanner {
    void scan(Document &doc) override;
};
struct IMachine : IPrinter, IScanner { };
struct Machine : IMachine {
    IPrinter&   m_printer;
    IScanner&   m_scanner;
    Machine(IPrinter &p, IScanner &s) : printer{p}, scanner{s} { }
    void print(Document &doc) override { printer.print(doc); }
    void scan(Document &doc) override { scanner.scan(doc); }
};
```

D — Dependency Inversion (Принцип инверсии зависимостей)

> *Модули верхнего уровня не должны зависеть от модулей нижнего уровня. И те, и другие должны зависеть от абстракций. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.*



```
struct YandexCloud {
    void uploadToCloud(string filepath) { /* ... */ }
};

struct FileUploader {
    FileUploader(YandexCloud& yaCloud);
    void scheduleUpload(string filepath);
};
```

```
struct Cloud {
    virtual ~Cloud() = default;
    virtual void uploadToCloud(string filepath) = 0;
};

struct YandexCloud : public Cloud {
    void uploadToCloud(string filepath) override { /* ... */ }
};

struct FileUploader {
    FileUploader(Cloud& cloud);
    void scheduleUpload(string filepath);
};
```

Ссылки:

1.Anti-patterns:https://techrocks.ru/2020/08/26/solid-principles-in-plain-russian/

2. Habr article: https://habr.com/ru/company/productivity_inside/blog/505430/

3. Examples: https://platis.solutions/blog/2020/06/22/how-to-write-solid-cpp/