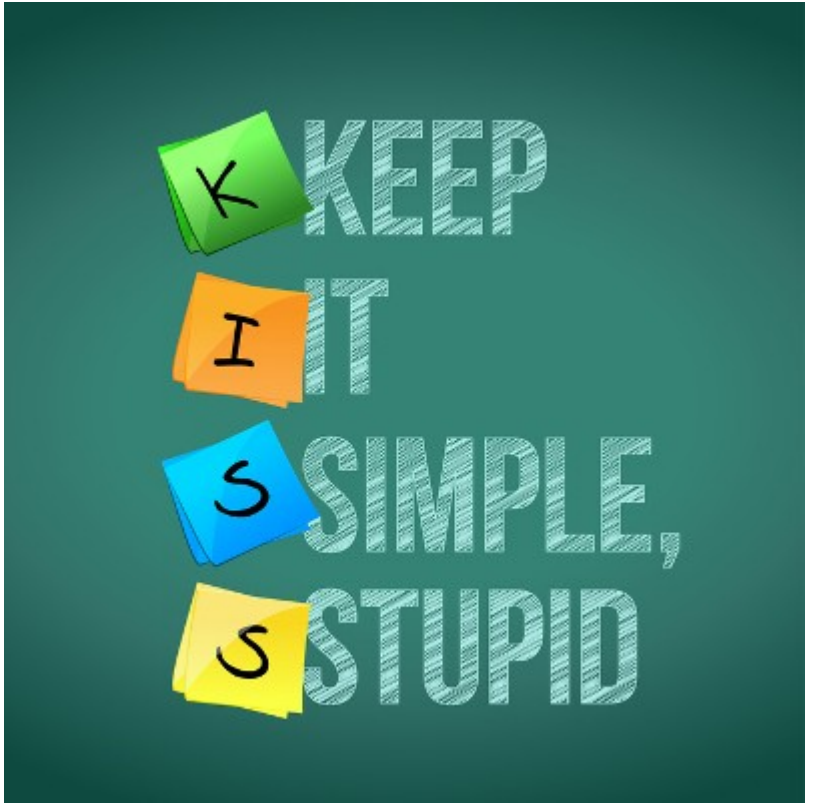
Промышленное программирование

Лекция 6

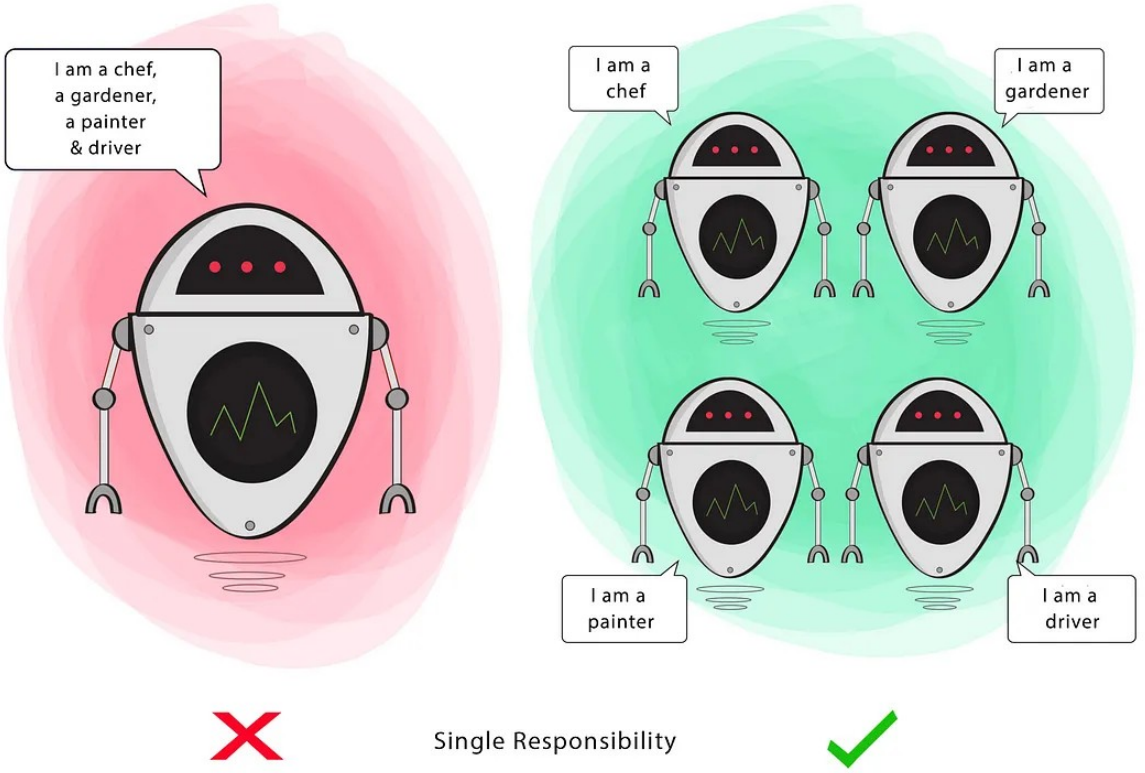
1. Принципы проектирования
2. Протокол HTTP

DRY & KISS

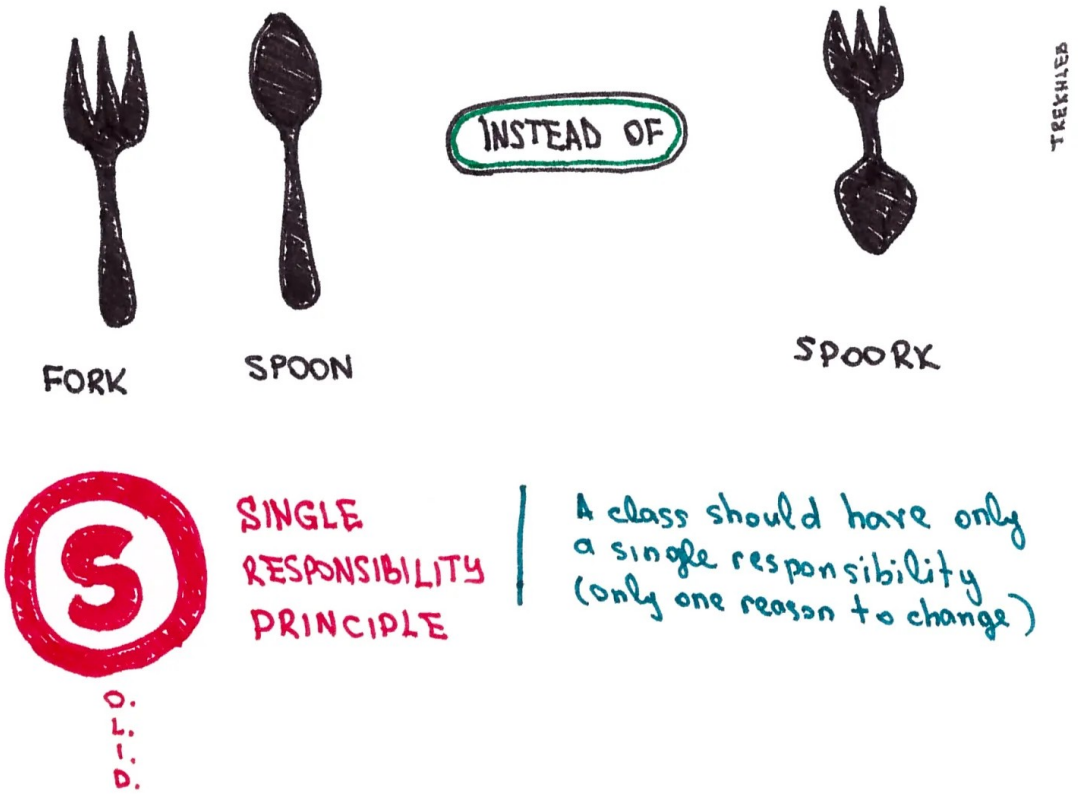
Teachers: your code should follow the principle of DRY: Don’t Repeat Yourself
My code:



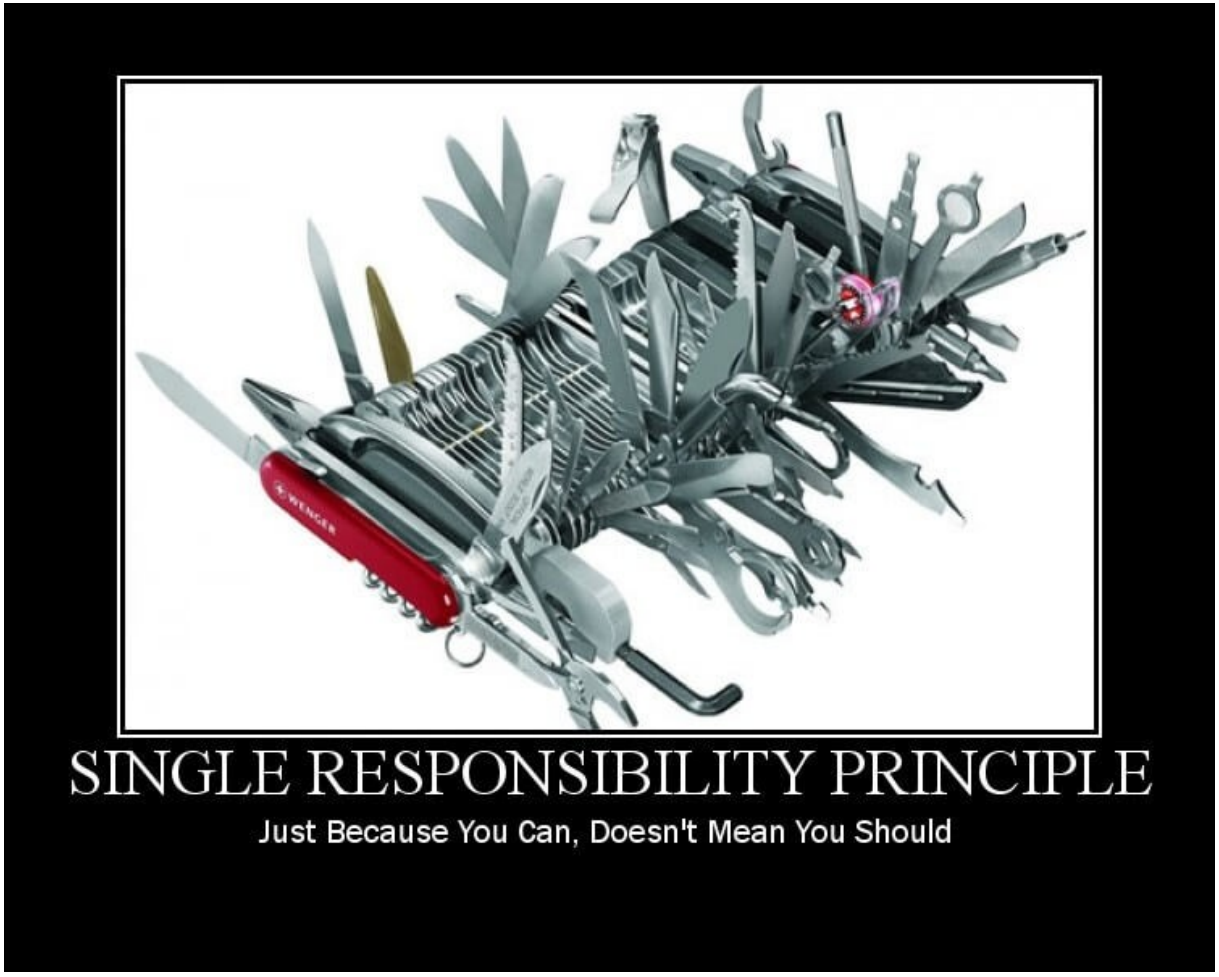
SOLID (1): SRP


I am a chef,
a gardener,
a painter
& driver
I am a
chef
I am a
gardener
I am a
painter
I am a
driver
Single Responsibility

SOLID (1): SRP


FORK
SPOON
INSTEAD OF
SPOORK
TREKHLEB
S
O.
L.
I.
D.
SINGLE
RESPONSIBILITY
PRINCIPLE
A class should have only
a single responsibility
(only one reason to change)

SOLID (1): SRP


SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

SOLID (1): SRP

```
class AggregationCalculator
{
    public AggregationCalculator(IEnumerable<?> items);

    public float Compute();

    public string Print()
    {
        return $"AverageMark: {Compute()}";
    }
}
```

SOLID (1): SRP

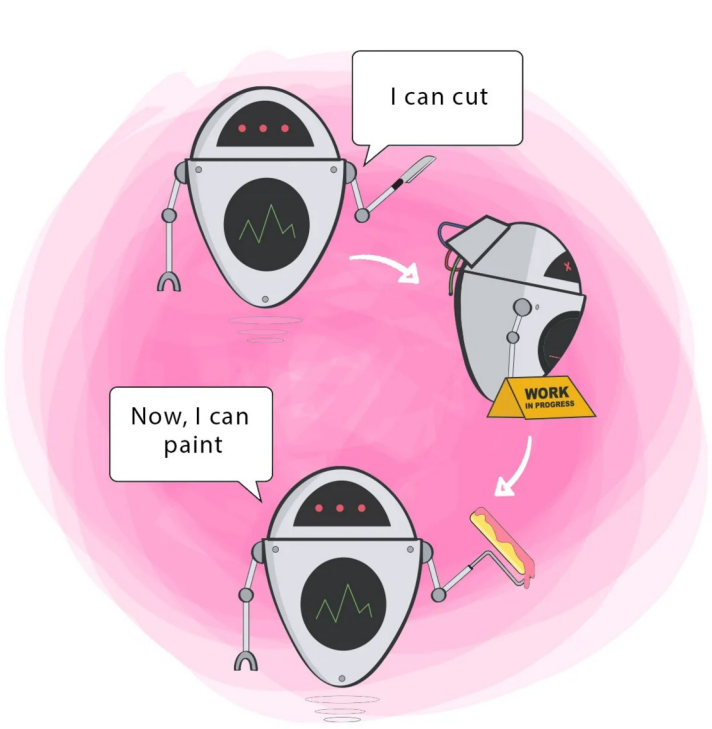
```
class AggregationCalculator
{
    public AggregationCalculator(IEnumerable<?> items);

    public float Compute();
}

class AggregationCalculatorPrinter
{
    public AggregationCalculatorPrinter(AggregationCalculator aggregationCalculator);

    public string ToJson();
    public string ToXml();
}
```

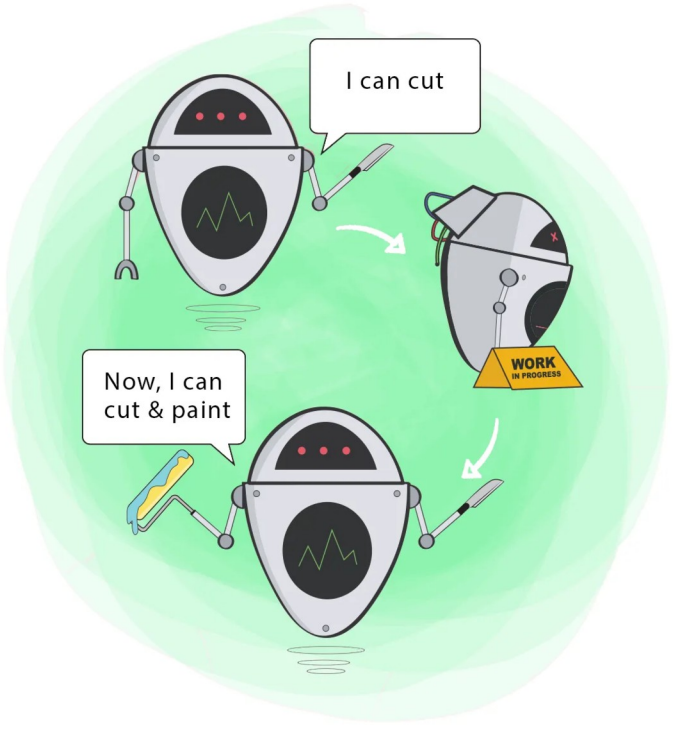
SOLID (1): SRP

Каждый класс должен иметь только одну зону ответственности.

Класс должен иметь только одну причину для своего изменения.
Только одно потенциальное изменение в ТЗ (логика хранения,
или логика передачи, или логика журналирования, и т. п.)
должно приводить к изменению спецификации на класс.

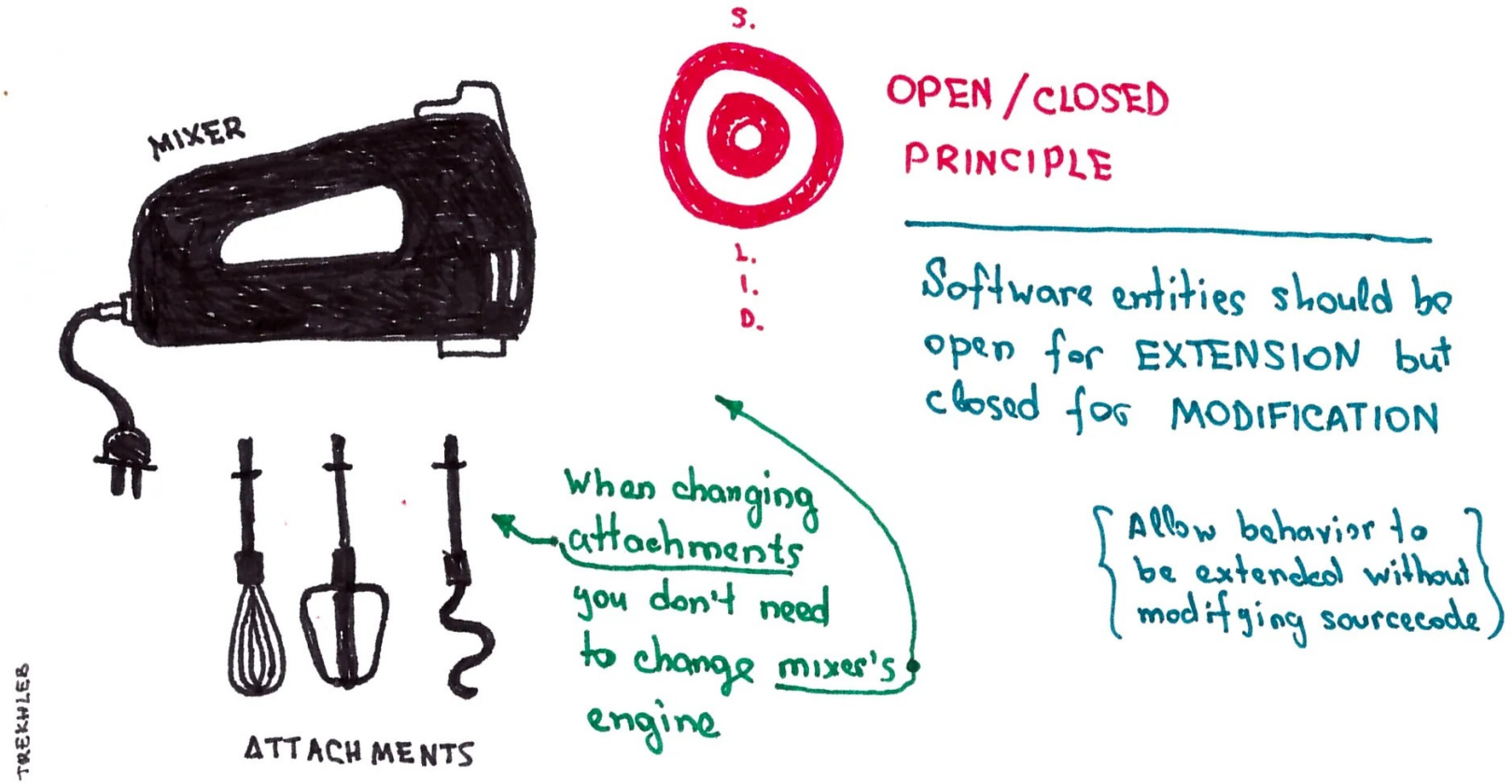
SOLID (2): OCP



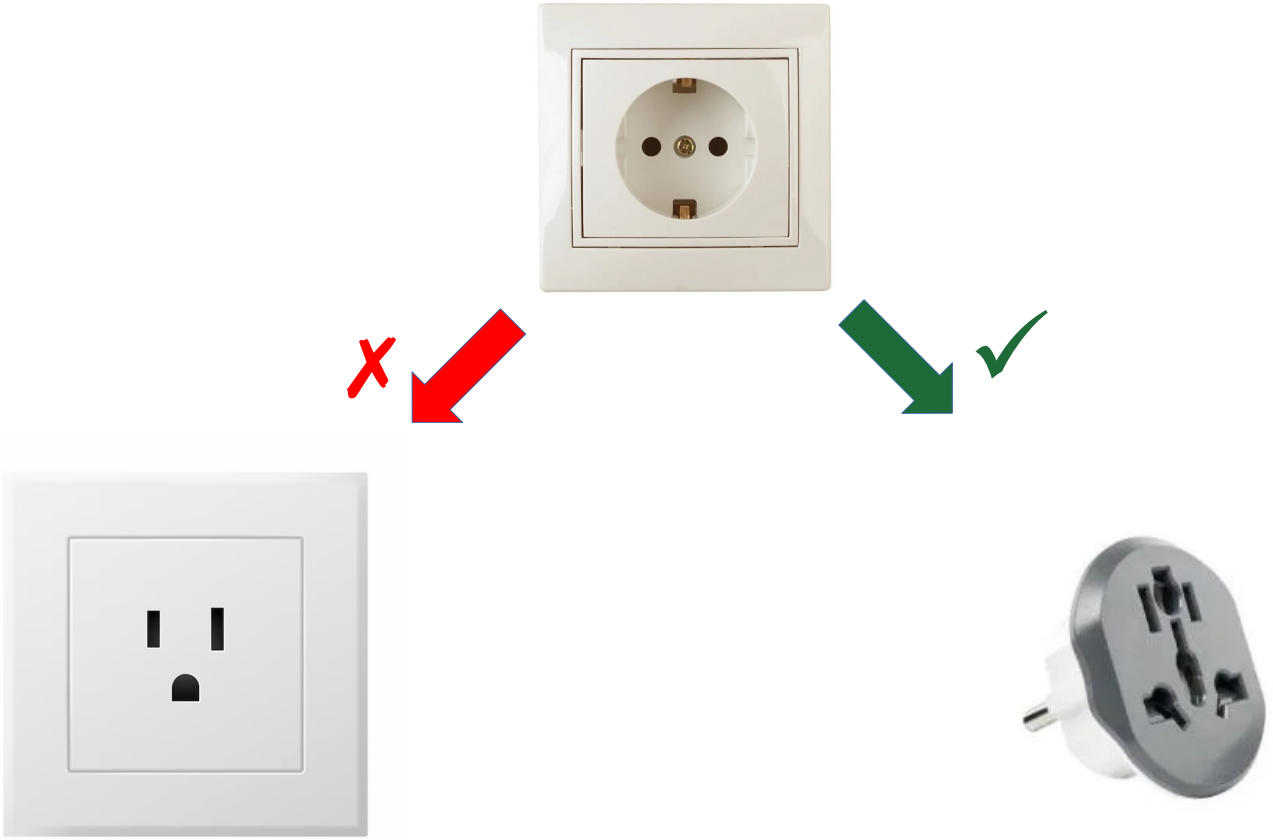


Open-Closed

SOLID (2): OCP


MIXER
ATTACHMENTS
TREKWLEB
S.
L.
I.
D.
OPEN / CLOSED
PRINCIPLE
Software entities should be open for EXTENSION but closed for MODIFICATION
When changing attachments you don't need to change mixer's engine
Allow behavior to be extended without modifying sourcecode

SOLID (2): OCP



SOLID (2): OCP

```
class Square: Figure { public float Length { get; set; } }

class Circle: Figure { public float Radius { get; set; } }

static class FigureExtensions
{
    public static float GetTotalArea(IEnumerable<Figure> items)
    {
        return items.Select(GetArea).DefaultIfEmpty().Sum();

        static float GetArea(Figure f) => f switch
        {
            Square s => s.Length * s.Length,
            Circle c => (float) Math.PI * c.Radius * c.Radius,
            _ => throw new NotSupportedException(f.GetType().Name)
        };
    }
}
```

SOLID (2): OCP

```
abstract class Figure
{
    public float Area => GetArea();
    protected abstract float GetArea();
}

…

static class FigureExtensions
{
    public static float GetTotalArea(IEnumerable<Figure> items)
    {
        return items.Select(o => o.GetArea()).DefaultIfEmpty().Sum();
    }
}
```

SOLID (2): OCP

```
public class FigurePersistence
{
    public void SaveToDb(Figure f, string connectionString);
    public void SaveToFile(Figure f, string path);
}
```

```
public interface FigurePersistence
{
    void Save(Figure f);
}

public void DbFigurePersistence()
{
    public DbFigurePersistence(string connectionString);

    public void Save(Figure f);
}

…
```

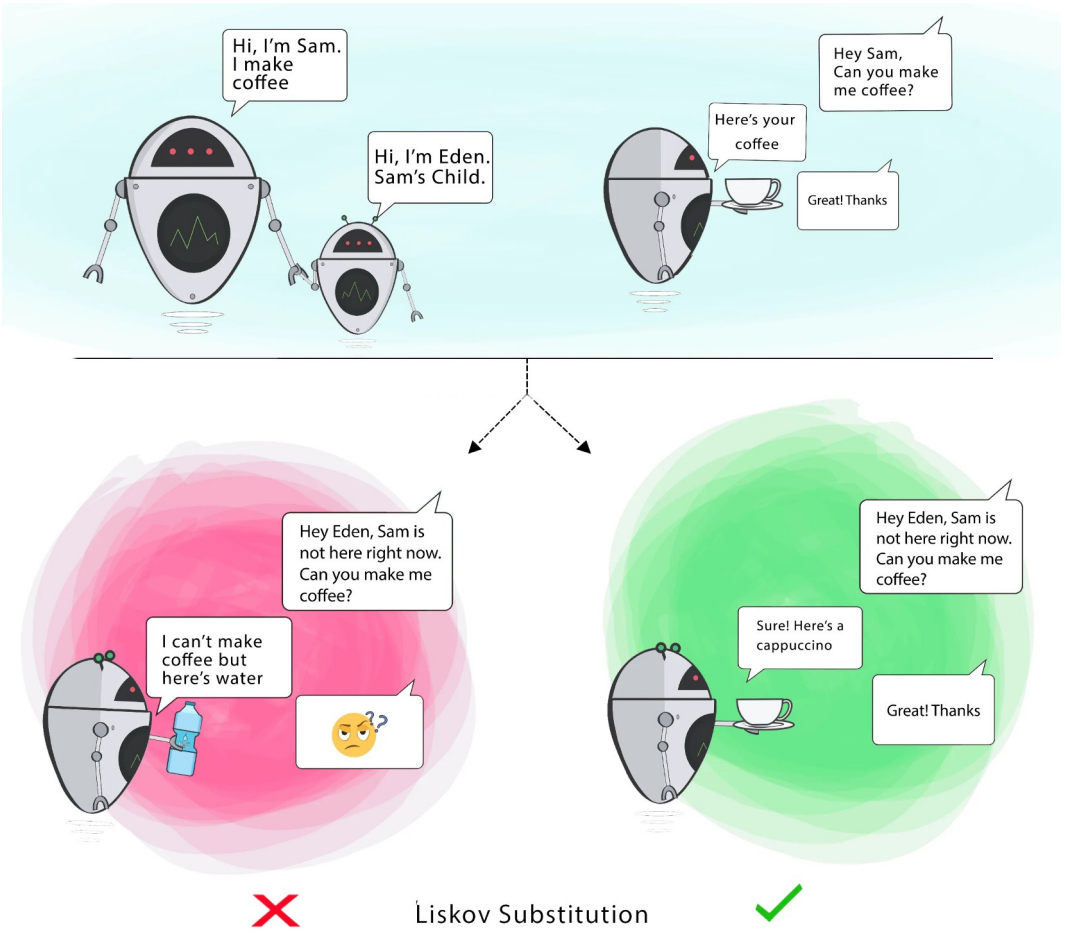
SOLID (2): OCP

**Программные сущности (классы, модули, функции и т. п.)
должны быть открыты для расширения, но закрыты для изменения.**

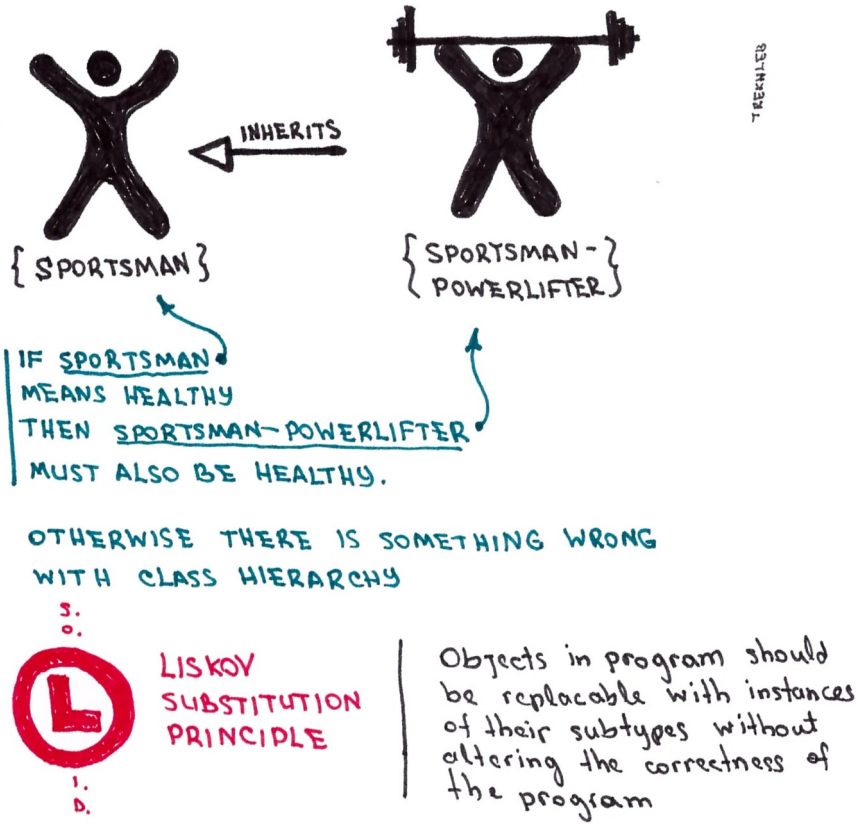
ПО изменяется путём добавления нового кода, а не изменения существующего. Изначальный код остаётся «нетронутым» и стабильным, новый функционал внедряется через *наследование* реализации или использование интерфейсов.

Созданный изначально интерфейс должен быть закрыт для модификаций, а новые реализации как минимум соответсвуют этому изначальному интерфейсу, но могут поддерживать и другие, более расширенные.

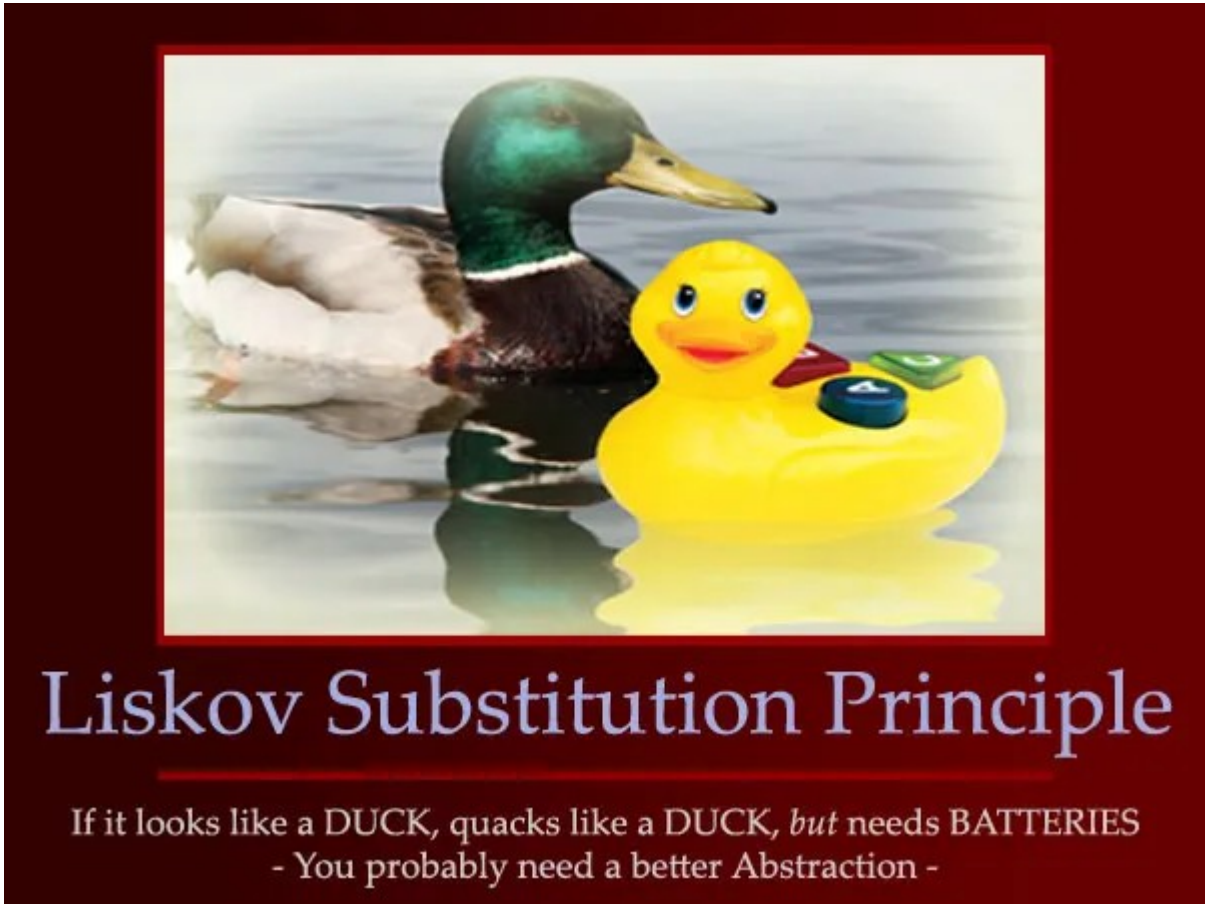
SOLID (3): LSP


Hi, I'm Sam. I make coffee
Hi, I'm Eden. Sam's Child.
Hey Sam, Can you make me coffee?
Here's your coffee
Great! Thanks
Hey Eden, Sam is not here right now. Can you make me coffee?
I can't make coffee but here's water
Hey Eden, Sam is not here right now. Can you make me coffee?
Sure! Here's a cappuccino
Great! Thanks
Liskov Substitution

SOLID (3): LSP


INHERITS
{ SPORTSMAN }
{ SPORTSMAN - POWERLIFTER }
IF SPORTSMAN MEANS HEALTHY
THEN SPORTSMAN-POWERLIFTER
MUST ALSO BE HEALTHY.
OTHERWISE THERE IS SOMETHING WRONG WITH CLASS HIERARCHY
S.
O.
L
I.
D.
LISKOV SUBSTITUTION PRINCIPLE
Objects in program should be replacable with instances of their subtypes without altering the correctness of the program

SOLID (3): LSP


Liskov Substitution Principle
If it looks like a DUCK, quacks like a DUCK, *but* needs BATTERIES
- You probably need a better Abstraction -

SOLID (3): LSP

```
interface IRectangle {
    int Width { get; }
    int Height { get; }
    int Area => Width * Height;
}

class Square : IRectangle {
    public int Width => _size;
    public int Height => _size;

    public Square(int size);

    private readonly int _size;
}

IRectangle r = new Square(2);
var test = r.Area == 4;
```

SOLID (3): LSP

```
interface IRectangle {
    int Width { get; set; }
    int Height { get; set; }
    int Area => Width * Height;
}

class Square : IRectangle {
    public int Width { get => _size; set => _size = value; }
    public int Height { get => _size; set => _size = value; }

    public Square(int size);

    private int _size;
}

static bool Test(IRectangle r) {
    var w = r.Width;
    r.Height = 3;
    return r.Area == w * 3;
}
```

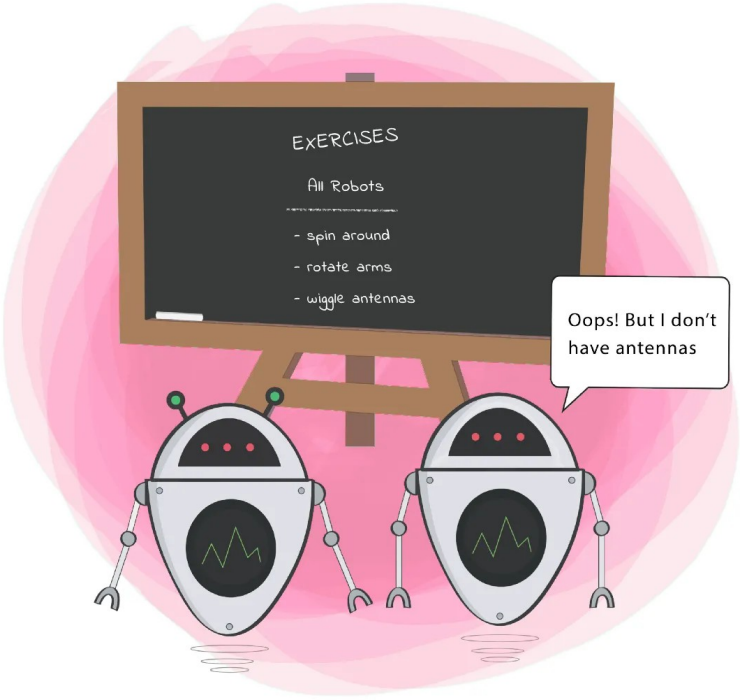
SOLID (3): LSP

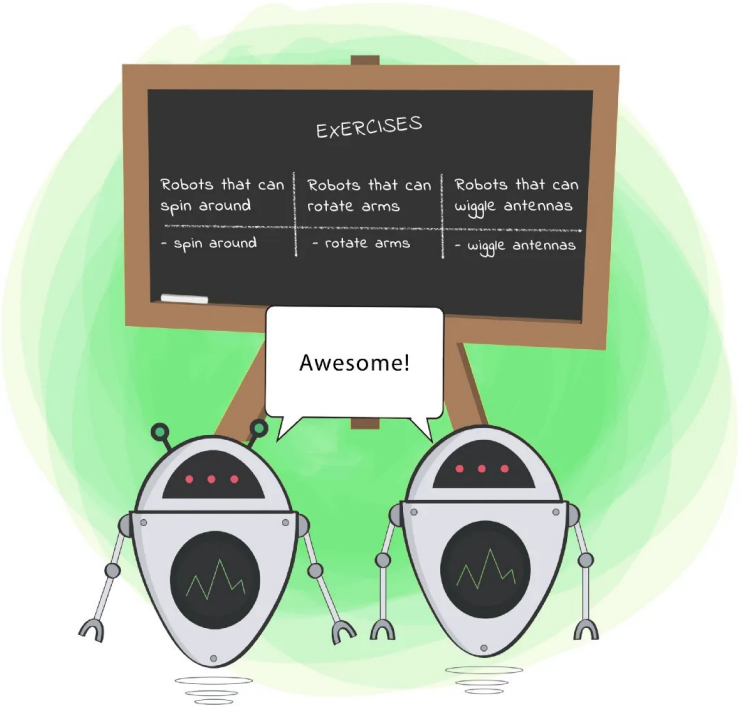
Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом.

Если базовый тип реализует определённое поведение, то это поведение должно быть корректно реализовано и для всех его наследников.

В любое место, где ожидается базовый класс, может быть подсунут любой наследник.

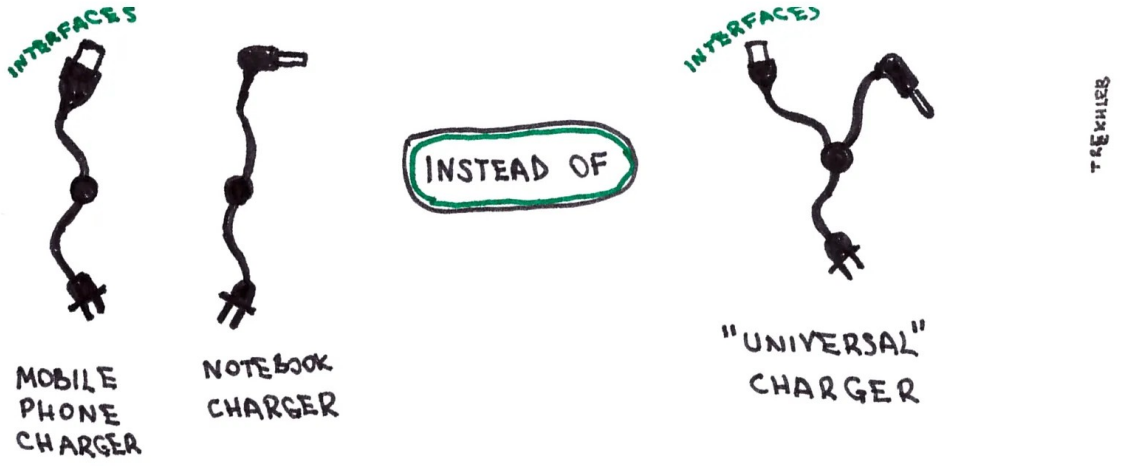
SOLID (4): ISP





✗ Interface Segregation ✓

SOLID (4): ISP


INTERFACES
MOBILE PHONE CHARGER
NOTEBOOK CHARGER
INSTEAD OF
INTERFACES
"UNIVERSAL" CHARGER
S.
O.
L.
I
D.
INTERFACE SEGREGATION PRINCIPLE
Many client-specific interfaces are better than one general-purpose interface.
NO CLIENT SHOULD BE FORCED TO DEPEND ON METHODS IT DOESN'T USE

SOLID (4): ISP

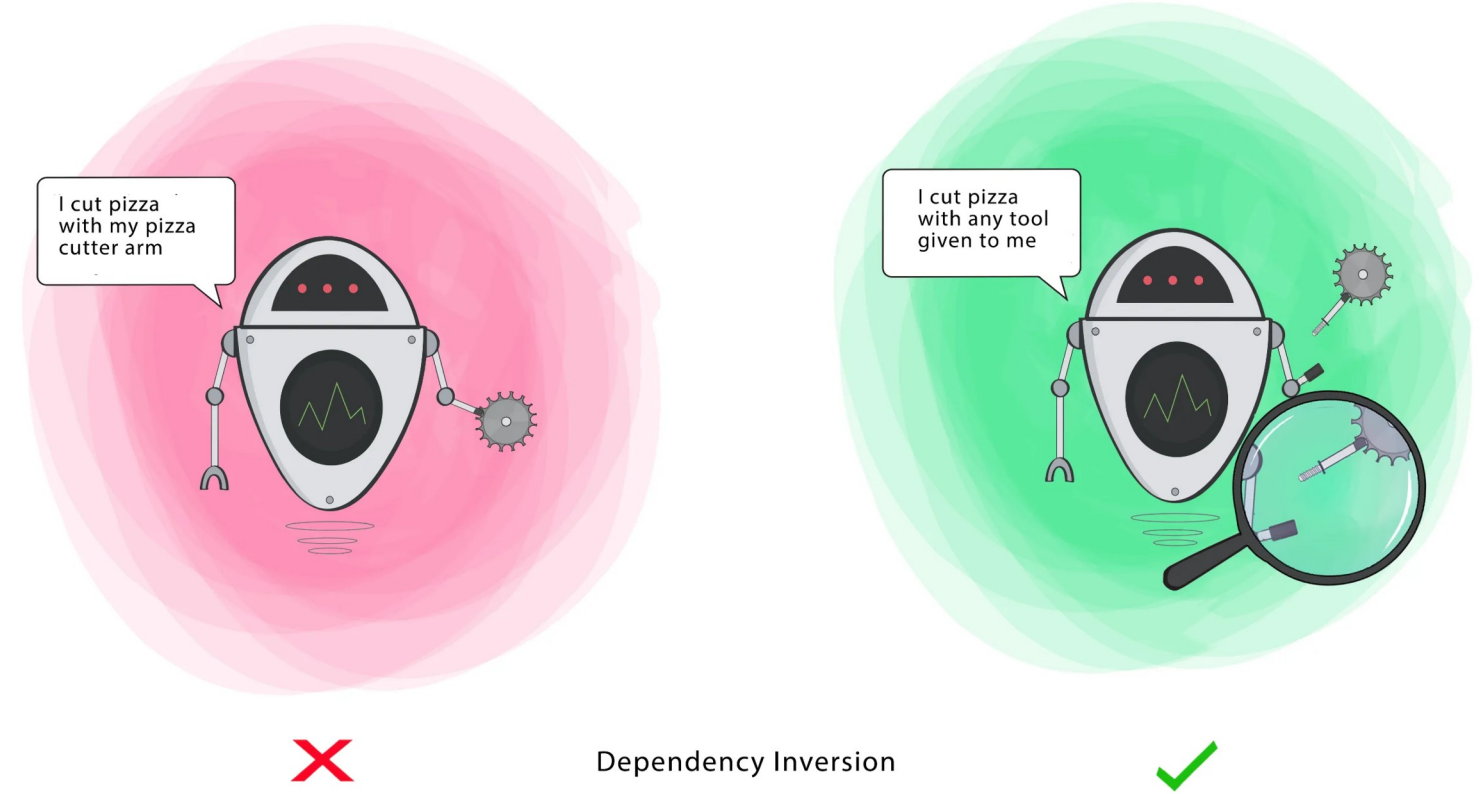
```
interface IShape
{
    float Area { get; }
    float Volume { get; }
}



class Square : IShape
{ … }

class Cuboid : IShape
{ … }
```

```
interface IShape
{
    float Area { get; }
}

interface IShape3D
{
    float Volume { get; }
}

class Square : IShape
{ … }

class Cuboid : IShape, IShape3D
{ … }
```

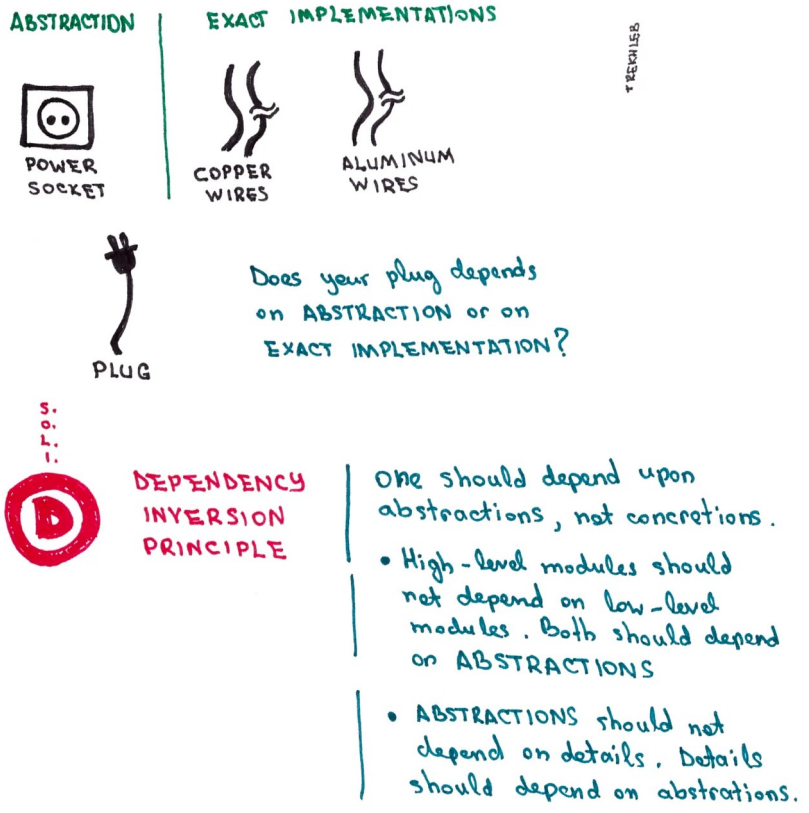
SOLID (4): ISP

Клиенты (классы) не должны зависеть от методов, которые они не используют.

SOLID (5): DIP


I cut pizza with my pizza cutter arm
I cut pizza with any tool given to me
Dependency Inversion

SOLID (5): DIP


ABSTRACTION
EXACT IMPLEMENTATIONS
POWER SOCKET
COPPER WIRES
ALUMINUM WIRES
PLUG
Does your plug depends on ABSTRACTION or on EXACT IMPLEMENTATION?
S.O.L.I.D
DEPENDENCY INVERSION PRINCIPLE
One should depend upon abstractions, not concretions.
• High-level modules should not depend on low-level modules. Both should depend on ABSTRACTIONS
• ABSTRACTIONS should not depend on details. Details should depend on abstractions.

SOLID (5): DIP

```
class Cash
{
    public void Pay(decimal amount);
}

class Shop
{
    private readonly Cash _cash;

    public void Buy(…, decimal amount)
    {
        _cash.Pay(amount);
    }
}
```

SOLID (5): DIP

```
interface IPay
{
    void Pay(decimal amount);
}

class Cash : IPay
{
    public void Pay(decimal amount);
}

class Shop
{
    private readonly IPay _pay;

    public void Buy(…, decimal amount)
    {
        _pay.Pay(amount);
    }
}
```

SOLID (5): DIP

1. Модули верхних уровней не должны зависеть от модулей нижних уровней.
Оба типа модулей должны зависеть от абстракций.

2. Абстракции не должны зависеть от деталей.
Детали должны зависеть от абстракций.

switch – потенциальный индикатор нарушения принципов DRY и Open-Close

```
enum Mode { A, B, C }

void Process(Mode m)
{
    switch (a)
    {
    case Mode.A:
        /* A */ break;
    case Mode.B:
        /* B */ break;
    case Mode.C:
        /* C */ break;
    default:
        /* D */ break;
    }
}
```

```
interface IAbstraction
{
    void Do();
}

class A : IAbstraction
{
    public void Do() { /* A */ }
}

…

void Process(IAbstraction m)
{
    m.Do();
}
```

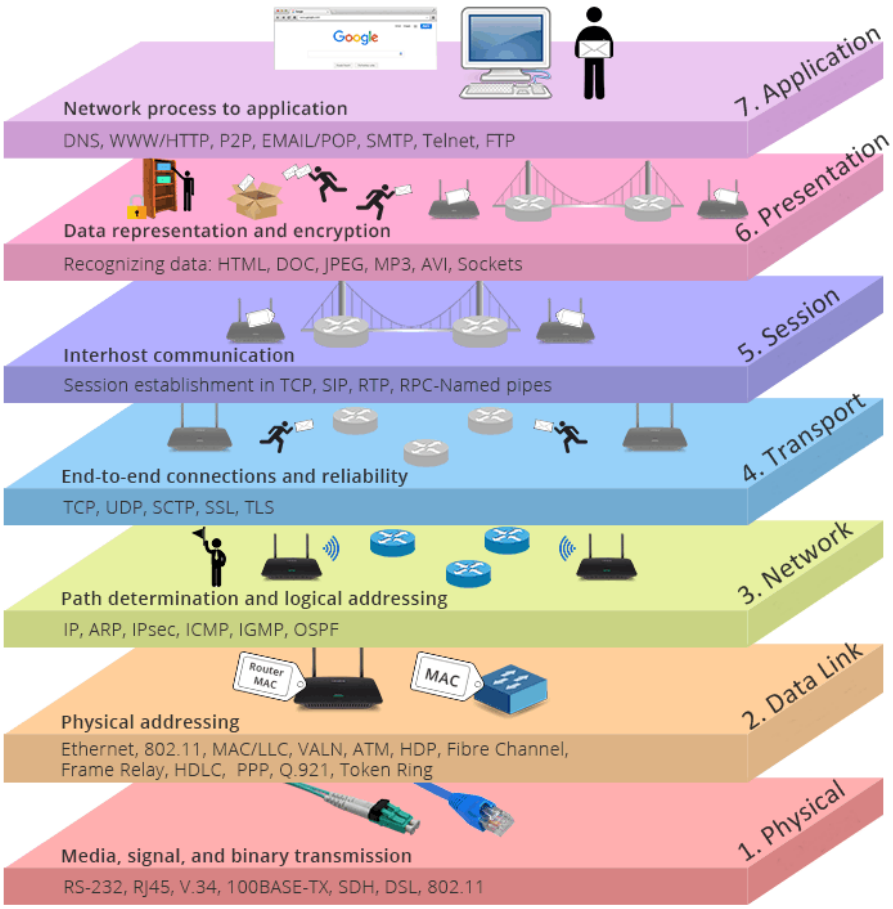
Как вносится новый функционал

Мы вообще живем в мире прототипов,
которые волевым усилием отправили в продакшен.

1. Рефакторинг (при необходимости)
Изменяется архитектура приложения без изменения функциональности.
Модульные тесты позволяют провести fast-check, что ничего не сломалось.

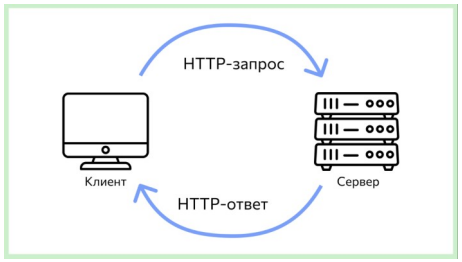
2. Добавление новой функциональности
Добавляется новая функциональность и тесты для неё.

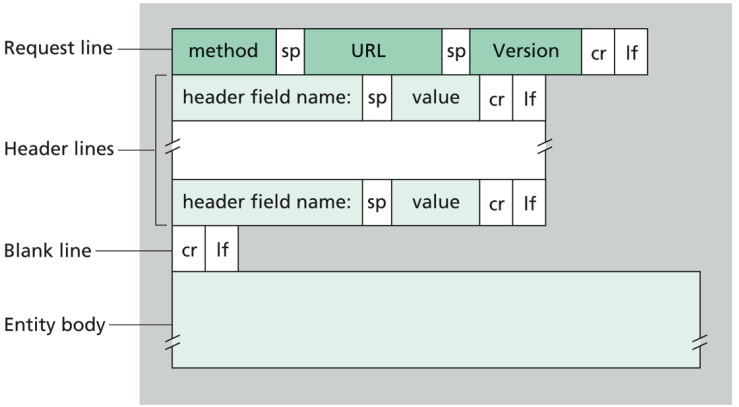
HTTP



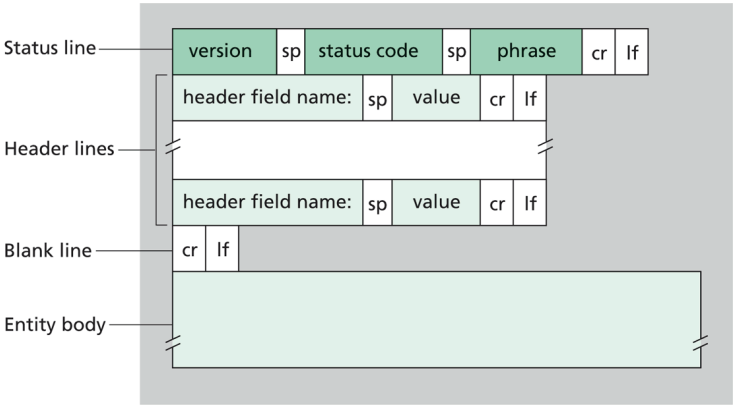
HTTP (HyperText Transfer Protocol) — протокол прикладного уровня передачи данных.

HTTP Messages





REQUEST



RESPONSE

http://www.columbia.edu/~fdc/sample.html

```
GET /~fdc/sample.html HTTP/1.1
Host: www.columbia.edu
User-Agent: Firefox/111.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://www.google.com/
DNT: 1
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Date: Thu, 16 Mar 2023 06:25:27 GMT
Server: Apache
Content-Encoding: gzip
Content-Length: 12038
Content-Type: text/html
Set-Cookie: KEY=VALUE

<!DOCTYPE HTML>
<html lang="en">
...
</html>
```

HTTP Methods

Properties of request methods

Request method	RFC	Request has payload body	Response has payload body	Safe	Idempotent	Cacheable
GET	RFC 9110	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 9110	Optional	No	Yes	Yes	Yes
POST	RFC 9110	Yes	Yes	No	No	Yes
PUT	RFC 9110	Yes	Yes	No	Yes	No
DELETE	RFC 9110	Optional	Yes	No	Yes	No
CONNECT	RFC 9110	Optional	Yes	No	No	No
OPTIONS	RFC 9110	Optional	Yes	Yes	Yes	No
TRACE	RFC 9110	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

/books

GET	/books	Lists all the books in the database
DELETE	/books/{bookId}	Deletes a book based on their id
POST	/books	Creates a Book
PUT	/books/{bookId}	Method to update a book
GET	/books/{bookId}	Retrieves a book based on their id

HTTP Status Codes