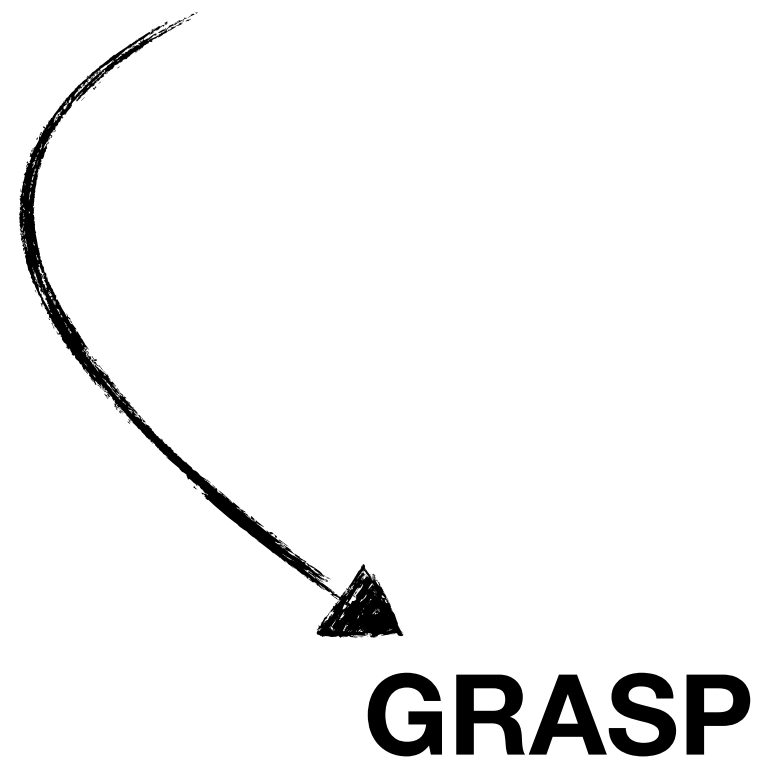


# объектно-ориентированное программирование

GRASP

general responsibility assignment software principles  
общие принципы распределения ответственностей в ПО



information expert

# information expert

## пример несоблюдения

```
public record OrderItem(
    int Id,
    decimal Price,
    int Quantity);

public class Order
{
    private readonly List<OrderItem> _items;

    public Order()
    {
        _items = new List<OrderItem>();
    }

    public IReadOnlyCollection<OrderItem> Items => _items;
}
```

```
public record Receipt(
    decimal TotalCost,
    DateTime Timestamp);

public class ReceiptService
{
    public Receipt CalculateReceipt(Order customer)
    {
        var totalCost = customer.Items
            .Sum(order => order.Price * order.Quantity);

        var timestamp = DateTime.Now;

        return new Receipt(totalCost, timestamp);
    }
}
```

# information expert

## почему нарушать плохо

- нарушение SRP
- проблемы с переиспользованием  
либо копипаста, либо нелогичная связь между модулями
- усложнённое тестирование

# information expert

## пример соблюдения

```
public record OrderItem(
    int Id,
    decimal Price,
    int Quantity)
{
    public decimal Cost ⇒ Price * Quantity;
}

public class Order
{
    private readonly List<OrderItem> _items;

    public Order()
    {
        _items = new List<OrderItem>();
    }

    public IReadOnlyCollection<OrderItem> Items ⇒ _items;

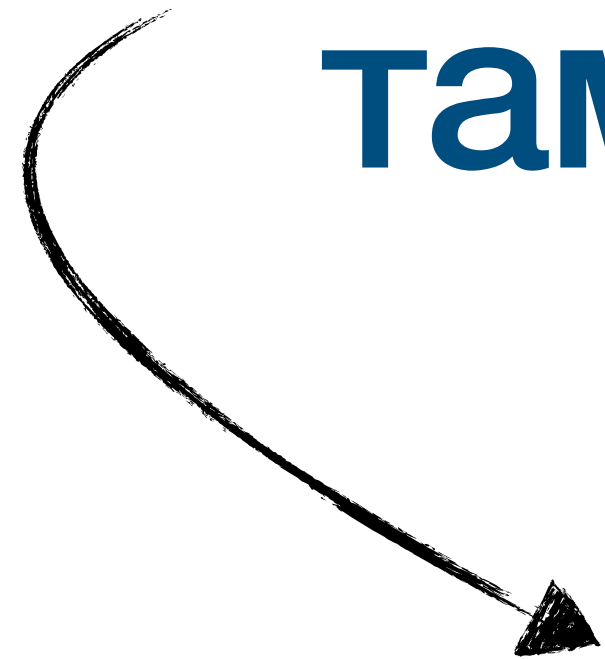
    public decimal TotalCost ⇒ _items.Sum(x ⇒ x.Cost);
}
```

```
public record Receipt(
    decimal TotalCost,
    DateTime Timestamp);

public class ReceiptService
{
    public Receipt CalculateReceipt(Order customer)
    {
        var totalCost = customer.TotalCost;
        var timestamp = DateTime.Now;

        return new Receipt(totalCost, timestamp);
    }
}
```

**информация должна обрабатываться  
там, где она содержится**



**information expert**

creator



# creator

## пример несоблюдения

```
public class Order
{
    private readonly List<OrderItem> _items;

    ...

    public Order AddItem(OrderItem item)
    {
        _items.Add(item);
        return this;
    }
}
```

```
public class OrderService
{
    public Order CreateDefaultOrder()
    {
        var order = new Order()
            .AddItem(new OrderItem(1, 100, 1))
            .AddItem(new OrderItem(2, 1000, 3));

        return order;
    }
}
```

# creator

## пример соблюдения

```
public class Order
{
    private readonly List<OrderItem> _items;

    public Order AddItem(
        int id,
        decimal price,
        int quantity)
    {
        _items.Add(new OrderItem(id, price, quantity));
        return this;
    }
}
```

```
public class OrderService
{
    public Order CreateDefaultOrder()
    {
        var order = new Order()
            .AddItem(1, 100, 1)
            .AddItem(2, 1000, 3);

        return order;
    }
}
```

**ответственность за создание используемых  
объектов должна лежать на типах, их  
использующих**



**creator**

# creator

## подводные камни

- добавляется неявная связанность между конструктором модели и методом создателя
- необходимость обладать всеми данными для создания может привести к нарушению SRP создателем
- пересборка объектов может ухудшить производительность

```
public class OrderService
{
    public Order CreateDefaultOrder(
        IEnumerable<OrderItem> items)
    {
        var order = new Order()
            .AddItem(1, 100, 1)
            .AddItem(2, 1000, 3);

        foreach (var item in items)
        {
            order.AddItem(
                item.Id,
                item.Price,
                item.Quantity);
        }

        return order;
    }
}
```

**controller**

# controller

## use-case controller

```
public class UserController
{
    private readonly IUserService _userService;

    // ...

    public void AddUser(User user)
    {
        _userService.AddUser(user);
    }
}
```

# controller

## use-case group controller

```
public class UserController
{
    private readonly IUserService _userService;

    // ...

    public void AddUser(User user)
    {
        _userService.AddUser(user);
    }

    public void DeleteUser(UserId id)
    {
        _userService.DeleteUser(id);
    }
}
```

# controller

## facade controller


```
public class FacadeController
{
    private readonly IUserService _userService;
    private readonly IMailingService _mailingService;

    // ...
}
```



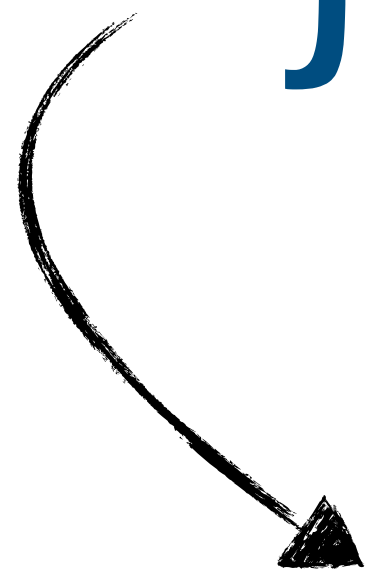
**low coupling & high cohesion**

**мера зависимости модулей друг  
друга**



**coupling**

**мера логической соотнесённости  
логики в рамках модуля**



**cohesion**

# low cohesion

^ плохо

```
public class DataProvider
{
    private readonly HttpClient _httpClient;

    public DataProvider(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public (double Value, DateTime Timestamp) GetTemperatureData()
    {
        var response = _httpClient.GetAsync("temperature-sensor-uri").Result;
        var value = response.Content.ReadFromJsonAsync<double>().Result;

        return (value, DateTime.Now);
    }

    public (double Value, DateTime Timestamp) GetUsedMemoryData()
    {
        return (GC.GetTotalMemory(false), DateTime.Now);
    }
}
```

# high cohesion

^ хорошо

```
public class TemperatureDataProvider
{
    private readonly HttpClient _httpClient;

    public TemperatureDataProvider(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public (double Value, DateTime Timestamp) GetTemperatureData()
    {
        var response = _httpClient.GetAsync("temperature-sensor-uri").Result;
        var value = response.Content.ReadFromJsonAsync<double>().Result;

        return (value, DateTime.Now);
    }
}

public class UsedMemoryDataProvider
{
    public (double Value, DateTime Timestamp) GetUsedMemoryData()
    {
        return (GC.GetTotalMemory(false), DateTime.Now);
    }
}
```

# high coupling

^ плохо

```
public class DataMonitor
{
    private readonly TemperatureDataProvider _temperatureProvider;
    private readonly UsedMemoryDataProvider _usedMemoryProvider;

    ...

    public void Monitor(MetricType type, CancellationToken cancellationToken)
    {
        while (cancellationToken.IsCancellationRequested is false)
        {
            var (value, timestamp) = type switch
            {
                MetricType.Temperature => _temperatureProvider.GetTemperatureData(),
                MetricType.UsedMemory => _usedMemoryProvider.GetUsedMemoryData(),
            };

            Console.WriteLine($"{type} = {value}, at {timestamp}");
        }
    }
}
```

# low coupling

^ хорошо

```
public interface IChronologicalDataProvider
{
    string Kind { get; }

    (double Value, DateTime Timestamp) GetData();
}

public class DataMonitor
{
    private readonly IChronologicalDataProvider _provider;

    public DataMonitor(IChronologicalDataProvider provider)
    {
        _provider = provider;
    }

    public void Monitor(CancellationToken cancellationToken)
    {
        while (cancellationToken.IsCancellationRequested is false)
        {
            var (value, timestamp) = _provider.GetData();
            Console.WriteLine($"{_provider.Kind} = {value}, at {timestamp}");
        }
    }
}
```

**indirection**



любое взаимодействие с данными, поведением,  
модулями, реализованное не напрямую, а через  
какой-либо агрегирующий их объект



**object indirection**

любое взаимодействие с данными, поведением,  
модулями, реализованное не напрямую, а через  
какой-либо интерфейс



**interface segregation**

polymorphism

# ЛОЛ

**используйте полиморфизм**

protected variations

# protected variations

- коррелирует с OCP из SOLID  
делает большой акцент на определение точек нестабильности
- подразумевает выделение стабильного интерфейса над нестабильной реализацией

pure fabrication

# pure fabrication

- подразумевает наличие в системе искусственной, выдуманной сущности, не отражающей конкретный объект моделируемых бизнес процессов
- обычно это инфраструктуры модули, сервисы
- не рекомендуется вносить такие типы в доменную модель