GRASP

ПРИНЦИПИ

Виконав:
Кравець Назар - ПМОм-11
2025

Зміст:

Що таке GRASP

GRASP (General Responsibility Assignment Software Patterns) – це набір шаблонів проєктування, що допомагають у розподілі відповідальності між об'єктами в об'єктно-орієнтованому програмуванні.

Принципи GRASP та їх приклади

Information Expert

відповідальність надається класу, який володіє необхідними даними

```
public class Customer
  {
     private readonly List<Order> _orders = new List<Order>();

     public decimal GetTotalAmount(Guid orderId)
     {
        return this._orders.Sum(x => x.Amount);
     }
  }
```

Creator

клас, який створює інші об'єкти

```
public class Customer
  {
     private readonly List<Order> _orders = new List<Order>();

     public void AddOrder(List<OrderProduct> orderProducts)
     {
        var order = new Order(orderProducts); // creator
        _orders.Add(order);
     }
  }
```

Принципи GRASP та їх приклади

Controller

клас, що обробляє запити від користувача та керує потоком подій у системі

```
public class OrderController
{
    private readonly OrderService orderService;

    public void CreateOrder(User user, List<Product> products)
    {
        orderService.ProcessOrder(user, products);
    }
}
```

Принципи GRASP та їх приклади

Low Coupling (Низьке зчеплення) – принцип, що мінімізує залежність між класами, полегшуючи змінність і підтримку.

High Cohesion (Висока зв'язність) – забезпечує, щоб клас мав чітко визначений набір відповідальностей, що сприяє його зручному використанню та модифікації.

Low Coupling та High Cohesion

```
public interface IPaymentProcessor
{
    void ProcessPayment(Order order);
}

public class PaymentProcessor : IPaymentProcessor
{
    public void ProcessPayment(Order order)
    {
        // Логіка обробки платежу
    }
}

public class OrderService
{
    private readonly IPaymentProcessor _paymentProcessor;

    public OrderService(IPaymentProcessor paymentProcessor)
    {
        _paymentProcessor = paymentProcessor;
    }

    public void CompleteOrder(Order order)
    {
        _paymentProcessor.ProcessPayment(order);
        // Інші дії, наприклад оновлення статусу замовлення
    }
}
```

Принципи GRASP та їх приклади

Pure Fabrication

створення окремих класів для підтримки низького зчеплення та високої зв'язності

```
public class Logger
{
    public void Log(string message)
    {
        Console.WriteLine($"[LOG]: {message}");
    }
}

public class OrderService
{
    private readonly Logger _logger;

    public OrderService(Logger logger)
    {
        _logger = logger;
    }

    public void CompleteOrder(Order order)
    {
        _logger.Log("Order completed");
        // Додаткова логіка обробки замовлення
    }
}
```

Принципи GRASP та їх приклади

Polymorphism

використання загальних інтерфейсів або абстрактних класів для роботи з різними реалізаціями.

```
public interface IShape
{
    double GetArea();
}

public class Circle : IShape
{
    public double Radius { get; set; }

    public double GetArea() => Math.PI * Radius * Radius;
}

public class Rectangle : IShape
{
    public double Width { get; set; }
    public double Height { get; set; }

    public double GetArea() => Width * Height;
}
```

Принципи GRASP та їх приклади

Indirection

введення проміжного об'єкта для зменшення прямої залежності між класами

```
public class Database
{
    public void Save(string data)
    {
        Console.WriteLine("Data saved: " + data);
    }
}

public class DataManager
{
    private readonly Database _database;

    public DataManager(Database database)
    {
        _database = database;
    }

    public void SaveData(string data)
    {
        _database.Save(data);
    }
}
```

Принципи GRASP та їх приклади

Protected Variations

використання абстракцій та патернів, що запобігають впливу змін в одному компоненті на інші

```
public interface IPaymentMethod
{
    void ProcessPayment(decimal amount);
}

public class CreditCardPayment : IPaymentMethod
{
    public void ProcessPayment(decimal amount)
    {
        Console.WriteLine($"Processing credit card payment of {amount}");
    }
}

public class PayPalPayment : IPaymentMethod
{
    public void ProcessPayment(decimal amount)
    {
        Console.WriteLine($"Processing PayPal payment of {amount}");
    }
}

public class PaymentProcessor
{
    private readonly IPaymentMethod _paymentMethod;

    public PaymentProcessor(IPaymentMethod paymentMethod)
    {
        _paymentMethod = paymentMethod;
    }

    public void ProcessPayment(decimal amount)
    {
        _paymentMethod.ProcessPayment(amount);
    }
}
```

Q&A