Полиморфизъм — 12б, ООП

1. Увод

Полиморфизмът е принцип на ООП, който позволява обекти от различни типове да се третират като обекти от общ базов тип, като всеки обект изпълнява същите операции по различен начин.

2. Какво е полиморфизъм?

Полиморфизмът означава:

- **Един интерфейс** - различни имплементации
- **Позднно свързване** - решението се взема по време на изпълнение
- **Гъвкавост** - лесно добавяне на нови типове
- **Преизползване** - един код работи с различни обекти
- **Разширяемост** - лесно добавяне на нова функционалност

3. Видове полиморфизъм

Compile-time полиморфизъм

- Методно претоварване
- Операторно претоварване
- Шаблонни функции

- Решението се взема по време на компилация

Runtime полиморфизъм

- Virtual методи
- Override методи
- Интерфейси
- Решението се взема по време на изпълнение

4. Compile-time полиморфизъм

Методно претоварване (Method Overloading):

- **Също име** - различни параметри
- **Различни типове** - int, string, double
- **Различен брой параметри** - 1, 2, 3 параметъра
- **Различни модификатори** - ref, out, params

Пример с методно претоварване:

```
class Calculator {
    // Претоварване с различни типове
    public int Add(int a, int b) {
        Console.WriteLine("Извиква се Add(int, int)");
        return a + b;
    }

    public double Add(double a, double b) {
        Console.WriteLine("Извиква се Add(double, double)");
        return a + b;
    }
```

```
    public string Add(string a, string b) {
        Console.WriteLine("Извиква се Add(string, string)");
        return a + b;
    }

    // Претоварване с различен брой параметри
    public int Add(int a, int b, int c) {
        Console.WriteLine("Извиква се Add(int, int, int)");
        return a + b + c;
    }

    public int Add(params int[] numbers) {
        Console.WriteLine("Извиква се Add(params int[])");
        int sum = 0;
        foreach (int num in numbers) {
            sum += num;
        }
        return sum;
    }

    // Претоварване с различни модификатори
    public void ProcessValue(int value) {
        Console.WriteLine($"Обработва се стойност: {value}");
    }

    public void ProcessValue(ref int value) {
        value *= 2;
        Console.WriteLine($"Обработена стойност (ref): {value}");
    }

    public void ProcessValue(out int value) {
        value = 42;
        Console.WriteLine($"Създадена стойност (out): {value}");
    }
}

// Използване
Calculator calc = new Calculator();

// Различни типове
int result1 = calc.Add(5, 3);
double result2 = calc.Add(3.14, 2.71);
string result3 = calc.Add("Здравей", " Свят");

// Различен брой параметри
int result4 = calc.Add(1, 2, 3);
int result5 = calc.Add(1, 2, 3, 4, 5);
```

```
// Различни модификатори
int value = 10;
calc.ProcessValue(value);
calc.ProcessValue(ref value);
calc.ProcessValue(out int newValue);
```

5. Runtime полиморфизъм

Virtual и Override методи:

- **Virtual** - метод в базовия клас, който може да се пренапише
- **Override** - пренаписване на виртуален метод
- **Позднно свързване** - решението се взема по време на изпълнение
- **Полиморфизъм** - различни обекти, различни имплементации

Пример с virtual и override:

```
// Базов клас
class Animal {
    protected string name;
    protected int age;

    public Animal(string name, int age) {
        this.name = name;
        this.age = age;
    }

    // Виртуален метод - може да се пренапише
    public virtual void MakeSound() {
        Console.WriteLine($"{name} издава звук...");
    }

    public virtual void Move() {
        Console.WriteLine($"{name} се движи...");
    }
```

```
    // Виртуален метод с връщане на стойност
    public virtual string GetInfo() {
        return $"Животно: {name}, {age} години";
    }

    // Обикновен метод - не може да се пренапише
    public void Sleep() {
        Console.WriteLine($"{name} спи...");
    }
}

// Производен клас
class Dog : Animal {
    private string breed;

    public Dog(string name, int age, string breed) : base(name, age) {
        this.breed = breed;
    }

    // Override на виртуалния метод
    public override void MakeSound() {
        Console.WriteLine($"{name} лае: Woof! Woof!");
    }

    public override void Move() {
        Console.WriteLine($"{name} тича на четири крака");
    }

    public override string GetInfo() {
        return base.GetInfo() + $", порода: {breed}";
    }

    // Нов метод специфичен за кучето
    public void Fetch() {
        Console.WriteLine($"{name} донесе топката!");
    }
}

// Друг производен клас
class Cat : Animal {
    private bool isIndoor;

    public Cat(string name, int age, bool isIndoor) : base(name, age) {
        this.isIndoor = isIndoor;
    }
```

```
    public override void MakeSound() {
        Console.WriteLine($"{name} мяука: Meow! Meow!");
    }

    public override void Move() {
        Console.WriteLine($"{name} се движи тихо и грациозно");
    }

    public override string GetInfo() {
        return base.GetInfo() + $", домашен: {(isIndoor ? "Да" : "Не")}";
    }

    // Нов метод специфичен за котката
    public void Climb() {
        Console.WriteLine($"{name} се качва на дърво");
    }
}

// Използване на полиморфизъм
Animal[] animals = {
    new Dog("Рекс", 3, "Лабрадор"),
    new Cat("Мурка", 2, true),
    new Dog("Бади", 1, "Джак Ръсел"),
    new Cat("Тигър", 4, false)
};

Console.WriteLine("=== Полиморфизъм в действие ===");
foreach (Animal animal in animals) {
    Console.WriteLine($"\n{animal.GetInfo()}");
    animal.MakeSound();
    animal.Move();

    // Проверка за конкретен тип
    if (animal is Dog dog) {
        dog.Fetch();
    } else if (animal is Cat cat) {
        cat.Climb();
    }
}
```

6. Полиморфизъм с интерфейси

Предимства на полиморфизма с интерфейси:

- **Множествено наследяване** - клас може да имплементира множество интерфейси
- **Гъвкавост** - лесно добавяне на нови способности
- **Стандартизация** - дефиниране на общи контракти
- **Тестване** - лесно създаване на mock обекти

Пример с полиморфизъм и интерфейси:

```
// Интерфейс за летене
interface IFlyable {
    void Fly();
    double GetMaxAltitude();
}

// Интерфейс за плуване
interface ISwimmable {
    void Swim();
    double GetMaxDepth();
}

// Интерфейс за ходене
interface IWalkable {
    void Walk();
    double GetMaxSpeed();
}

// Базов клас за животни
class Animal {
    protected string name;
    protected int age;

    public Animal(string name, int age) {
        this.name = name;
        this.age = age;
    }

    public virtual void DisplayInfo() {
```

```
        Console.WriteLine($"Животно: {name}, {age} години");
    }
}

// Клас за птици
class Bird : Animal, IFlyable, IWalkable {
    private double maxAltitude;
    private double maxSpeed;

    public Bird(string name, int age, double maxAltitude, double maxSpeed)
        : base(name, age) {
        this.maxAltitude = maxAltitude;
        this.maxSpeed = maxSpeed;
    }

    public void Fly() {
        Console.WriteLine($"{name} лети в небето");
    }

    public double GetMaxAltitude() {
        return maxAltitude;
    }

    public void Walk() {
        Console.WriteLine($"{name} ходи по земята");
    }

    public double GetMaxSpeed() {
        return maxSpeed;
    }

    public override void DisplayInfo() {
        base.DisplayInfo();
        Console.WriteLine($"Максимална височина: {maxAltitude}м");
        Console.WriteLine($"Максимална скорост: {maxSpeed} км/ч");
    }
}

// Клас за риби
class Fish : Animal, ISwimmable {
    private double maxDepth;

    public Fish(string name, int age, double maxDepth) : base(name, age) {
        this.maxDepth = maxDepth;
    }

    public void Swim() {
```

```
        Console.WriteLine($"{name} плува във водата");
    }

    public double GetMaxDepth() {
        return maxDepth;
    }

    public override void DisplayInfo() {
        base.DisplayInfo();
        Console.WriteLine($"Максимална дълбочина: {maxDepth}м");
    }
}

// Клас за водоплаващи птици
class Waterfowl : Animal, IFlyable, ISwimmable, IWalkable {
    private double maxAltitude;
    private double maxDepth;
    private double maxSpeed;

    public Waterfowl(string name, int age, double maxAltitude, double
maxDepth, double maxSpeed)
        : base(name, age) {
        this.maxAltitude = maxAltitude;
        this.maxDepth = maxDepth;
        this.maxSpeed = maxSpeed;
    }

    public void Fly() {
        Console.WriteLine($"{name} лети над водата");
    }

    public double GetMaxAltitude() {
        return maxAltitude;
    }

    public void Swim() {
        Console.WriteLine($"{name} плува в езерото");
    }

    public double GetMaxDepth() {
        return maxDepth;
    }

    public void Walk() {
        Console.WriteLine($"{name} ходи по брега");
    }
```

```
    public double GetMaxSpeed() {
        return maxSpeed;
    }

    public override void DisplayInfo() {
        base.DisplayInfo();
        Console.WriteLine($"Максимална височина: {maxAltitude}м");
        Console.WriteLine($"Максимална дълбочина: {maxDepth}м");
        Console.WriteLine($"Максимална скорост: {maxSpeed} км/ч");
    }
}

// Използване на полиморфизъм с интерфейси
Animal[] animals = {
    new Bird("Орел", 5, 3000, 80),
    new Fish("Златна рибка", 2, 10),
    new Waterfowl("Патък", 3, 1000, 5, 15)
};

Console.WriteLine("=== Полиморфизъм с интерфейси ===");
foreach (Animal animal in animals) {
    Console.WriteLine($"\n{animal.GetType().Name}:");
    animal.DisplayInfo();

    // Полиморфизъм с интерфейси
    if (animal is IFlyable flyable) {
        flyable.Fly();
        Console.WriteLine($"Максимална височина: {flyable.GetMaxAltitude()}
м");
    }

    if (animal is ISwimmable swimmable) {
        swimmable.Swim();
        Console.WriteLine($"Максимална дълбочина: {swimmable.GetMaxDepth()}
м");
    }

    if (animal is IWalkable walkable) {
        walkable.Walk();
        Console.WriteLine($"Максимална скорост: {walkable.GetMaxSpeed()}
км/ч");
    }
}
```

7. Пълен пример - Система за плащания

Реален пример с полиморфизъм:

```
// Интерфейс за плащания
interface IPayment {
    decimal Amount { get; set; }
    string Currency { get; set; }
    bool ProcessPayment();
    string GetPaymentDetails();
}

// Интерфейс за валидация
interface IValidatable {
    bool IsValid { get; }
    List ValidationErrors { get; }
    bool Validate();
}

// Интерфейс за нотификации
interface INotifiable {
    event EventHandler PaymentProcessed;
    void SendNotification(string message);
}

// Базов клас за плащания
public abstract class PaymentBase : IPayment, IValidatable, INotifiable {
    public decimal Amount { get; set; }
    public string Currency { get; set; }
    public bool IsValid { get; protected set; }
    public List ValidationErrors { get; protected set; }

    public event EventHandler PaymentProcessed;

    protected PaymentBase(decimal amount, string currency) {
        Amount = amount;
        Currency = currency;
        ValidationErrors = new List();
        IsValid = false;
    }

    // Абстрактен метод - трябва да се имплементира
    public abstract bool ProcessPayment();
```

```
    // Виртуален метод - може да се override
    public virtual string GetPaymentDetails() {
        return $"Плащане: {Amount:C} {Currency}";
    }

    // Виртуален метод за валидация
    public virtual bool Validate() {
        ValidationErrors.Clear();

        if (Amount <= 0) {
            ValidationErrors.Add("Сумата трябва да е положителна");
        }

        if (string.IsNullOrEmpty(Currency)) {
            ValidationErrors.Add("Валутата не може да бъде празна");
        }

        IsValid = ValidationErrors.Count == 0;
        return IsValid;
    }

    // Виртуален метод за нотификации
    public virtual void SendNotification(string message) {
        Console.WriteLine($"Изпратено уведомление: {message}");
        PaymentProcessed?.Invoke(this, message);
    }

    // Защитен метод за обща логика
    protected virtual void OnPaymentProcessed() {
        Console.WriteLine("Плащането е обработено успешно");
        SendNotification($"Плащането от {Amount:C} е завършено");
    }
}

// Конкретна имплементация за кредитна карта
public class CreditCardPayment : PaymentBase {
    public string CardNumber { get; set; }
    public string CardHolderName { get; set; }
    public DateTime ExpiryDate { get; set; }
    public string CVV { get; set; }

    public CreditCardPayment(decimal amount, string currency, string
cardNumber,
                             string cardHolderName, DateTime expiryDate,
string cvv)
                             : base(amount, currency) {
        CardNumber = cardNumber;
```

```
        CardHolderName = cardHolderName;
        ExpiryDate = expiryDate;
        CVV = cvv;
    }

    public override bool ProcessPayment() {
        if (!Validate()) {
            Console.WriteLine("Плащането не е валидно");
            return false;
        }

        Console.WriteLine("Обработва се плащане с кредитна карта...");

        if (SimulatePaymentProcessing()) {
            OnPaymentProcessed();
            return true;
        }

        Console.WriteLine("Плащането е неуспешно");
        return false;
    }

    public override string GetPaymentDetails() {
        return base.GetPaymentDetails() +
               $"\nКарта: {MaskCardNumber(CardNumber)}" +
               $"\nПритежател: {CardHolderName}" +
               $"\nИзтича: {ExpiryDate:MM/yyyy}";
    }

    public override bool Validate() {
        bool baseValid = base.Validate();

        if (string.IsNullOrEmpty(CardNumber) || CardNumber.Length != 16) {
            ValidationErrors.Add("Невалиден номер на карта");
        }

        if (string.IsNullOrEmpty(CardHolderName)) {
            ValidationErrors.Add("Името на притежателя не може да бъде
празно");
        }

        if (ExpiryDate < DateTime.Now) {
            ValidationErrors.Add("Картата е изтекла");
        }

        if (string.IsNullOrEmpty(CVV) || CVV.Length != 3) {
            ValidationErrors.Add("Невалиден CVV код");
```

```
        }

        IsValid = baseValid && ValidationErrors.Count == 0;
        return IsValid;
    }

    private bool SimulatePaymentProcessing() {
        return new Random().Next(1, 10) > 2; // 80% успех
    }

    private string MaskCardNumber(string cardNumber) {
        if (cardNumber.Length < 8) return "****";
        return cardNumber.Substring(0, 4) + " **** **** " +
cardNumber.Substring(cardNumber.Length - 4);
    }
}

// Конкретна имплементация за банков превод
public class BankTransferPayment : PaymentBase {
    public string BankName { get; set; }
    public string AccountNumber { get; set; }
    public string RoutingNumber { get; set; }

    public BankTransferPayment(decimal amount, string currency, string
bankName,
                           string accountNumber, string routingNumber)
                           : base(amount, currency) {
        BankName = bankName;
        AccountNumber = accountNumber;
        RoutingNumber = routingNumber;
    }

    public override bool ProcessPayment() {
        if (!Validate()) {
            Console.WriteLine("Плащането не е валидно");
            return false;
        }

        Console.WriteLine("Обработва се банков превод...");

        if (SimulateBankTransfer()) {
            OnPaymentProcessed();
            return true;
        }

        Console.WriteLine("Банковият превод е неуспешен");
        return false;
```

```
    }

    public override string GetPaymentDetails() {
        return base.GetPaymentDetails() +
               $"\nБанка: {BankName}" +
               $"\nСметка: {MaskAccountNumber(AccountNumber)}" +
               $"\nРутинг: {RoutingNumber}";
    }

    public override bool Validate() {
        bool baseValid = base.Validate();

        if (string.IsNullOrEmpty(BankName)) {
            ValidationErrors.Add("Името на банката не може да бъде
празно");
        }

        if (string.IsNullOrEmpty(AccountNumber) || AccountNumber.Length <
8) {
            ValidationErrors.Add("Невалиден номер на сметка");
        }

        if (string.IsNullOrEmpty(RoutingNumber) || RoutingNumber.Length !=
9) {
            ValidationErrors.Add("Невалиден рутинг номер");
        }

        IsValid = baseValid && ValidationErrors.Count == 0;
        return IsValid;
    }

    private bool SimulateBankTransfer() {
        return new Random().Next(1, 10) > 1; // 90% успех
    }

    private string MaskAccountNumber(string accountNumber) {
        if (accountNumber.Length < 4) return "****";
        return "****" + accountNumber.Substring(accountNumber.Length - 4);
    }
}

// Конкретна имплементация за кеш
public class CashPayment : PaymentBase {
    public decimal AmountReceived { get; set; }
    public decimal Change { get; private set; }

    public CashPayment(decimal amount, string currency, decimal
```

```
amountReceived)
        : base(amount, currency) {
        AmountReceived = amountReceived;
        Change = 0;
    }

    public override bool ProcessPayment() {
        if (!Validate()) {
            Console.WriteLine("Плащането не е валидно");
            return false;
        }

        Console.WriteLine("Обработва се плащане в брой...");

        if (AmountReceived >= Amount) {
            Change = AmountReceived - Amount;
            OnPaymentProcessed();
            return true;
        } else {
            Console.WriteLine("Недостатъчна сума");
            return false;
        }
    }

    public override string GetPaymentDetails() {
        return base.GetPaymentDetails() +
               $"\nПолучено: {AmountReceived:C}" +
               $"\nРесто: {Change:C}";
    }

    public override bool Validate() {
        bool baseValid = base.Validate();

        if (AmountReceived < 0) {
            ValidationErrors.Add("Получената сума не може да бъде
отрицателна");
        }

        IsValid = baseValid && ValidationErrors.Count == 0;
        return IsValid;
    }
}

// Клас за обработка на плащания
public class PaymentProcessor {
    public void ProcessPayments(List payments) {
        Console.WriteLine("=== Обработка на плащания ===");
```

```
        foreach (var payment in payments) {
            Console.WriteLine($"\n{payment.GetType().Name}:");
            Console.WriteLine(payment.GetPaymentDetails());

            if (payment.ProcessPayment()) {
                Console.WriteLine("✓ Плащането е успешно");
            } else {
                Console.WriteLine("✗ Плащането е неуспешно");

                if (payment is IValidatable validatable) {
                    Console.WriteLine("Грешки при валидация:");
                    foreach (var error in validatable.ValidationErrors) {
                        Console.WriteLine($"  - {error}");
                    }
                }
            }
        }
    }

    public void ProcessPaymentsByType(List payments) {
        Console.WriteLine("\n=== Обработка по тип ===");

        var creditCardPayments = payments.OfType();
        var bankTransferPayments = payments.OfType();
        var cashPayments = payments.OfType();

        Console.WriteLine($"Кредитни карти: {creditCardPayments.Count()}");
        Console.WriteLine($"Банкови преводи:
{bankTransferPayments.Count()}");
        Console.WriteLine($"Плащания в брой: {cashPayments.Count()}");
    }
}

// Използване
var payments = new List {
    new CreditCardPayment(1000, "BGN", "1234567890123456",
                          "Иван Петров", DateTime.Now.AddYears(2), "123"),
    new BankTransferPayment(500, "BGN", "Добротворна банка",
                            "1234567890", "123456789"),
    new CashPayment(200, "BGN", 250),
    new CreditCardPayment(1500, "BGN", "9876543210987654",
                          "Мария Георгиева", DateTime.Now.AddYears(1),
"456")
};

var processor = new PaymentProcessor();
```

```
processor.ProcessPayments(payments);
processor.ProcessPaymentsByType(payments);

// Полиморфизъм с интерфейси
Console.WriteLine("\n=== Полиморфизъм с интерфейси ===");
foreach (var payment in payments) {
    if (payment is IValidatable validatable) {
        Console.WriteLine($"{payment.GetType().Name} валиден:
{validatable.IsValid}");
    }

    if (payment is INotifiable notifiable) {
        notifiable.PaymentProcessed += (sender, message) => {
            Console.WriteLine($"Получено уведомление: {message}");
        };
    }
}
```

8. Предимства на полиморфизма

Ключови предимства:

- **Гъвкавост** - лесно добавяне на нови типове
- **Преизползване** - един код работи с различни обекти
- **Поддръжка** - лесно модифициране на функционалността
- **Тестване** - лесно създаване на mock обекти
- **Разширяемост** - лесно добавяне на нова функционалност

9. Практически задачи

Задачи за упражнение:

- **Създай йерархия от класове** - Vehicle -> Car, Motorcycle, Truck

- **Имплементирай полиморфизъм** - различни методи за Start, Stop, Accelerate
- **Направи интерфейс IShape** - с методи CalculateArea, CalculatePerimeter
- **Създай полиморфна система** - за различни геометрични фигури
- **Имплементирай полиморфизъм** - за различни типове документи

10. Заключение

Полиморфизмът е мощна функционалност, която ни позволява да създаваме гъвкав, преизползваем и лесен за поддръжка код, който работи с различни типове обекти чрез един интерфейс.

Ключови принципи:

- **Един интерфейс** - различни имплементации
- **Позднно свързване** - решението се взема по време на изпълнение
- **Гъвкавост** - лесно добавяне на нови типове
- **Преизползване** - един код за множество обекти
- **Разширяемост** - лесно добавяне на нова функционалност