Абстракция и абстрактни класове — 12б, ООП

1. Увод

Абстракцията е принцип на ООП, който ни позволява да скриваме сложността и да показваме само необходимата информация. Абстрактните класове са инструмент за дефиниране на общи интерфейси и функционалност.

2. Какво е абстракция?

Абстракцията включва:

- **Скриване на сложността** - показване само на необходимото
- **Общи интерфейси** - дефиниране на стандартни методи
- **Принудителна имплементация** - наследниците трябва да имплементират абстрактните методи
- **Преизползване на код** - обща функционалност в базовия клас

3. Абстрактни класове

Характеристики на абстрактните класове:

- **Не могат да се инстанцират** - не може да се създават обекти

- **Съдържат абстрактни методи** - без имплементация
- **Могат да съдържат конкретни методи** - с имплементация
- **Могат да имат конструктори** - за инициализация
- **Трябва да се наследяват** - за да се използват

Основен пример с абстрактен клас:

```
// Абстрактен клас
abstract class Shape {
    protected string color;
    protected DateTime createdDate;

    // Конструктор на абстрактния клас
    public Shape(string color) {
        this.color = color;
        this.createdDate = DateTime.Now;
        Console.WriteLine($"Създадена е {GetType().Name} с цвят {color}");
    }

    // Абстрактен метод - трябва да се имплементира
    public abstract double CalculateArea();

    public abstract double CalculatePerimeter();

    // Конкретен метод - може да се наследява
    public virtual void DisplayInfo() {
        Console.WriteLine($"Цвят: {color}");
        Console.WriteLine($"Създадена: {createdDate:dd.MM.yyyy HH:mm}");
        Console.WriteLine($"Площ: {CalculateArea():F2}");
        Console.WriteLine($"Периметър: {CalculatePerimeter():F2}");
    }

    // Конкретен метод
    public void ChangeColor(string newColor) {
        color = newColor;
        Console.WriteLine($"Цвятът е променен на {newColor}");
    }
}

// Конкретна имплементация
class Rectangle : Shape {
```

```
    private double width;
    private double height;

    public Rectangle(string color, double width, double height) :
base(color) {
        this.width = width;
        this.height = height;
        Console.WriteLine($"Създаден правоъгълник с размери
{width}x{height}");
    }

    // Имплементация на абстрактния метод
    public override double CalculateArea() {
        return width * height;
    }

    public override double CalculatePerimeter() {
        return 2 * (width + height);
    }
}

// Конкретна имплементация
class Circle : Shape {
    private double radius;

    public Circle(string color, double radius) : base(color) {
        this.radius = radius;
        Console.WriteLine($"Създаден кръг с радиус {radius}");
    }

    public override double CalculateArea() {
        return Math.PI * radius * radius;
    }

    public override double CalculatePerimeter() {
        return 2 * Math.PI * radius;
    }
}

// Използване
Shape rect = new Rectangle("Червен", 5, 3);
Shape circle = new Circle("Син", 4);

rect.DisplayInfo();
circle.DisplayInfo();
```

4. Абстрактни методи

Характеристики на абстрактните методи:

- **Нямат имплементация** - само декларация
- **Трябва да се override** - в производните класове
- **Могат да са virtual** - за допълнително override
- **Не могат да са static** - не могат да бъдат статични
- **Не могат да са private** - трябва да са достъпни за наследниците

Пример с различни абстрактни методи:

```
abstract class Animal {
    protected string name;
    protected int age;

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

    // Абстрактен метод без параметри
    public abstract void MakeSound();

    // Абстрактен метод с параметри
    public abstract void Move(int distance);

    // Абстрактен метод с връщане на стойност
    public abstract bool IsDangerous();

    // Абстрактен метод с параметри и връщане на стойност
    public abstract string GetInfo(string prefix);

    // Конкретен метод
    public void DisplayBasicInfo() {
        Console.WriteLine($"Име: {name}, Възраст: {age}");
    }
}
```

```
class Dog : Animal {
    private string breed;

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

    public override void MakeSound() {
        Console.WriteLine($"{name} лае: Woof! Woof!");
    }

    public override void Move(int distance) {
        Console.WriteLine($"{name} тича {distance} метра");
    }

    public override bool IsDangerous() {
        return false; // Кучетата обикновено не са опасни
    }

    public override string GetInfo(string prefix) {
        return $"{prefix} {name} е {breed} на {age} години";
    }
}

class Lion : Animal {
    private bool isMale;

    public Lion(string name, int age, bool isMale) : base(name, age) {
        this.isMale = isMale;
    }

    public override void MakeSound() {
        Console.WriteLine($"{name} реве: ROAR!");
    }

    public override void Move(int distance) {
        Console.WriteLine($"{name} бяга {distance} метра");
    }

    public override bool IsDangerous() {
        return true; // Лъвовете са опасни
    }

    public override string GetInfo(string prefix) {
        string gender = isMale ? "мъжки" : "женски";
        return $"{prefix} {name} е {gender} лъв на {age} години";
```

```
    }
}
```

5. Конкретни методи в абстрактни класове

Видове конкретни методи:

- **Virtual методи** - могат да се override
- **Обикновени методи** - не могат да се override
- **Статични методи** - достъпни без обект
- **Приватни методи** - само за вътрешно използване

Пример с различни типове методи:

```
abstract class Vehicle {
    protected string brand;
    protected int year;
    protected double fuelLevel;
    protected bool isRunning;

    public Vehicle(string brand, int year) {
        this.brand = brand;
        this.year = year;
        this.fuelLevel = 100.0;
        this.isRunning = false;
    }

    // Абстрактни методи - трябва да се имплементират
    public abstract void Start();
    public abstract void Stop();
    public abstract double CalculateFuelConsumption();

    // Virtual методи - могат да се override
    public virtual void Accelerate() {
        if (isRunning) {
            Console.WriteLine($"{brand} ускорява...");
```

```
            ConsumeFuel(5.0);
        }
    }

    public virtual void Brake() {
        Console.WriteLine($"{brand} спира...");
    }

    // Конкретни методи - не могат да се override
    public void Refuel(double amount) {
        fuelLevel = Math.Min(100.0, fuelLevel + amount);
        Console.WriteLine($"Заредено {amount} литра. Текущо ниво:
{fuelLevel:F1}%");
    }

    public void DisplayStatus() {
        Console.WriteLine($"Марка: {brand}");
        Console.WriteLine($"Година: {year}");
        Console.WriteLine($"Гориво: {fuelLevel:F1}%");
        Console.WriteLine($"Статус: {(isRunning ? "Работи" : "Спира")}");
    }

    // Приватен метод - само за вътрешно използване
    protected void ConsumeFuel(double amount) {
        fuelLevel = Math.Max(0, fuelLevel - amount);
        if (fuelLevel <= 0) {
            Console.WriteLine("Горивото свърши!");
            isRunning = false;
        }
    }

    // Статичен метод
    public static void DisplayVehicleInfo() {
        Console.WriteLine("Това е система за управление на превозни
средства");
    }
}

class Car : Vehicle {
    private int doors;

    public Car(string brand, int year, int doors) : base(brand, year) {
        this.doors = doors;
    }

    public override void Start() {
        isRunning = true;
```

```
        Console.WriteLine($"{brand} колата стартира с ключа");
    }

    public override void Stop() {
        isRunning = false;
        Console.WriteLine($"{brand} колата спира");
    }

    public override double CalculateFuelConsumption() {
        return 8.5; // литра на 100 км
    }

    public override void Accelerate() {
        base.Accelerate(); // Извиква базовия метод
        Console.WriteLine($"Колата с {doors} врати ускорява");
    }
}

class Motorcycle : Vehicle {
    private bool hasWindshield;

    public Motorcycle(string brand, int year, bool hasWindshield) :
base(brand, year) {
        this.hasWindshield = hasWindshield;
    }

    public override void Start() {
        isRunning = true;
        Console.WriteLine($"{brand} мотоциклетът стартира с кнопка");
    }

    public override void Stop() {
        isRunning = false;
        Console.WriteLine($"{brand} мотоциклетът спира");
    }

    public override double CalculateFuelConsumption() {
        return 4.2; // литра на 100 км
    }

    public void Wheelie() {
        if (isRunning) {
            Console.WriteLine($"{brand} мотоциклетът прави wheelie!");
        }
    }
}
```

6. Пълен пример - Система за документи

Реален пример с абстрактен клас:

```
// Абстрактен клас за документ
public abstract class Document {
    protected string title;
    protected string author;
    protected DateTime createdDate;
    protected bool isSigned;
    protected string documentId;
    protected static int documentCounter = 0;

    // Конструктор на абстрактния клас
    public Document(string title, string author) {
        this.title = title;
        this.author = author;
        this.createdDate = DateTime.Now;
        this.isSigned = false;
        this.documentId = GenerateDocumentId();
        documentCounter++;
        Console.WriteLine($"Създаден документ: {title} от {author}");
    }

    // Абстрактни методи - трябва да се имплементират
    public abstract void GenerateContent();
    public abstract bool Validate();
    public abstract string GetDocumentType();
    public abstract void Process();

    // Virtual методи - могат да се override
    public virtual void Sign(string signerName) {
        if (Validate()) {
            isSigned = true;
            Console.WriteLine($"Документът е подписан от {signerName}");
        } else {
            Console.WriteLine("Документът не може да бъде подписан - не е
валиден");
        }
    }
```

```
    public virtual void DisplayInfo() {
        Console.WriteLine($"ID: {documentId}");
        Console.WriteLine($"Заглавие: {title}");
        Console.WriteLine($"Автор: {author}");
        Console.WriteLine($"Създаден: {createdDate:dd.MM.yyyy HH:mm}");
        Console.WriteLine($"Тип: {GetDocumentType()}");
        Console.WriteLine($"Подписан: {(isSigned ? "Да" : "Не")}");
    }

    // Конкретни методи
    public void Archive() {
        Console.WriteLine($"Документът {title} е архивиран");
    }

    public void SendNotification(string recipient) {
        Console.WriteLine($"Изпратено уведомление до {recipient} за
документ {title}");
    }

    // Приватен метод за генериране на ID
    private string GenerateDocumentId() {
        return $"DOC_{DateTime.Now:yyyyMMdd}_{documentCounter:D4}";
    }

    // Статичен метод
    public static int GetTotalDocuments() {
        return documentCounter;
    }

    // Свойства
    public string Title { get { return title; } }
    public string Author { get { return author; } }
    public DateTime CreatedDate { get { return createdDate; } }
    public bool IsSigned { get { return isSigned; } }
    public string DocumentId { get { return documentId; } }
}

// Конкретна имплементация за договор
public class Contract : Document {
    private decimal amount;
    private DateTime expiryDate;
    private List parties;

    public Contract(string title, string author, decimal amount, DateTime
expiryDate)
        : base(title, author) {
        this.amount = amount;
```

```
        this.expiryDate = expiryDate;
        this.parties = new List();
        Console.WriteLine($"Създаден договор за сума {amount:C}");
    }

    public void AddParty(string partyName) {
        parties.Add(partyName);
        Console.WriteLine($"Добавена страна: {partyName}");
    }

    public override void GenerateContent() {
        Console.WriteLine($"Генерира се договор за сума {amount:C}");
        Console.WriteLine($"Валидност до: {expiryDate:dd.MM.yyyy}");
        Console.WriteLine($"Страни: {string.Join(", ", parties)}");
    }

    public override bool Validate() {
        bool isValid = amount > 0 && expiryDate > DateTime.Now &&
parties.Count >= 2;
        if (!isValid) {
            Console.WriteLine("Договорът не е валиден - проверете сумата,
датата и страните");
        }
        return isValid;
    }

    public override string GetDocumentType() {
        return "Договор";
    }

    public override void Process() {
        Console.WriteLine("Обработва се договор...");
        GenerateContent();
        if (Validate()) {
            Console.WriteLine("Договорът е готов за подписване");
        }
    }

    public override void Sign(string signerName) {
        if (parties.Contains(signerName)) {
            base.Sign(signerName);
            Console.WriteLine($"Договорът е подписан от {signerName}");
        } else {
            Console.WriteLine($"{signerName} не е страна в договора");
        }
    }
}
```

```
// Конкретна имплементация за фактура
public class Invoice : Document {
    private string invoiceNumber;
    private List items;
    private decimal totalAmount;

    public Invoice(string title, string author, string invoiceNumber)
        : base(title, author) {
        this.invoiceNumber = invoiceNumber;
        this.items = new List();
        this.totalAmount = 0;
        Console.WriteLine($"Създадена фактура №{invoiceNumber}");
    }

    public void AddItem(string description, decimal price, int quantity) {
        var item = new InvoiceItem(description, price, quantity);
        items.Add(item);
        totalAmount += item.TotalPrice;
        Console.WriteLine($"Добавен артикул: {description} x{quantity} =
{item.TotalPrice:C}");
    }

    public override void GenerateContent() {
        Console.WriteLine($"Генерира се фактура №{invoiceNumber}");
        Console.WriteLine($"Артикули ({items.Count}):");
        foreach (var item in items) {
            Console.WriteLine($"  - {item.Description}: {item.Price:C} x
{item.Quantity} = {item.TotalPrice:C}");
        }
        Console.WriteLine($"Обща сума: {totalAmount:C}");
    }

    public override bool Validate() {
        bool isValid = !string.IsNullOrEmpty(invoiceNumber) && items.Count
> 0 && totalAmount > 0;
        if (!isValid) {
            Console.WriteLine("Фактурата не е валидна - проверете номера,
артикулите и сумата");
        }
        return isValid;
    }

    public override string GetDocumentType() {
        return "Фактура";
    }
```

```
    public override void Process() {
        Console.WriteLine("Обработва се фактура...");
        GenerateContent();
        if (Validate()) {
            Console.WriteLine("Фактурата е готова за изпращане");
        }
    }
}

// Помощен клас за артикули във фактурата
public class InvoiceItem {
    public string Description { get; }
    public decimal Price { get; }
    public int Quantity { get; }
    public decimal TotalPrice { get; }

    public InvoiceItem(string description, decimal price, int quantity) {
        Description = description;
        Price = price;
        Quantity = quantity;
        TotalPrice = price * quantity;
    }
}

// Конкретна имплементация за отчет
public class Report : Document {
    private string reportType;
    private List sections;
    private int pageCount;

    public Report(string title, string author, string reportType)
        : base(title, author) {
        this.reportType = reportType;
        this.sections = new List();
        this.pageCount = 0;
        Console.WriteLine($"Създаден {reportType} отчет");
    }

    public void AddSection(string sectionTitle) {
        sections.Add(sectionTitle);
        pageCount += 2; // Приблизително 2 страници на секция
        Console.WriteLine($"Добавена секция: {sectionTitle}");
    }

    public override void GenerateContent() {
        Console.WriteLine($"Генерира се {reportType} отчет");
        Console.WriteLine($"Секции ({sections.Count}):");
```

```
        foreach (var section in sections) {
            Console.WriteLine($"  - {section}");
        }
        Console.WriteLine($"Общо страници: {pageCount}");
    }

    public override bool Validate() {
        bool isValid = !string.IsNullOrEmpty(reportType) && sections.Count
> 0;
        if (!isValid) {
            Console.WriteLine("Отчетът не е валиден - проверете типа и
секциите");
        }
        return isValid;
    }

    public override string GetDocumentType() {
        return $"Отчет ({reportType})";
    }

    public override void Process() {
        Console.WriteLine("Обработва се отчет...");
        GenerateContent();
        if (Validate()) {
            Console.WriteLine("Отчетът е готов за преглед");
        }
    }
}

// Използване
Document contract = new Contract("Договор за работа", "Иван Петров", 5000,
DateTime.Now.AddMonths(12));
contract.AddParty("Иван Петров");
contract.AddParty("Мария Георгиева");
contract.Process();
contract.Sign("Иван Петров");
contract.DisplayInfo();

Document invoice = new Invoice("Фактура за услуги", "Петър Стоянов", "INV-
2024-001");
invoice.AddItem("Консултантски услуги", 100, 10);
invoice.AddItem("Техническа поддръжка", 50, 5);
invoice.Process();
invoice.Sign("Анна Димитрова");
invoice.DisplayInfo();

Document report = new Report("Годишен отчет", "Анна Димитрова",
```

```
"Финансов");
report.AddSection("Приходи");
report.AddSection("Разходи");
report.AddSection("Заключение");
report.Process();
report.DisplayInfo();

Console.WriteLine($"Общо документи: {Document.GetTotalDocuments()}");
```

7. Предимства на абстрактните класове

Ключови предимства:

- **Принудителна имплементация** - наследниците трябва да имплементират абстрактните методи
- **Преизползване на код** - обща функционалност в базовия клас
- **Стандартизация** - дефиниране на общ интерфейс
- **Гъвкавост** - комбинация от абстрактни и конкретни методи
- **Типова безопасност** - компилаторът проверява имплементацията

8. Разлика между абстрактни класове и интерфейси

Абстрактни класове

- Могат да имат полета
- Могат да имат конструктори
- Могат да имат конкретни методи
- Поддържат единично наследяване
- Могат да имат модификатори за достъп

Интерфейси

- Не могат да имат полета (само свойства)
- Не могат да имат конструктори
- Всички методи са абстрактни
- Поддържат множествено наследяване
- Всички членове са публични

9. Практически задачи

Задачи за упражнение:

- **Създай абстрактен клас Media** с конкретни имплементации Book, Movie, Music
- **Имплементирай абстрактен клас Payment** с CreditCard, BankTransfer, Cash
- **Направи абстрактен клас Game** с BoardGame, VideoGame, CardGame
- **Създай абстрактен клас Notification** с Email, SMS, Push
- **Имплементирай абстрактен клас Storage** с FileStorage, DatabaseStorage, CloudStorage

10. Заключение

Абстрактните класове са мощна функционалност, която ни позволява да дефинираме общи интерфейси и функционалност, като в същото време принуждаваме наследниците да имплементират специфичната логика.

Ключови принципи:

- **Абстракция** - скриване на сложността
- **Стандартизация** - дефиниране на общ интерфейс
- **Принудителна имплементация** - гарантиране на необходимите методи
- **Преизползване** - обща функционалност в базовия клас
- **Гъвкавост** - комбинация от абстрактни и конкретни методи