Наследяване на класове — 12б, ООП

1. Увод

Наследяването е един от основните принципи на ООП, който ни позволява да създаваме нови класове, базирани на съществуващи, като преизползваме техния код и добавяме нова функционалност.

2. Основни понятия

Терминология при наследяване:

- **Базов клас (Parent/Base)** - класът, който се наследява
- **Производен клас (Child/Derived)** - класът, който наследява
- **Override** - пренаписване на наследен метод
- **Protected** - достъп само за наследниците
- **Virtual** - метод, който може да се пренапише

3. Синтаксис на наследяване

Основен синтаксис:

- **Двоеточие (:)** - показва наследяване
- **Base клас** - името на класа, който се наследява
- **Единично наследяване** - C# поддържа само едно наследяване

- **Множествено наследяване** - чрез интерфейси

Основен пример с наследяване:

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

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

    public virtual void MakeSound() {
        Console.WriteLine("Животното издава звук...");
    }

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

    public void DisplayInfo() {
        Console.WriteLine($"Име: {name}, Възраст: {age}");
    }
}

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

    // Конструктор на производния клас
    public Dog(string name, int age, string breed) : base(name, age) {
        this.breed = breed;
    }

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

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

```
    }

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

    public void DisplayBreed() {
        Console.WriteLine($"Порода: {breed}");
    }
}

// Използване
Animal animal = new Animal("Животно", 3);
Dog dog = new Dog("Рекс", 2, "Лабрадор");

animal.MakeSound(); // "Животното издава звук..."
dog.MakeSound();    // "Рекс лае: Woof! Woof!"
dog.Fetch();        // "Рекс донесе топката!"
```

4. Модификатори за достъп при наследяване

Достъп до членове:

- **Public** - достъпен отвсякъде
- **Protected** - достъпен в класа и наследниците
- **Private** - достъпен само в класа
- **Internal** - достъпен в същото assembly

Пример с различни модификатори:

```
class Vehicle {
    // Публично поле - достъпно отвсякъде
    public string brand;

    // Защитено поле - достъпно в класа и наследниците
```

```
    protected int year;

    // Приватно поле - достъпно само в този клас
    private string vin;

    public Vehicle(string brand, int year, string vin) {
        this.brand = brand;
        this.year = year;
        this.vin = vin;
    }

    // Публичен метод
    public void Start() {
        Console.WriteLine($"{brand} стартира...");
    }

    // Виртуален метод - може да се пренапише
    public virtual void DisplayInfo() {
        Console.WriteLine($"Марка: {brand}, Година: {year}");
    }

    // Защитен метод - достъпен в наследниците
    protected void CheckEngine() {
        Console.WriteLine("Проверка на двигателя...");
    }
}

class Car : Vehicle {
    private int doors;

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

    // Достъп до защитените членове
    public void PerformMaintenance() {
        CheckEngine(); // Може да извика защитения метод
        Console.WriteLine($"Поддръжка на {brand} завършена");
    }

    // Override на виртуалния метод
    public override void DisplayInfo() {
        base.DisplayInfo(); // Извиква базовия метод
        Console.WriteLine($"Брой врати: {doors}");
    }
```

```
    // Нов метод
    public void OpenTrunk() {
        Console.WriteLine("Багажникът е отворен");
    }
}
```

5. Конструктори при наследяване

Правила за конструктори:

- **Base конструктор** - винаги се извиква първи
- **Base ключова дума** - извикване на базовия конструктор
- **Параметри** - предаване на аргументи към базовия конструктор
- **По подразбиране** - ако няма base(), се извиква конструкторът по подразбиране

Пример с конструктори:

```
class Person {
    protected string name;
    protected int age;
    protected string email;

    // Конструктор по подразбиране
    public Person() {
        name = "Неизвестен";
        age = 0;
        email = "";
        Console.WriteLine("Person конструктор по подразбиране");
    }

    // Конструктор с параметри
    public Person(string name, int age) {
        this.name = name;
        this.age = age;
```

```
        this.email = "";
        Console.WriteLine($"Person конструктор: {name}");
    }

    // Конструктор с всички параметри
    public Person(string name, int age, string email) {
        this.name = name;
        this.age = age;
        this.email = email;
        Console.WriteLine($"Person конструктор: {name}, {email}");
    }
}

class Student : Person {
    private string studentId;
    private double gpa;

    // Конструктор по подразбиране
    public Student() : base() {
        studentId = "";
        gpa = 0.0;
        Console.WriteLine("Student конструктор по подразбиране");
    }

    // Конструктор с параметри - извиква base(name, age)
    public Student(string name, int age, string studentId)
        : base(name, age) {
        this.studentId = studentId;
        this.gpa = 0.0;
        Console.WriteLine($"Student конструктор: {name}, {studentId}");
    }

    // Конструктор с всички параметри
    public Student(string name, int age, string email, string studentId,
double gpa)
        : base(name, age, email) {
        this.studentId = studentId;
        this.gpa = gpa;
        Console.WriteLine($"Student конструктор: {name}, {studentId},
{gpa}");
    }
}

// Използване
Student student1 = new Student(); // Извиква всички конструктори
Student student2 = new Student("Иван", 20, "ST001"); // Извиква base(name,
age)
```

```
Student student3 = new Student("Мария", 19, "maria@email.com", "ST002",
5.5);
```

6. Override и Virtual методи

Ключови думи:

- **Virtual** - метод в базовия клас, който може да се пренапише
- **Override** - пренаписване на виртуален метод
- **New** - скриване на наследен метод (не препоръчва се)
- **Base** - извикване на базовия метод

Пример с virtual и override:

```
class Shape {
    protected string color;

    public Shape(string color) {
        this.color = color;
    }

    // Виртуален метод - може да се пренапише
    public virtual double CalculateArea() {
        return 0;
    }

    // Виртуален метод
    public virtual double CalculatePerimeter() {
        return 0;
    }

    // Обикновен метод - не може да се пренапише
    public void DisplayColor() {
        Console.WriteLine($"Цвят: {color}");
    }
}
```

```
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;
    }

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

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

class Circle : Shape {
    private double radius;

    public Circle(string color, double radius) : base(color) {
        this.radius = radius;
    }

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

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

// Използване
Shape shape1 = new Rectangle("Червен", 5, 3);
Shape shape2 = new Circle("Син", 4);

Console.WriteLine($"Правоъгълник - площ: {shape1.CalculateArea():F2}");
Console.WriteLine($"Кръг - площ: {shape2.CalculateArea():F2}");
```

7. Пълен пример - Система за служители

Реален пример с наследяване:

```
// Базов клас за служител
public class Employee {
    protected string firstName;
    protected string lastName;
    protected string employeeId;
    protected decimal baseSalary;
    protected DateTime hireDate;

    public Employee(string firstName, string lastName, string employeeId,
decimal baseSalary) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.employeeId = employeeId;
        this.baseSalary = baseSalary;
        this.hireDate = DateTime.Now;
    }

    // Виртуален метод за изчисляване на заплата
    public virtual decimal CalculateSalary() {
        return baseSalary;
    }

    // Виртуален метод за заглавие
    public virtual string GetJobTitle() {
        return "Служител";
    }

    // Виртуален метод за работа
    public virtual void Work() {
        Console.WriteLine($"{GetFullName()} работи...");
    }

    // Обикновен метод
    public string GetFullName() {
        return $"{firstName} {lastName}";
    }

    public int GetYearsOfService() {
        return DateTime.Now.Year - hireDate.Year;
    }
```

```
    public virtual void DisplayInfo() {
        Console.WriteLine($"ID: {employeeId}");
        Console.WriteLine($"Име: {GetFullName()}");
        Console.WriteLine($"Длъжност: {GetJobTitle()}");
        Console.WriteLine($"Заплата: {CalculateSalary():C}");
        Console.WriteLine($"Стаж: {GetYearsOfService()} години");
    }
}

// Производен клас за мениджър
public class Manager : Employee {
    private decimal bonus;
    private int teamSize;
    private List teamMembers;

    public Manager(string firstName, string lastName, string employeeId,
                   decimal baseSalary, decimal bonus, int teamSize)
                   : base(firstName, lastName, employeeId, baseSalary) {
        this.bonus = bonus;
        this.teamSize = teamSize;
        this.teamMembers = new List();
    }

    // Override на виртуалния метод
    public override decimal CalculateSalary() {
        return baseSalary + bonus;
    }

    public override string GetJobTitle() {
        return "Мениджър";
    }

    public override void Work() {
        Console.WriteLine($"{GetFullName()} управлява екип от {teamSize}
души");
        ConductMeeting();
    }

    // Нови методи специфични за мениджъра
    public void ConductMeeting() {
        Console.WriteLine($"{GetFullName()} провежда среща с екипа");
    }

    public void AddTeamMember(string memberName) {
        if (teamMembers.Count < teamSize) {
            teamMembers.Add(memberName);
```

```
            Console.WriteLine($"Добавен член в екипа: {memberName}");
        } else {
            Console.WriteLine("Екипът е пълен!");
        }
    }

    public void RemoveTeamMember(string memberName) {
        if (teamMembers.Remove(memberName)) {
            Console.WriteLine($"Премахнат член от екипа: {memberName}");
        } else {
            Console.WriteLine("Членът не е намерен в екипа");
        }
    }

    public override void DisplayInfo() {
        base.DisplayInfo(); // Извиква базовия метод
        Console.WriteLine($"Бонус: {bonus:C}");
        Console.WriteLine($"Размер на екипа: {teamSize}");
        Console.WriteLine($"Членове на екипа: {string.Join(", ",
teamMembers)}");
    }
}

// Производен клас за разработчик
public class Developer : Employee {
    private string programmingLanguage;
    private int projectsCompleted;
    private List skills;

    public Developer(string firstName, string lastName, string employeeId,
                     decimal baseSalary, string programmingLanguage, int
projectsCompleted)
                     : base(firstName, lastName, employeeId, baseSalary) {
        this.programmingLanguage = programmingLanguage;
        this.projectsCompleted = projectsCompleted;
        this.skills = new List();
    }

    public override decimal CalculateSalary() {
        return baseSalary + (projectsCompleted * 500); // Бонус за проекти
    }

    public override string GetJobTitle() {
        return "Разработчик";
    }

    public override void Work() {
```

```
        Console.WriteLine($"{GetFullName()} пише код на
{programmingLanguage}");
        CompleteProject();
    }

    // Нови методи специфични за разработчика
    public void CompleteProject() {
        projectsCompleted++;
        Console.WriteLine($"Проект завършен! Общо проекти:
{projectsCompleted}");
    }

    public void AddSkill(string skill) {
        if (!skills.Contains(skill)) {
            skills.Add(skill);
            Console.WriteLine($"Добавено умение: {skill}");
        }
    }

    public void WriteCode() {
        Console.WriteLine($"{GetFullName()} пише код на
{programmingLanguage}");
    }

    public override void DisplayInfo() {
        base.DisplayInfo();
        Console.WriteLine($"Език за програмиране: {programmingLanguage}");
        Console.WriteLine($"Завършени проекти: {projectsCompleted}");
        Console.WriteLine($"Умения: {string.Join(", ", skills)}");
    }
}

// Производен клас за стажант
public class Intern : Employee {
    private string university;
    private int monthsRemaining;

    public Intern(string firstName, string lastName, string employeeId,
                  decimal baseSalary, string university, int
monthsRemaining)
                  : base(firstName, lastName, employeeId, baseSalary) {
        this.university = university;
        this.monthsRemaining = monthsRemaining;
    }

    public override decimal CalculateSalary() {
        return baseSalary * 0.5m; // Стажантите получават 50% от базовата
```

```
заплата
    }

    public override string GetJobTitle() {
        return "Стажант";
    }

    public override void Work() {
        Console.WriteLine($"{GetFullName()} учи и помага с задачи");
        Learn();
    }

    public void Learn() {
        Console.WriteLine($"{GetFullName()} учи от опита на колегите");
        monthsRemaining--;
        Console.WriteLine($"Останали месеци: {monthsRemaining}");
    }

    public bool IsInternshipComplete() {
        return monthsRemaining <= 0;
    }

    public override void DisplayInfo() {
        base.DisplayInfo();
        Console.WriteLine($"Университет: {university}");
        Console.WriteLine($"Останали месеци: {monthsRemaining}");
        Console.WriteLine($"Стажът е завършен: {(IsInternshipComplete() ?
"Да" : "Не")}");
    }
}

// Използване
Employee emp1 = new Employee("Иван", "Иванов", "EMP001", 2000);
Manager mgr1 = new Manager("Мария", "Петрова", "MGR001", 3000, 1000, 5);
Developer dev1 = new Developer("Петър", "Стоянов", "DEV001", 2500, "C#",
3);
Intern intern1 = new Intern("Анна", "Димитрова", "INT001", 1000, "СУ", 6);

// Добавяне на членове в екипа на мениджъра
mgr1.AddTeamMember("Петър Стоянов");
mgr1.AddTeamMember("Анна Димитрова");

// Добавяне на умения на разработчика
dev1.AddSkill("C#");
dev1.AddSkill("SQL");
dev1.AddSkill("ASP.NET");
```

```
// Показване на информация
emp1.DisplayInfo();
Console.WriteLine();
mgr1.DisplayInfo();
Console.WriteLine();
dev1.DisplayInfo();
Console.WriteLine();
intern1.DisplayInfo();
```

8. Предимства на наследяването

Ключови предимства:

- **Преизползване на код** - не се налага да пренаписваме функционалност
- **Иерархия** - логическо групиране на свързани класове
- **Полиморфизъм** - един интерфейс, различни имплементации
- **Поддръжка** - лесно добавяне на нови функции
- **Специализация** - всеки клас може да има специфична функционалност

9. Практически задачи

Задачи за упражнение:

- **Създай йерархия от класове** - Vehicle -> Car, Motorcycle, Truck
- **Имплементирай система за животни** - Animal -> Mammal, Bird, Fish
- **Направи йерархия за документи** - Document -> Contract, Invoice, Report
- **Създай система за геометрични фигури** - Shape -> Rectangle, Circle, Triangle

- **Имплементирай йерархия за мебели** - Furniture -> Chair, Table, Sofa

10. Заключение

Наследяването е мощна функционалност, която ни позволява да създаваме йерархии от класове и да преизползваме код ефективно, като в същото време добавяме специфична функционалност за всеки клас.

Ключови принципи:

- **IS-A връзка** - производният клас Е вид на базовия
- **Преизползване** - наследяване на код и функционалност
- **Специализация** - добавяне на специфична функционалност
- **Полиморфизъм** - различни имплементации на същия интерфейс
- **Иерархия** - логическо групиране на свързани класове