Дефиниране на класове — 12б, ООП

1. Увод

Класовете са основната единица в обектно-ориентираното програмиране. Те дефинират структурата и поведението на обектите, които ще създаваме в нашите програми.

2. Основни компоненти на клас

Класът се състои от:

- **Полета (Fields)** - променливи, които съхраняват данни
- **Свойства (Properties)** - контролиран достъп до полетата
- **Методи (Methods)** - функции, които извършват действия
- **Конструктор (Constructor)** - метод за инициализация на обект

3. Полета (Fields)

Характеристики на полетата:

- **Данни** - съхраняват състоянието на обекта
- **Модификатори за достъп** - private, protected, public
- **Тип данни** - int, string, double, bool и др.

- **Инициализация** - могат да имат стойности по подразбиране

Пример с полета:

```
class Student {
    // Публични полета (не препоръчва се)
    public string name;
    public int age;

    // Приватни полета (добра практика)
    private double grade;
    private bool isActive;

    // Полета с инициализация
    private DateTime enrollmentDate = DateTime.Now;
    private static int totalStudents = 0;
}
```

4. Свойства (Properties)

Предимства на свойствата:

- **Инкапсулация** - контролиран достъп до данни
- **Валидация** - проверка на входните данни
- **Гъвкавост** - get и set логика
- **Безопасност** - скриване на вътрешната реализация

Основни типове свойства:

```
class Student {
    private string name;
```

```
    private int age;
    private double grade;

    // Автоматично свойство
    public string StudentId { get; set; }

    // Свойство с get и set
    public string Name {
        get { return name; }
        set {
            if (!string.IsNullOrEmpty(value)) {
                name = value;
            }
        }
    }

    // Read-only свойство
    public int Age {
        get { return age; }
    }

    // Свойство с валидация
    public double Grade {
        get { return grade; }
        set {
            if (value >= 2.0 && value <= 6.0) {
                grade = value;
            } else {
                throw new ArgumentException("Оценката трябва да е между 2.0
и 6.0");
            }
        }
    }

    // Свойство само за четене
    public bool IsExcellent {
        get { return grade >= 5.5; }
    }
}
```

5. Методи (Methods)

Видове методи:

- **Публични** - достъпни отвън
- **Приватни** - само вътре в класа
- **Статични** - извикват се без обект
- **Виртуални** - могат да се пренаписват

Примери с методи:

```
class Student {
    private string name;
    private double grade;

    // Публичен метод
    public void Study() {
        Console.WriteLine($"{name} учи...");
        grade += 0.1; // Увеличава оценката леко
    }

    // Метод с параметри и връщане на стойност
    public bool CanGraduate(double minimumGrade) {
        return grade >= minimumGrade;
    }

    // Приватен метод (вътрешен помощен)
    private void UpdateStatus() {
        if (grade >= 5.5) {
            Console.WriteLine("Отличен успех!");
        }
    }

    // Статичен метод
    public static void DisplaySchoolInfo() {
        Console.WriteLine("Добротворно училище 'Св. Климент Охридски'");
    }

    // Метод с ref параметър
    public void UpdateGrade(ref double newGrade) {
        if (newGrade >= 2.0 && newGrade <= 6.0) {
```

```
            grade = newGrade;
        }
    }

    // Метод с out параметър
    public bool TryGetInfo(out string studentInfo) {
        studentInfo = $"Име: {name}, Оценка: {grade}";
        return !string.IsNullOrEmpty(name);
    }
}
```

6. Конструктори (Constructors)

Видове конструктори:

- **По подразбиране** - без параметри
- **С параметри** - приема стойности
- **Копиращ** - копира от друг обект
- **Статичен** - инициализира статични членове

Примери с конструктори:

```
class Student {
    private string name;
    private int age;
    private double grade;
    private static int totalStudents;

    // Статичен конструктор
    static Student() {
        totalStudents = 0;
        Console.WriteLine("Класът Student е инициализиран");
    }

    // Конструктор по подразбиране
    public Student() {
```

```
        name = "Неизвестен";
        age = 0;
        grade = 2.0;
        totalStudents++;
    }

    // Конструктор с параметри
    public Student(string name, int age) {
        this.name = name;
        this.age = age;
        this.grade = 2.0;
        totalStudents++;
        Console.WriteLine($"Създаден студент: {name}");
    }

    // Конструктор с всички параметри
    public Student(string name, int age, double grade) : this(name, age) {
        this.grade = grade;
    }

    // Копиращ конструктор
    public Student(Student other) {
        this.name = other.name;
        this.age = other.age;
        this.grade = other.grade;
        totalStudents++;
    }

    // Статично свойство
    public static int TotalStudents {
        get { return totalStudents; }
    }
}
```

7. Пълен пример - Клас BankAccount

Реален пример с всички компоненти:

```
public class BankAccount {
    // Полета
    private string accountNumber;
```

```
    private string ownerName;
    private decimal balance;
    private bool isActive;
    private DateTime createdDate;
    private static int accountCounter = 0;

    // Статичен конструктор
    static BankAccount() {
        Console.WriteLine("Банковата система е инициализирана");
    }

    // Конструктор по подразбиране
    public BankAccount() {
        accountNumber = GenerateAccountNumber();
        ownerName = "Неизвестен";
        balance = 0;
        isActive = true;
        createdDate = DateTime.Now;
        accountCounter++;
    }

    // Конструктор с параметри
    public BankAccount(string ownerName, decimal initialBalance) {
        this.accountNumber = GenerateAccountNumber();
        this.ownerName = ownerName;
        this.balance = initialBalance;
        this.isActive = true;
        this.createdDate = DateTime.Now;
        accountCounter++;
        Console.WriteLine($"Създадена сметка за {ownerName}");
    }

    // Свойства
    public string AccountNumber {
        get { return accountNumber; }
    }

    public string OwnerName {
        get { return ownerName; }
        set {
            if (!string.IsNullOrEmpty(value)) {
                ownerName = value;
            }
        }
    }

    public decimal Balance {
```

```
        get { return balance; }
    }

    public bool IsActive {
        get { return isActive; }
    }

    public DateTime CreatedDate {
        get { return createdDate; }
    }

    public static int TotalAccounts {
        get { return accountCounter; }
    }

    // Методи
    public void Deposit(decimal amount) {
        if (!isActive) {
            Console.WriteLine("Сметката е неактивна!");
            return;
        }

        if (amount > 0) {
            balance += amount;
            Console.WriteLine($"Депозит от {amount:C}. Нов баланс:
{balance:C}");
        } else {
            Console.WriteLine("Сумата трябва да е положителна!");
        }
    }

    public bool Withdraw(decimal amount) {
        if (!isActive) {
            Console.WriteLine("Сметката е неактивна!");
            return false;
        }

        if (amount > 0 && amount <= balance) {
            balance -= amount;
            Console.WriteLine($"Теглене от {amount:C}. Остатък:
{balance:C}");
            return true;
        } else {
            Console.WriteLine("Недостатъчен баланс или невалидна сума!");
            return false;
        }
    }
```

```
    public void CloseAccount() {
        if (balance == 0) {
            isActive = false;
            accountCounter--;
            Console.WriteLine("Сметката е закрита успешно");
        } else {
            Console.WriteLine("Не може да се закрие сметка с ненулев
баланс!");
        }
    }

    public void DisplayInfo() {
        Console.WriteLine($"Сметка: {accountNumber}");
        Console.WriteLine($"Собственик: {ownerName}");
        Console.WriteLine($"Баланс: {balance:C}");
        Console.WriteLine($"Статус: {(isActive ? "Активна" :
"Неактивна")}");
        Console.WriteLine($"Създадена: {createdDate:dd.MM.yyyy HH:mm}");
    }

    // Приватен помощен метод
    private string GenerateAccountNumber() {
        return $"BG{DateTime.Now:yyyyMMdd}{accountCounter:D4}";
    }

    // Статичен метод
    public static void DisplayBankInfo() {
        Console.WriteLine($"Общо сметки: {accountCounter}");
        Console.WriteLine("Добротворна банка - Надеждност и сигурност");
    }
}

// Използване на класа
class Program {
    static void Main() {
        // Създаване на сметки
        BankAccount account1 = new BankAccount("Иван Петров", 1000);
        BankAccount account2 = new BankAccount("Мария Георгиева", 500);

        // Операции със сметките
        account1.Deposit(200);
        account1.Withdraw(150);
        account1.DisplayInfo();

        account2.Deposit(300);
        account2.DisplayInfo();
```

```
        // Статичен метод
        BankAccount.DisplayBankInfo();
    }
}
```

8. Модификатори за достъп

Основни модификатори:

- **public** - достъпен отвсякъде
- **private** - достъпен само в класа
- **protected** - достъпен в класа и наследниците
- **internal** - достъпен в същото assembly
- **protected internal** - комбинация от protected и internal

Пример с модификатори:

```
public class ExampleClass {
    public string publicField = "Публично поле";
    private string privateField = "Приватно поле";
    protected string protectedField = "Защитено поле";
    internal string internalField = "Вътрешно поле";

    public void PublicMethod() {
        Console.WriteLine("Публичен метод");
        PrivateMethod(); // Може да извика приватния метод
    }

    private void PrivateMethod() {
        Console.WriteLine("Приватен метод");
    }

    protected void ProtectedMethod() {
        Console.WriteLine("Защитен метод");
```

```
    }

    internal void InternalMethod() {
        Console.WriteLine("Вътрешен метод");
    }
}
```

9. Практически задачи

Задачи за упражнение:

- **Създай клас Car** с полета за марка, модел, година, цена и цвят
- **Добави свойства** с валидация за всички полета
- **Имплементирай методи** за стартиране, спиране и показване на информация
- **Създай конструктори** - по подразбиране и с параметри
- **Добави статичен брояч** за общия брой коли

10. Заключение

Класовете са основата на ООП. Те ни позволяват да групираме данни и функционалност в логически единици, което прави кода по-организиран и лесен за поддръжка.

Ключови принципи:

- **Инкапсулация** - скриване на вътрешната реализация
- **Валидация** - проверка на входните данни
- **Инициализация** - правилно задаване на начални стойности

- **Контрол на достъпа** - използване на подходящи модификатори
- **Статични членове** - споделени между всички инстанции