Статични членове на клас — 12б, ООП

1. Увод

Статичните членове принадлежат на класа, а не на конкретна инстанция. Те се споделят между всички обекти от дадения клас и могат да се използват без да се създава обект.

2. Основни характеристики

Статичните членове:

- **Принадлежат на класа** - не на инстанцията
- **Споделени** - една копия за всички обекти
- **Достъпни без обект** - чрез името на класа
- **Инициализират се веднъж** - при първото използване
- **Живеят през цялата програма** - до нейното завършване

3. Статични полета

Характеристики на статичните полета:

- **Една копия** - за всички инстанции
- **Инициализация** - при първото използване

- **Достъп** - чрез името на класа
- **Живот** - през цялата програма

Пример с статични полета:

```
class Student {
    // Обикновени полета (за всяка инстанция)
    public string name;
    public int age;

    // Статично поле (споделено за всички)
    public static int totalStudents = 0;
    public static string schoolName = "Добротворно училище";

    // Константа (статична по природа)
    public const int MAX_AGE = 25;

    // Readonly статично поле
    public static readonly DateTime schoolFounded = new DateTime(1990, 9,
15);

    public Student(string name, int age) {
        this.name = name;
        this.age = age;
        totalStudents++; // Увеличаваме брояча
    }

    public void DisplayInfo() {
        Console.WriteLine($"Име: {name}, Възраст: {age}");
        Console.WriteLine($"Общо студенти: {totalStudents}");
        Console.WriteLine($"Училище: {schoolName}");
    }
}

// Използване
Student student1 = new Student("Иван", 18);
Student student2 = new Student("Мария", 19);

Console.WriteLine($"Общо студенти: {Student.totalStudents}");
```

```
Console.WriteLine($"Училище: {Student.schoolName}");
Console.WriteLine($"Максимална възраст: {Student.MAX_AGE}");
```

4. Статични методи

Характеристики на статичните методи:

- **Извикват се без обект** - чрез името на класа
- **Нямат достъп до this** - не могат да използват инстанционни членове
- **Могат да използват статични членове** - полета и методи
- **Утилитарни функции** - често се използват за помощни операции

Пример с статични методи:

```
class MathHelper {
    // Статично поле
    private static int operationCount = 0;

    // Статичен метод
    public static double CalculateCircleArea(double radius) {
        operationCount++;
        return Math.PI * radius * radius;
    }

    public static double CalculateRectangleArea(double width, double
height) {
        operationCount++;
        return width * height;
    }

    public static double CalculateTriangleArea(double base, double height)
{
        operationCount++;
        return 0.5 * base * height;
    }
```

```
    // Статичен метод за валидация
    public static bool IsValidNumber(double number) {
        return !double.IsNaN(number) && !double.IsInfinity(number);
    }

    // Статично свойство
    public static int OperationCount {
        get { return operationCount; }
    }

    // Статичен метод за нулиране
    public static void ResetCounter() {
        operationCount = 0;
    }
}

// Използване
double circleArea = MathHelper.CalculateCircleArea(5);
double rectangleArea = MathHelper.CalculateRectangleArea(4, 6);
double triangleArea = MathHelper.CalculateTriangleArea(3, 4);

Console.WriteLine($"Кръг: {circleArea:F2}");
Console.WriteLine($"Правоъгълник: {rectangleArea:F2}");
Console.WriteLine($"Триъгълник: {triangleArea:F2}");
Console.WriteLine($"Извършени операции: {MathHelper.OperationCount}");
```

5. Статичен конструктор

Характеристики на статичния конструктор:

- **Извиква се автоматично** - при първото използване на класа
- **Без параметри** - не може да приема аргументи
- **Единствен** - може да има само един статичен конструктор
- **Инициализация** - за статични полета

Пример с статичен конструктор:

```
class DatabaseManager {
    // Статични полета
    private static string connectionString;
    private static bool isInitialized;
    private static DateTime lastConnection;

    // Статичен конструктор
    static DatabaseManager() {
        Console.WriteLine("Инициализиране на базата данни...");
        connectionString =
"Server=localhost;Database=SchoolDB;Trusted_Connection=true;";
        isInitialized = true;
        lastConnection = DateTime.Now;
        Console.WriteLine("Базата данни е готова за използване");
    }

    // Статични методи
    public static bool Connect() {
        if (!isInitialized) {
            Console.WriteLine("Базата данни не е инициализирана!");
            return false;
        }

        Console.WriteLine("Свързване с базата данни...");
        lastConnection = DateTime.Now;
        return true;
    }

    public static void Disconnect() {
        Console.WriteLine("Прекъсване на връзката с базата данни");
    }

    public static string GetConnectionString() {
        return connectionString;
    }

    public static DateTime GetLastConnection() {
        return lastConnection;
    }
}

// Използване - статичният конструктор се извиква автоматично
bool connected = DatabaseManager.Connect();
Console.WriteLine($"Връзката е: {(connected ? "Успешна" : "Неуспешна")}");
```

```
Console.WriteLine($"Последна връзка:
{DatabaseManager.GetLastConnection()}");
```

6. Статични класове

Характеристики на статичните класове:

- **Не могат да се инстанцират** - не може да се създават обекти
- **Съдържат само статични членове** - полета, свойства, методи
- **Не могат да се наследяват** - sealed по природа
- **Утилитарни класове** - за помощни функции

Пример с статичен клас:

```
public static class StringHelper {
    // Статични константи
    public const string DEFAULT_SEPARATOR = ", ";
    public const int MAX_LENGTH = 1000;

    // Статични методи
    public static string Reverse(string input) {
        if (string.IsNullOrEmpty(input)) return string.Empty;

        char[] chars = input.ToCharArray();
        Array.Reverse(chars);
        return new string(chars);
    }

    public static string CapitalizeFirst(string input) {
        if (string.IsNullOrEmpty(input)) return string.Empty;

        return char.ToUpper(input[0]) + input.Substring(1).ToLower();
    }

    public static string RemoveSpaces(string input) {
        if (string.IsNullOrEmpty(input)) return string.Empty;
```

```
        return input.Replace(" ", "");
    }

    public static bool IsPalindrome(string input) {
        if (string.IsNullOrEmpty(input)) return false;

        string cleaned = RemoveSpaces(input.ToLower());
        string reversed = Reverse(cleaned);

        return cleaned == reversed;
    }

    public static string[] SplitWords(string input) {
        if (string.IsNullOrEmpty(input)) return new string[0];

        return input.Split(new char[] { ' ', '\t', '\n', '\r' },
                           StringSplitOptions.RemoveEmptyEntries);
    }

    public static string JoinWords(string[] words, string separator =
DEFAULT_SEPARATOR) {
        if (words == null || words.Length == 0) return string.Empty;

        return string.Join(separator, words);
    }
}

// Използване
string text = "Hello World";
Console.WriteLine($"Оригинал: {text}");
Console.WriteLine($"Обърнат: {StringHelper.Reverse(text)}");
Console.WriteLine($"С главна буква: {StringHelper.CapitalizeFirst(text)}");
Console.WriteLine($"Без интервали: {StringHelper.RemoveSpaces(text)}");

string[] words = StringHelper.SplitWords("This is a test");
string joined = StringHelper.JoinWords(words);
Console.WriteLine($"Свързани: {joined}");
```

7. Пълен пример - Статичен клас за валидация

Реален пример с всички статични членове:

```
public static class Validator {
    // Статични константи
    public const int MIN_PASSWORD_LENGTH = 8;
    public const int MAX_EMAIL_LENGTH = 100;
    public const int MIN_AGE = 0;
    public const int MAX_AGE = 150;

    // Статични полета
    private static int validationCount = 0;
    private static List validationErrors = new List();

    // Статичен конструктор
    static Validator() {
        Console.WriteLine("Validator системата е инициализирана");
        validationErrors.Clear();
    }

    // Статични свойства
    public static int ValidationCount {
        get { return validationCount; }
    }

    public static List Errors {
        get { return new List(validationErrors); }
    }

    // Статични методи за валидация
    public static bool IsValidEmail(string email) {
        validationCount++;
        validationErrors.Clear();

        if (string.IsNullOrEmpty(email)) {
            validationErrors.Add("Email не може да бъде празен");
            return false;
        }

        if (email.Length > MAX_EMAIL_LENGTH) {
            validationErrors.Add($"Email не може да бъде по-дълъг от
{MAX_EMAIL_LENGTH} символа");
            return false;
        }

        if (!email.Contains("@")) {
            validationErrors.Add("Email трябва да съдържа @");
            return false;
        }
```

```
        if (!email.Contains(".")) {
            validationErrors.Add("Email трябва да съдържа точка");
            return false;
        }

        return true;
    }

    public static bool IsValidPassword(string password) {
        validationCount++;
        validationErrors.Clear();

        if (string.IsNullOrEmpty(password)) {
            validationErrors.Add("Паролата не може да бъде празна");
            return false;
        }

        if (password.Length < MIN_PASSWORD_LENGTH) {
            validationErrors.Add($"Паролата трябва да бъде поне
{MIN_PASSWORD_LENGTH} символа");
            return false;
        }

        if (!password.Any(char.IsUpper)) {
            validationErrors.Add("Паролата трябва да съдържа поне една
главна буква");
            return false;
        }

        if (!password.Any(char.IsLower)) {
            validationErrors.Add("Паролата трябва да съдържа поне една
малка буква");
            return false;
        }

        if (!password.Any(char.IsDigit)) {
            validationErrors.Add("Паролата трябва да съдържа поне една
цифра");
            return false;
        }

        return true;
    }

    public static bool IsValidAge(int age) {
        validationCount++;
```

```
        validationErrors.Clear();

        if (age < MIN_AGE) {
            validationErrors.Add($"Възрастта не може да бъде по-малка от
{MIN_AGE}");
            return false;
        }

        if (age > MAX_AGE) {
            validationErrors.Add($"Възрастта не може да бъде по-голяма от
{MAX_AGE}");
            return false;
        }

        return true;
    }

    public static bool IsValidPhone(string phone) {
        validationCount++;
        validationErrors.Clear();

        if (string.IsNullOrEmpty(phone)) {
            validationErrors.Add("Телефонът не може да бъде празен");
            return false;
        }

        // Премахваме всички символи освен цифри
        string cleanPhone = new
string(phone.Where(char.IsDigit).ToArray());

        if (cleanPhone.Length != 10) {
            validationErrors.Add("Телефонът трябва да съдържа точно 10
цифри");
            return false;
        }

        if (!cleanPhone.StartsWith("0")) {
            validationErrors.Add("Телефонът трябва да започва с 0");
            return false;
        }

        return true;
    }

    // Статичен метод за показване на грешки
    public static void DisplayErrors() {
        if (validationErrors.Count > 0) {
```

```
            Console.WriteLine("Грешки при валидация:");
            foreach (string error in validationErrors) {
                Console.WriteLine($"- {error}");
            }
        } else {
            Console.WriteLine("Няма грешки при валидация");
        }
    }

    // Статичен метод за нулиране
    public static void Reset() {
        validationCount = 0;
        validationErrors.Clear();
        Console.WriteLine("Validator системата е нулирана");
    }
}

// Използване
Console.WriteLine("=== Тестване на Validator ===");

// Тестване на email
bool validEmail = Validator.IsValidEmail("user@example.com");
Console.WriteLine($"Email валиден: {validEmail}");
if (!validEmail) Validator.DisplayErrors();

// Тестване на парола
bool validPassword = Validator.IsValidPassword("MyPass123");
Console.WriteLine($"Парола валидна: {validPassword}");
if (!validPassword) Validator.DisplayErrors();

// Тестване на възраст
bool validAge = Validator.IsValidAge(25);
Console.WriteLine($"Възраст валидна: {validAge}");

// Тестване на телефон
bool validPhone = Validator.IsValidPhone("0888123456");
Console.WriteLine($"Телефон валиден: {validPhone}");

Console.WriteLine($"Общо валидации: {Validator.ValidationCount}");
```

8. Разлика между статични и инстанционни членове

Статични членове

- Принадлежат на класа
- Една копия за всички обекти
- Достъпни без обект
- Използват `static` ключова дума
- Живеят през цялата програма

Инстанционни членове

- Принадлежат на обекта
- Отделна копия за всеки обект
- Достъпни чрез обект
- Без `static` ключова дума
- Живеят докато обектът съществува

9. Практически задачи

Задачи за упражнение:

- **Създай статичен клас Calculator** с методи за основни математически операции
- **Имплементирай статичен клас FileHelper** за работа с файлове
- **Създай клас с статичен брояч** за проследяване на създадените обекти
- **Направи статичен клас Logger** за записване на съобщения
- **Имплементирай статичен клас RandomHelper** за генериране на случайни числа

10. Заключение

Статичните членове са мощна функционалност в C#, която ни позволява да създаваме утилитарни функции и да споделяме данни между всички инстанции на клас.

Ключови принципи:

- **Споделяне на данни** - една копия за всички обекти
- **Утилитарни функции** - помощни методи без нужда от обект
- **Инициализация** - статичен конструктор за подготовка
- **Константи** - неизменяеми стойности
- **Производителност** - по-бърз достъп без създаване на обект