Ламбда изрази и функции — 12б, ООП

1. Увод

Ламбда изразите са анонимни функции, които ни позволяват да пишем по-кратък и изразителен код. Те са особено полезни за работа с делегати, събития и LINQ заявки.

2. Какво са ламбда изразите?

Ламбда изразите са:

- **Анонимни функции** - без име, директно използване
- **Кратък синтаксис** - по-малко код за същата функционалност
- **Изразителни** - ясно показват намерението
- **Функционални** - могат да се предават като параметри
- **Затваряния** - могат да използват променливи от външния обхват

3. Синтаксис на ламбда изразите

Основен синтаксис:

- **Параметри => израз** - за изрази
- **Параметри => { блок }** - за блокове от код

- **Тип параметри** - (int x, int y) => x + y
- **Автоматично определяне** - (x, y) => x + y
- **Един параметър** - x => x * 2

Основни примери с ламбда изрази:

```
// Делегат за математически операции
delegate int MathOperation(int x, int y);

// Използване на ламбда изрази
MathOperation add = (x, y) => x + y;
MathOperation multiply = (x, y) => x * y;
MathOperation subtract = (x, y) => x - y;

// Извикване
int result1 = add(5, 3);        // 8
int result2 = multiply(4, 6);   // 24
int result3 = subtract(10, 4);  // 6

Console.WriteLine($"Събиране: {result1}");
Console.WriteLine($"Умножение: {result2}");
Console.WriteLine($"Изваждане: {result3}");

// Ламбда изрази с блокове
MathOperation complexOperation = (x, y) => {
    int temp = x * 2;
    return temp + y;
};

int result4 = complexOperation(3, 5); // 11
Console.WriteLine($"Сложна операция: {result4}");

// Ламбда изрази без параметри
Func getMessage = () => "Здравей от ламбда израз!";
Console.WriteLine(getMessage());

// Ламбда изрази с един параметър
Func square = x => x * x;
Func getLength = s => s.Length;
```

```
Console.WriteLine($"Квадрат на 5: {square(5)}");
Console.WriteLine($"Дължина на 'Здравей': {getLength("Здравей")}");
```

4. Вградени делегати

Основни вградени делегати:

- **Action** - не връща стойност (void)
- **Func<T>** - връща стойност от тип T
- **Predicate<T>** - връща bool (за условия)
- **Comparison<T>** - за сравняване на два обекта

Примери с вградени делегати:

```
// Action делегати
Action printMessage = message => Console.WriteLine(message);
Action printSum = (x, y) => Console.WriteLine($"Сума: {x + y}");

printMessage("Здравей от Action!");
printSum(10, 20);

// Func делегати
Func add = (x, y) => x + y;
Func concatenate = (s1, s2) => s1 + " " + s2;
Func isEven = x => x % 2 == 0;

Console.WriteLine($"Събиране: {add(5, 3)}");
Console.WriteLine($"Свързване: {concatenate("Здравей", "Свят")}");
Console.WriteLine($"4 е четно: {isEven(4)}");

// Predicate делегати
Predicate isPositive = x => x > 0;
Predicate isNotEmpty = s => !string.IsNullOrEmpty(s);

Console.WriteLine($"5 е положително: {isPositive(5)}");
Console.WriteLine($"'Здравей' не е празно: {isNotEmpty("Здравей")}");
```

```
// Comparison делегати
Comparison compareNumbers = (x, y) => x.CompareTo(y);
Comparison compareStrings = (s1, s2) => string.Compare(s1, s2);

Console.WriteLine($"Сравнение на 5 и 3: {compareNumbers(5, 3)}");
Console.WriteLine($"Сравнение на 'a' и 'b': {compareStrings("a", "b")}");
```

5. Ламбда изрази с колекции

LINQ методи с ламбда изрази:

- **Where** - филтриране на елементи
- **Select** - трансформация на елементи
- **OrderBy** - сортиране
- **GroupBy** - групиране
- **Aggregate** - агрегация

Примери с LINQ и ламбда изрази:

```
// Данни за тестване
var numbers = new List { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var names = new List { "Иван", "Мария", "Петър", "Анна", "Георги" };
var students = new List {
    new Student("Иван", 20, 5.5),
    new Student("Мария", 19, 6.0),
    new Student("Петър", 21, 4.5),
    new Student("Анна", 18, 5.0),
    new Student("Георги", 22, 5.5)
};

// Where - филтриране
var evenNumbers = numbers.Where(x => x % 2 == 0);
var longNames = names.Where(name => name.Length > 4);
var excellentStudents = students.Where(s => s.Grade >= 5.5);
```

```
Console.WriteLine("Четни числа:");
foreach (var num in evenNumbers) {
    Console.WriteLine(num);
}

Console.WriteLine("\nДълги имена:");
foreach (var name in longNames) {
    Console.WriteLine(name);
}

Console.WriteLine("\nОтлични студенти:");
foreach (var student in excellentStudents) {
    Console.WriteLine(student);
}

// Select - трансформация
var squares = numbers.Select(x => x * x);
var upperNames = names.Select(name => name.ToUpper());
var studentInfo = students.Select(s => $"{s.Name} - {s.Grade}");

Console.WriteLine("\nКвадрати:");
foreach (var square in squares) {
    Console.WriteLine(square);
}

Console.WriteLine("\nГлавни букви:");
foreach (var name in upperNames) {
    Console.WriteLine(name);
}

Console.WriteLine("\nИнформация за студенти:");
foreach (var info in studentInfo) {
    Console.WriteLine(info);
}

// OrderBy - сортиране
var sortedNumbers = numbers.OrderBy(x => x);
var sortedNames = names.OrderBy(name => name.Length);
var sortedStudents = students.OrderBy(s => s.Grade).ThenBy(s => s.Name);

Console.WriteLine("\nСортирани числа:");
foreach (var num in sortedNumbers) {
    Console.WriteLine(num);
}

Console.WriteLine("\nСортирани по дължина:");
```

```
foreach (var name in sortedNames) {
    Console.WriteLine(name);
}

Console.WriteLine("\nСортирани студенти:");
foreach (var student in sortedStudents) {
    Console.WriteLine(student);
}

// GroupBy - групиране
var groupedByGrade = students.GroupBy(s => s.Grade);
var groupedByLength = names.GroupBy(name => name.Length);

Console.WriteLine("\nГрупирани по оценка:");
foreach (var group in groupedByGrade) {
    Console.WriteLine($"Оценка {group.Key}:");
    foreach (var student in group) {
        Console.WriteLine($"  {student.Name}");
    }
}

// Aggregate - агрегация
var sum = numbers.Aggregate((x, y) => x + y);
var product = numbers.Aggregate(1, (x, y) => x * y);
var concatenated = names.Aggregate((x, y) => x + ", " + y);

Console.WriteLine($"\nСума: {sum}");
Console.WriteLine($"Произведение: {product}");
Console.WriteLine($"Свързани имена: {concatenated}");
```

6. Затваряния (Closures)

Затварянията позволяват:

- **Достъп до външни променливи** - от обхвата на ламбда израза
- **Запазване на състояние** - между извикванията
- **Функционално програмиране** - създаване на функции
- **Конфигуриране на поведение** - динамично задаване на параметри

Примери с затваряния:

```
// Затваряне с външна променлива
int multiplier = 10;
Func multiplyByTen = x => x * multiplier;

Console.WriteLine($"5 * 10 = {multiplyByTen(5)}");

// Промяна на външната променлива
multiplier = 20;
Console.WriteLine($"5 * 20 = {multiplyByTen(5)}");

// Затваряне с множество променливи
string prefix = "Резултат: ";
int offset = 100;
Func formatResult = x => $"{prefix}{x + offset}";

Console.WriteLine(formatResult(50));

// Затваряне в цикъл
var functions = new List>();
for (int i = 0; i < 5; i++) {
    int captured = i; // Важно: копиране на стойността
    functions.Add(() => captured * 2);
}

Console.WriteLine("\nЗатваряния в цикъл:");
foreach (var func in functions) {
    Console.WriteLine(func());
}

// Затваряне за създаване на функции
Func> createAdder = x => y => x + y;
var addFive = createAdder(5);
var addTen = createAdder(10);

Console.WriteLine($"5 + 3 = {addFive(3)}");
Console.WriteLine($"10 + 7 = {addTen(7)}");

// Затваряне за конфигуриране
Func> createFilter = keyword =>
    text => text.ToLower().Contains(keyword.ToLower());

var containsHello = createFilter("hello");
```

```
var containsWorld = createFilter("world");

Console.WriteLine($"'Hello World' съдържа 'hello': {containsHello("Hello
World")}");
Console.WriteLine($"'Hello World' съдържа 'world': {containsWorld("Hello
World")}");
```

7. Пълен пример - Система за филтриране

Реален пример с ламбда изрази:

```
// Клас за продукт
class Product {
    public string Name { get; set; }
    public decimal Price { get; set; }
    public string Category { get; set; }
    public int Stock { get; set; }
    public double Rating { get; set; }

    public Product(string name, decimal price, string category, int stock,
double rating) {
        Name = name;
        Price = price;
        Category = category;
        Stock = stock;
        Rating = rating;
    }

    public override string ToString() {
        return $"{Name} - {Price:C} ({Category}) [Оценка: {Rating:F1},
Наличност: {Stock}]";
    }
}

// Клас за филтриране на продукти
class ProductFilter {
    private List products;

    public ProductFilter(List products) {
        this.products = products;
    }
```

```
    // Метод за филтриране с ламбда израз
    public List Filter(Func predicate) {
        return products.Where(predicate).ToList();
    }

    // Метод за сортиране с ламбда израз
    public List SortBy(Func keySelector) {
        return products.OrderBy(keySelector).ToList();
    }

    // Метод за групиране с ламбда израз
    public Dictionary> GroupBy(Func keySelector) {
        return products.GroupBy(keySelector).ToDictionary(g => g.Key, g =>
g.ToList());
    }

    // Метод за трансформация с ламбда израз
    public List Select(Func selector) {
        return products.Select(selector).ToList();
    }

    // Метод за агрегация с ламбда израз
    public TResult Aggregate(TResult seed, Func func) {
        return products.Aggregate(seed, func);
    }

    // Метод за търсене с ламбда израз
    public List Search(Func searchPredicate) {
        return products.Where(searchPredicate).ToList();
    }

    // Метод за статистика
    public void DisplayStatistics() {
        Console.WriteLine($"Общо продукти: {products.Count}");
        Console.WriteLine($"Средна цена: {products.Average(p =>
p.Price):C}");
        Console.WriteLine($"Най-висока цена: {products.Max(p =>
p.Price):C}");
        Console.WriteLine($"Най-ниска цена: {products.Min(p =>
p.Price):C}");
        Console.WriteLine($"Средна оценка: {products.Average(p =>
p.Rating):F1}");
    }
}

// Използване
```

```
var products = new List {
    new Product("Laptop", 1500, "Електроника", 10, 4.5),
    new Product("Книга", 25, "Книги", 50, 4.2),
    new Product("Телефон", 800, "Електроника", 15, 4.8),
    new Product("Молив", 2, "Канцеларски", 100, 3.5),
    new Product("Монитор", 400, "Електроника", 8, 4.3),
    new Product("Роман", 30, "Книги", 25, 4.7),
    new Product("Тетрадка", 5, "Канцеларски", 200, 4.0),
    new Product("Таблет", 600, "Електроника", 12, 4.6)
};

var filter = new ProductFilter(products);

Console.WriteLine("=== Система за филтриране на продукти ===");
filter.DisplayStatistics();

// Филтриране по различни критерии
Console.WriteLine("\n=== Филтриране по цена (под 100 лв) ===");
var cheapProducts = filter.Filter(p => p.Price < 100);
foreach (var product in cheapProducts) {
    Console.WriteLine(product);
}

Console.WriteLine("\n=== Филтриране по категория (Електроника) ===");
var electronics = filter.Filter(p => p.Category == "Електроника");
foreach (var product in electronics) {
    Console.WriteLine(product);
}

Console.WriteLine("\n=== Филтриране по оценка (над 4.5) ===");
var highRated = filter.Filter(p => p.Rating > 4.5);
foreach (var product in highRated) {
    Console.WriteLine(product);
}

Console.WriteLine("\n=== Филтриране по наличност (под 20) ===");
var lowStock = filter.Filter(p => p.Stock < 20);
foreach (var product in lowStock) {
    Console.WriteLine(product);
}

// Сортиране по различни критерии
Console.WriteLine("\n=== Сортиране по цена ===");
var sortedByPrice = filter.SortBy(p => p.Price);
foreach (var product in sortedByPrice) {
    Console.WriteLine(product);
}
```

```
Console.WriteLine("\n=== Сортиране по оценка (низходящо) ===");
var sortedByRating = filter.SortBy(p => -p.Rating);
foreach (var product in sortedByRating) {
    Console.WriteLine(product);
}

Console.WriteLine("\n=== Сортиране по име ===");
var sortedByName = filter.SortBy(p => p.Name);
foreach (var product in sortedByName) {
    Console.WriteLine(product);
}

// Групиране по категории
Console.WriteLine("\n=== Групиране по категории ===");
var groupedByCategory = filter.GroupBy(p => p.Category);
foreach (var group in groupedByCategory) {
    Console.WriteLine($"\n{group.Key}:");
    foreach (var product in group.Value) {
        Console.WriteLine($"  {product.Name} - {product.Price:C}");
    }
}

// Търсене с различни критерии
Console.WriteLine("\n=== Търсене по име (съдържа 'L') ===");
var searchResults = filter.Search(p => p.Name.ToLower().Contains("l"));
foreach (var product in searchResults) {
    Console.WriteLine(product);
}

Console.WriteLine("\n=== Търсене по цена и оценка ===");
var complexSearch = filter.Search(p => p.Price > 50 && p.Rating > 4.0);
foreach (var product in complexSearch) {
    Console.WriteLine(product);
}

// Агрегация
Console.WriteLine("\n=== Агрегация ===");
var totalValue = filter.Aggregate(0m, (sum, product) => sum +
product.Price);
var totalStock = filter.Aggregate(0, (sum, product) => sum +
product.Stock);
var averageRating = filter.Aggregate(0.0, (sum, product) => sum +
product.Rating) / products.Count;

Console.WriteLine($"Обща стойност на наличностите: {totalValue:C}");
Console.WriteLine($"Общо количество: {totalStock}");
```

```
Console.WriteLine($"Средна оценка: {averageRating:F1}");

// Трансформация
Console.WriteLine("\n=== Трансформация ===");
var productNames = filter.Select(p => p.Name);
var productPrices = filter.Select(p => p.Price);
var productInfo = filter.Select(p => $"{p.Name}: {p.Price:C}");

Console.WriteLine("Имена на продуктите:");
foreach (var name in productNames) {
    Console.WriteLine($"  {name}");
}

Console.WriteLine("\nЦени на продуктите:");
foreach (var price in productPrices) {
    Console.WriteLine($"  {price:C}");
}

Console.WriteLine("\nИнформация за продуктите:");
foreach (var info in productInfo) {
    Console.WriteLine($"  {info}");
}
```

8. Практически задачи

Задачи за упражнение:

- **Създай ламбда изрази** за математически операции
- **Имплементирай филтриране** на списък с числа
- **Направи сортиране** с различни критерии
- **Създай търсене** в колекция от обекти
- **Имплементирай агрегация** с ламбда изрази

9. Заключение

Ламбда изразите са мощна функционалност, която ни позволява да пишем по-кратък, изразителен и функционален код, особено полезен за работа с колекции и делегати.

Ключови принципи:

- **Кратък синтаксис** - по-малко код за същата функционалност
- **Функционално програмиране** - предаване на функции като параметри
- **Затваряния** - достъп до външни променливи
- **LINQ интеграция** - мощни операции с колекции
- **Читаемост** - по-ясен и изразителен код