Комуникация между обекти. Делегати и събития — 12б, ООП

1. Увод

Делегатите и събитията са мощни механизми за комуникация между обекти в C#. Те позволяват слабо свързани системи, където обектите могат да комуникират без да знаят директно един за друг.

2. Какво са делегатите?

Делегатите са:

- **Типове за функции** - референции към методи
- **Типобезопасни** - компилаторът проверява сигнатурата
- **Множествено извикване** - могат да съдържат множество методи
- **Анонимни методи** - могат да се създават inline
- **Ламбда изрази** - могат да се използват с делегати

3. Дефиниране и използване на делегати

Синтаксис на делегатите:

- **delegate** - ключова дума за дефиниране
- **Сигнатура** - тип на връщане и параметри

- **Инстанциране** - създаване на обект от делегат
- **Извикване** - изпълнение на методите

Основни примери с делегати:

```
// Дефиниране на делегат
delegate int MathOperation(int x, int y);
delegate void DisplayMessage(string message);
delegate bool ValidationRule(string input);

// Клас с методи за делегати
class MathHelper {
    public static int Add(int x, int y) {
        Console.WriteLine($"Събиране: {x} + {y}");
        return x + y;
    }

    public static int Multiply(int x, int y) {
        Console.WriteLine($"Умножение: {x} * {y}");
        return x * y;
    }

    public static int Subtract(int x, int y) {
        Console.WriteLine($"Изваждане: {x} - {y}");
        return x - y;
    }
}

class MessageHelper {
    public static void PrintToConsole(string message) {
        Console.WriteLine($"Конзола: {message}");
    }

    public static void PrintToFile(string message) {
        Console.WriteLine($"Файл: {message}");
    }

    public static void PrintToDatabase(string message) {
        Console.WriteLine($"База данни: {message}");
    }
}
```

```
class ValidationHelper {
    public static bool IsNotEmpty(string input) {
        return !string.IsNullOrEmpty(input);
    }

    public static bool IsEmail(string input) {
        return input.Contains("@") && input.Contains(".");
    }

    public static bool IsLongEnough(string input) {
        return input.Length >= 5;
    }
}

// Използване на делегати
MathOperation operation = MathHelper.Add;
int result = operation(5, 3);
Console.WriteLine($"Резултат: {result}");

// Промяна на делегата
operation = MathHelper.Multiply;
result = operation(4, 6);
Console.WriteLine($"Резултат: {result}");

// Множествено извикване
DisplayMessage display = MessageHelper.PrintToConsole;
display += MessageHelper.PrintToFile;
display += MessageHelper.PrintToDatabase;

display("Важно съобщение");

// Валидация с делегати
ValidationRule validator = ValidationHelper.IsNotEmpty;
validator += ValidationHelper.IsEmail;
validator += ValidationHelper.IsLongEnough;

string email = "user@example.com";
bool isValid = validator(email);
Console.WriteLine($"Email '{email}' е валиден: {isValid}");
```

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 printAction = message => Console.WriteLine($"Action: {message}");
Action mathAction = (x, y) => Console.WriteLine($"Сума: {x + y}");

printAction("Здравей от Action!");
mathAction(10, 20);

// Func делегати
Func addFunc = (x, y) => x + y;
Func concatFunc = (s1, s2) => s1 + " " + s2;
Func isEvenFunc = x => x % 2 == 0;

Console.WriteLine($"Събиране: {addFunc(5, 3)}");
Console.WriteLine($"Свързване: {concatFunc("Здравей", "Свят")}");
Console.WriteLine($"4 е четно: {isEvenFunc(4)}");

// Predicate делегати
Predicate isPositivePredicate = x => x > 0;
Predicate isNotEmptyPredicate = s => !string.IsNullOrEmpty(s);

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

```
// Комбиниране на делегати
Action combinedAction = printAction;
combinedAction += message => Console.WriteLine($"Допълнително:
{message.ToUpper()}");
combinedAction("Тест на комбиниране");
```

5. Събития (Events)

Събитията са:

- **Специализирани делегати** - за уведомления
- **Безопасни** - само класът може да ги извиква
- **Множествено абониране** - множество слушатели
- **Автоматично отписване** - при унищожаване на обекта
- **Стандартизирани** - EventHandler<T> pattern

Примери с събития:

```
// Клас за банкова сметка с събития
class BankAccount {
    private decimal balance;
    private string accountNumber;

    public event EventHandler BalanceChanged;
    public event EventHandler TransactionCompleted;
    public event EventHandler LowBalanceWarning;

    public decimal Balance {
        get { return balance; }
        private set {
            decimal oldBalance = balance;
            balance = value;

            // Извикване на събитието за промяна на баланса
```

```
            BalanceChanged?.Invoke(this, balance);

            // Проверка за ниско салдо
            if (balance < 100 && oldBalance >= 100) {
                LowBalanceWarning?.Invoke(this, balance);
            }
        }
    }

    public string AccountNumber {
        get { return accountNumber; }
    }

    public BankAccount(string accountNumber, decimal initialBalance = 0) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void Deposit(decimal amount) {
        if (amount > 0) {
            Balance += amount;
            TransactionCompleted?.Invoke(this, $"Депозит от {amount:C}");
        }
    }

    public bool Withdraw(decimal amount) {
        if (amount > 0 && Balance >= amount) {
            Balance -= amount;
            TransactionCompleted?.Invoke(this, $"Теглене от {amount:C}");
            return true;
        }
        return false;
    }
}

// Клас за логиране на транзакции
class TransactionLogger {
    private List transactions;

    public TransactionLogger() {
        transactions = new List();
    }

    public void OnTransactionCompleted(object sender, string transaction) {
        string logEntry = $"[{DateTime.Now:HH:mm:ss}] {transaction}";
        transactions.Add(logEntry);
        Console.WriteLine($"LOG: {logEntry}");
```

```
    }

    public void DisplayTransactions() {
        Console.WriteLine("\n=== История на транзакциите ===");
        foreach (var transaction in transactions) {
            Console.WriteLine(transaction);
        }
    }
}

// Клас за мониторинг на баланса
class BalanceMonitor {
    public void OnBalanceChanged(object sender, decimal newBalance) {
        Console.WriteLine($"Монитор: Балансът е променен на
{newBalance:C}");
    }

    public void OnLowBalanceWarning(object sender, decimal balance) {
        Console.WriteLine($"ПРЕДУПРЕЖДЕНИЕ: Ниско салдо! Текущ баланс:
{balance:C}");
    }
}

// Използване
var account = new BankAccount("BG123456789", 1000);
var logger = new TransactionLogger();
var monitor = new BalanceMonitor();

// Абониране за събития
account.BalanceChanged += monitor.OnBalanceChanged;
account.TransactionCompleted += logger.OnTransactionCompleted;
account.LowBalanceWarning += monitor.OnLowBalanceWarning;

// Извършване на транзакции
account.Deposit(500);
account.Withdraw(200);
account.Withdraw(1000); // Това ще предизвика предупреждение
account.Withdraw(200);

// Показване на историята
logger.DisplayTransactions();
```

6. Пълен пример - Система за уведомления

Реален пример с делегати и събития:

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

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

    public override string ToString() {
        return $"{Name} - {Price:C} (Наличност: {Stock})";
    }
}

// Клас за магазин с събития
class Store {
    private List products;
    private Dictionary sales;

    public event EventHandler ProductAdded;
    public event EventHandler ProductSold;
    public event EventHandler LowStockWarning;
    public event EventHandler CategoryAdded;

    public Store() {
        products = new List();
        sales = new Dictionary();
    }

    public void AddProduct(Product product) {
        products.Add(product);
        ProductAdded?.Invoke(this, product);

        if (!sales.ContainsKey(product.Category)) {
            sales[product.Category] = 0;
            CategoryAdded?.Invoke(this, product.Category);
```

```
        }
    }

    public bool SellProduct(string productName, int quantity) {
        var product = products.FirstOrDefault(p => p.Name == productName);
        if (product != null && product.Stock >= quantity) {
            product.Stock -= quantity;
            sales[product.Category] += quantity;

            ProductSold?.Invoke(this, product);

            if (product.Stock < 5) {
                LowStockWarning?.Invoke(this, product);
            }

            return true;
        }
        return false;
    }

    public List GetProducts() {
        return new List(products);
    }

    public Dictionary GetSales() {
        return new Dictionary(sales);
    }
}

// Клас за статистика
class StatisticsManager {
    private int totalProducts;
    private int totalSales;
    private Dictionary categorySales;

    public StatisticsManager() {
        totalProducts = 0;
        totalSales = 0;
        categorySales = new Dictionary();
    }

    public void OnProductAdded(object sender, Product product) {
        totalProducts++;
        Console.WriteLine($"Статистика: Добавен продукт. Общо:
{totalProducts}");
    }
```

```
    public void OnProductSold(object sender, Product product) {
        totalSales++;
        Console.WriteLine($"Статистика: Продаден продукт. Общо продажби:
{totalSales}");
    }

    public void OnCategoryAdded(object sender, string category) {
        Console.WriteLine($"Статистика: Добавена нова категория:
{category}");
    }

    public void DisplayStatistics() {
        Console.WriteLine("\n=== Статистика ===");
        Console.WriteLine($"Общо продукти: {totalProducts}");
        Console.WriteLine($"Общо продажби: {totalSales}");
    }
}

// Клас за инвентарно управление
class InventoryManager {
    private List lowStockProducts;

    public InventoryManager() {
        lowStockProducts = new List();
    }

    public void OnLowStockWarning(object sender, Product product) {
        if (!lowStockProducts.Contains(product)) {
            lowStockProducts.Add(product);
        }
        Console.WriteLine($"Инвентар: ПРЕДУПРЕЖДЕНИЕ за ниски наличности -
{product}");
    }

    public void OnProductSold(object sender, Product product) {
        if (product.Stock >= 5 && lowStockProducts.Contains(product)) {
            lowStockProducts.Remove(product);
            Console.WriteLine($"Инвентар: Продуктът {product.Name} вече не
е с ниски наличности");
        }
    }

    public void DisplayLowStockProducts() {
        Console.WriteLine("\n=== Продукти с ниски наличности ===");
        if (lowStockProducts.Count == 0) {
            Console.WriteLine("Няма продукти с ниски наличности");
        } else {
```

```
            foreach (var product in lowStockProducts) {
                Console.WriteLine(product);
            }
        }
    }
}

// Клас за отчети
class ReportGenerator {
    private List reports;

    public ReportGenerator() {
        reports = new List();
    }

    public void OnProductAdded(object sender, Product product) {
        string report = $"[{DateTime.Now:HH:mm:ss}] Добавен: {product.Name}
в категория {product.Category}";
        reports.Add(report);
    }

    public void OnProductSold(object sender, Product product) {
        string report = $"[{DateTime.Now:HH:mm:ss}] Продаден:
{product.Name} (останали: {product.Stock})";
        reports.Add(report);
    }

    public void OnCategoryAdded(object sender, string category) {
        string report = $"[{DateTime.Now:HH:mm:ss}] Нова категория:
{category}";
        reports.Add(report);
    }

    public void GenerateReport() {
        Console.WriteLine("\n=== Дневен отчет ===");
        foreach (var report in reports) {
            Console.WriteLine(report);
        }
    }
}

// Клас за уведомления
class NotificationService {
    public void OnProductAdded(object sender, Product product) {
        Console.WriteLine($"Уведомление: Нов продукт '{product.Name}' е
добавен в магазина!");
    }
```

```
    public void OnProductSold(object sender, Product product) {
        Console.WriteLine($"Уведомление: Продукт '{product.Name}' е
продаден!");
    }

    public void OnLowStockWarning(object sender, Product product) {
        Console.WriteLine($"Уведомление: Продукт '{product.Name}' има ниски
наличности ({product.Stock} броя)!");
    }

    public void OnCategoryAdded(object sender, string category) {
        Console.WriteLine($"Уведомление: Нова категория '{category}' е
добавена!");
    }
}

// Използване
var store = new Store();
var statistics = new StatisticsManager();
var inventory = new InventoryManager();
var reports = new ReportGenerator();
var notifications = new NotificationService();

// Абониране за събития
store.ProductAdded += statistics.OnProductAdded;
store.ProductAdded += reports.OnProductAdded;
store.ProductAdded += notifications.OnProductAdded;

store.ProductSold += statistics.OnProductSold;
store.ProductSold += reports.OnProductSold;
store.ProductSold += inventory.OnProductSold;
store.ProductSold += notifications.OnProductSold;

store.LowStockWarning += inventory.OnLowStockWarning;
store.LowStockWarning += notifications.OnLowStockWarning;

store.CategoryAdded += statistics.OnCategoryAdded;
store.CategoryAdded += reports.OnCategoryAdded;
store.CategoryAdded += notifications.OnCategoryAdded;

// Добавяне на продукти
Console.WriteLine("=== Добавяне на продукти ===");
store.AddProduct(new Product("Laptop", 1500, 10, "Електроника"));
store.AddProduct(new Product("Книга", 25, 50, "Книги"));
store.AddProduct(new Product("Молив", 2, 100, "Канцеларски"));
```

```
// Продажби
Console.WriteLine("\n=== Продажби ===");
store.SellProduct("Laptop", 2);
store.SellProduct("Книга", 10);
store.SellProduct("Молив", 50);
store.SellProduct("Laptop", 8); // Това ще предизвика предупреждение

// Показване на статистика
statistics.DisplayStatistics();
inventory.DisplayLowStockProducts();
reports.GenerateReport();

// Показване на всички продукти
Console.WriteLine("\n=== Всички продукти ===");
foreach (var product in store.GetProducts()) {
    Console.WriteLine(product);
}
```

7. Практически задачи

Задачи за упражнение:

- **Създай система за уведомления** с делегати и събития
- **Имплементирай система за логиране** с различни изходи
- **Направи система за мониторинг** на производителността
- **Създай система за кеширане** с уведомления за изтичане
- **Имплементирай система за валидация** с различни правила

8. Заключение

Делегатите и събитията са мощни инструменти за създаване на слабо свързани, гъвкави и разширяеми системи. Те позволяват обектите да

комуникират без да знаят директно един за друг.

Ключови принципи:

- **Слабо свързване** - обектите не знаят директно един за друг
- **Типобезопасност** - компилаторът проверява сигнатурата
- **Множествено извикване** - един делегат може да съдържа множество методи
- **Събития** - специализирани делегати за уведомления
- **Разширяемост** - лесно добавяне на нови слушатели