Работа с обекти. Итератори и компаратори — 12б, ООП

1. Увод

Работата с обекти включва множество техники за манипулиране, сравняване и итериране. Итераторите и компараторите са ключови инструменти за ефективна работа с колекции от обекти.

2. Итератори (Iterators)

Итераторите позволяват:

- **Последователен достъп** - до елементите на колекция
- **foreach цикли** - автоматично итериране
- **yield return** - лениво генериране на стойности
- **Кастомна логика** - специфично поведение при итериране

Основен пример с итератор:

```
// Клас с итератор
class NumberSequence {
    private int start;
    private int end;
    private int step;

    public NumberSequence(int start, int end, int step = 1) {
```

```
        this.start = start;
        this.end = end;
        this.step = step;
    }

    // Итератор метод
    public IEnumerator GetEnumerator() {
        for (int i = start; i <= end; i += step) {
            yield return i; // yield return за лениво генериране
        }
    }

    // Метод за обратно итериране
    public IEnumerable GetReverseEnumerator() {
        for (int i = end; i >= start; i -= step) {
            yield return i;
        }
    }

    // Метод за филтриране
    public IEnumerable GetEvenNumbers() {
        for (int i = start; i <= end; i += step) {
            if (i % 2 == 0) {
                yield return i;
            }
        }
    }

    // Метод за трансформация
    public IEnumerable GetFormattedNumbers() {
        for (int i = start; i <= end; i += step) {
            yield return $"Число: {i}";
        }
    }
}

// Използване
var sequence = new NumberSequence(1, 10, 2);

Console.WriteLine("Обикновено итериране:");
foreach (int number in sequence) {
    Console.WriteLine(number);
}

Console.WriteLine("\nОбратно итериране:");
foreach (int number in sequence.GetReverseEnumerator()) {
    Console.WriteLine(number);
```

```
}

Console.WriteLine("\nЧетни числа:");
foreach (int number in sequence.GetEvenNumbers()) {
    Console.WriteLine(number);
}

Console.WriteLine("\nФорматирани числа:");
foreach (string formatted in sequence.GetFormattedNumbers()) {
    Console.WriteLine(formatted);
}
```

3. Компаратори (Comparators)

Видове компаратори:

- **IComparable<T>** - за сравняване на обекти
- **IComparer<T>** - за външно сравняване
- **Comparison<T>** - делегат за сравняване
- **Custom Comparers** - специфична логика за сравняване

Пример с IComparable:

```
// Клас, който имплементира IComparable
class Student : IComparable {
    public string Name { get; set; }
    public int Age { get; set; }
    public double Grade { get; set; }

    public Student(string name, int age, double grade) {
        Name = name;
        Age = age;
        Grade = grade;
    }

    // Имплементация на IComparable
```

```
    public int CompareTo(Student other) {
        if (other == null) return 1;

        // Първо сравняваме по оценка (низходящо)
        int gradeComparison = other.Grade.CompareTo(this.Grade);
        if (gradeComparison != 0) return gradeComparison;

        // След това по име (възходящо)
        int nameComparison = this.Name.CompareTo(other.Name);
        if (nameComparison != 0) return nameComparison;

        // Накрая по възраст (възходящо)
        return this.Age.CompareTo(other.Age);
    }

    public override string ToString() {
        return $"{Name} ({Age}г, {Grade:F1})";
    }
}

// Използване
var students = new List {
    new Student("Иван", 20, 5.5),
    new Student("Мария", 19, 6.0),
    new Student("Петър", 21, 5.5),
    new Student("Анна", 18, 5.0),
    new Student("Иван", 20, 4.5)
};

Console.WriteLine("Преди сортиране:");
foreach (var student in students) {
    Console.WriteLine(student);
}

students.Sort(); // Използва IComparable

Console.WriteLine("\nСлед сортиране:");
foreach (var student in students) {
    Console.WriteLine(student);
}
```

4. IComparer интерфейс

Предимства на IComparer:

- **Множествено сравняване** - различни критерии
- **Външна логика** - отделни класове за сравняване
- **Гъвкавост** - лесно смяна на критериите
- **Преизползване** - една логика за различни типове

Пример с IComparer:

```
// Компаратор за сравняване по име
class StudentNameComparer : IComparer {
    public int Compare(Student x, Student y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return string.Compare(x.Name, y.Name,
StringComparison.OrdinalIgnoreCase);
    }
}

// Компаратор за сравняване по възраст
class StudentAgeComparer : IComparer {
    public int Compare(Student x, Student y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return x.Age.CompareTo(y.Age);
    }
}

// Компаратор за сравняване по оценка
class StudentGradeComparer : IComparer {
    public int Compare(Student x, Student y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;
```

```
        return y.Grade.CompareTo(x.Grade); // Низходящо
    }
}

// Компаратор за сравняване по множество критерии
class StudentMultiComparer : IComparer {
    private string primaryCriteria;
    private string secondaryCriteria;

    public StudentMultiComparer(string primary, string secondary) {
        primaryCriteria = primary;
        secondaryCriteria = secondary;
    }

    public int Compare(Student x, Student y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        int primaryComparison = CompareByCriteria(x, y, primaryCriteria);
        if (primaryComparison != 0) return primaryComparison;

        return CompareByCriteria(x, y, secondaryCriteria);
    }

    private int CompareByCriteria(Student x, Student y, string criteria) {
        switch (criteria.ToLower()) {
            case "name":
                return string.Compare(x.Name, y.Name,
StringComparison.OrdinalIgnoreCase);
            case "age":
                return x.Age.CompareTo(y.Age);
            case "grade":
                return y.Grade.CompareTo(x.Grade);
            default:
                return 0;
        }
    }
}

// Използване
var students = new List {
    new Student("Иван", 20, 5.5),
    new Student("Мария", 19, 6.0),
    new Student("Петър", 21, 5.5),
    new Student("Анна", 18, 5.0)
};
```

```
Console.WriteLine("Сортиране по име:");
students.Sort(new StudentNameComparer());
foreach (var student in students) {
    Console.WriteLine(student);
}

Console.WriteLine("\nСортиране по възраст:");
students.Sort(new StudentAgeComparer());
foreach (var student in students) {
    Console.WriteLine(student);
}

Console.WriteLine("\nСортиране по оценка:");
students.Sort(new StudentGradeComparer());
foreach (var student in students) {
    Console.WriteLine(student);
}

Console.WriteLine("\nСортиране по оценка, след това по име:");
students.Sort(new StudentMultiComparer("grade", "name"));
foreach (var student in students) {
    Console.WriteLine(student);
}
```

5. Пълен пример - Система за управление на библиотека

Реален пример с итератори и компаратори:

```
// Клас за книга
class Book : IComparable {
    public string Title { get; set; }
    public string Author { get; set; }
    public int Year { get; set; }
    public string ISBN { get; set; }
    public decimal Price { get; set; }
    public int Pages { get; set; }
    public string Genre { get; set; }
    public bool IsAvailable { get; set; }
```

```
    public Book(string title, string author, int year, string isbn,
                decimal price, int pages, string genre) {
        Title = title;
        Author = author;
        Year = year;
        ISBN = isbn;
        Price = price;
        Pages = pages;
        Genre = genre;
        IsAvailable = true;
    }

    public int CompareTo(Book other) {
        if (other == null) return 1;

        // Първо по заглавие
        int titleComparison = string.Compare(Title, other.Title,
StringComparison.OrdinalIgnoreCase);
        if (titleComparison != 0) return titleComparison;

        // След това по автор
        return string.Compare(Author, other.Author,
StringComparison.OrdinalIgnoreCase);
    }

    public override string ToString() {
        return $"{Title} от {Author} ({Year}) - {Price:C}";
    }
}

// Клас за библиотека с итератори
class Library : IEnumerable {
    private List books;
    private Dictionary> booksByGenre;
    private Dictionary> booksByAuthor;

    public Library() {
        books = new List();
        booksByGenre = new Dictionary>();
        booksByAuthor = new Dictionary>();
    }

    public void AddBook(Book book) {
        books.Add(book);

        // Индексиране по жанр
```

```
        if (!booksByGenre.ContainsKey(book.Genre)) {
            booksByGenre[book.Genre] = new List();
        }
        booksByGenre[book.Genre].Add(book);

        // Индексиране по автор
        if (!booksByAuthor.ContainsKey(book.Author)) {
            booksByAuthor[book.Author] = new List();
        }
        booksByAuthor[book.Author].Add(book);
    }

    public void RemoveBook(Book book) {
        books.Remove(book);
        booksByGenre[book.Genre]?.Remove(book);
        booksByAuthor[book.Author]?.Remove(book);
    }

    // Основен итератор
    public IEnumerator GetEnumerator() {
        foreach (var book in books) {
            yield return book;
        }
    }

    System.Collections.IEnumerator
System.Collections.IEnumerable.GetEnumerator() {
        return GetEnumerator();
    }

    // Итератор за налични книги
    public IEnumerable GetAvailableBooks() {
        foreach (var book in books) {
            if (book.IsAvailable) {
                yield return book;
            }
        }
    }

    // Итератор за книги по жанр
    public IEnumerable GetBooksByGenre(string genre) {
        if (booksByGenre.ContainsKey(genre)) {
            foreach (var book in booksByGenre[genre]) {
                yield return book;
            }
        }
    }
```

```
    // Итератор за книги по автор
    public IEnumerable GetBooksByAuthor(string author) {
        if (booksByAuthor.ContainsKey(author)) {
            foreach (var book in booksByAuthor[author]) {
                yield return book;
            }
        }
    }

    // Итератор за книги в ценови диапазон
    public IEnumerable GetBooksInPriceRange(decimal minPrice, decimal
maxPrice) {
        foreach (var book in books) {
            if (book.Price >= minPrice && book.Price <= maxPrice) {
                yield return book;
            }
        }
    }

    // Итератор за книги по година
    public IEnumerable GetBooksByYear(int year) {
        foreach (var book in books) {
            if (book.Year == year) {
                yield return book;
            }
        }
    }

    // Итератор за търсене по заглавие
    public IEnumerable SearchByTitle(string searchTerm) {
        foreach (var book in books) {
            if (book.Title.ToLower().Contains(searchTerm.ToLower())) {
                yield return book;
            }
        }
    }

    // Итератор за търсене по автор
    public IEnumerable SearchByAuthor(string searchTerm) {
        foreach (var book in books) {
            if (book.Author.ToLower().Contains(searchTerm.ToLower())) {
                yield return book;
            }
        }
    }
```

```
    // Метод за сортиране
    public void SortBooks(IComparer comparer = null) {
        if (comparer != null) {
            books.Sort(comparer);
        } else {
            books.Sort(); // Използва IComparable
        }
    }

    // Метод за показване на статистика
    public void DisplayStatistics() {
        Console.WriteLine($"Общо книги: {books.Count}");
        Console.WriteLine($"Налични книги: {GetAvailableBooks().Count()}");
        Console.WriteLine($"Жанрове: {booksByGenre.Count}");
        Console.WriteLine($"Автори: {booksByAuthor.Count}");

        Console.WriteLine("\nКниги по жанр:");
        foreach (var genre in booksByGenre.Keys) {
            Console.WriteLine($"  {genre}: {booksByGenre[genre].Count}
книги");
        }
    }
}

// Компаратори за книги
class BookTitleComparer : IComparer {
    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return string.Compare(x.Title, y.Title,
StringComparison.OrdinalIgnoreCase);
    }
}

class BookAuthorComparer : IComparer {
    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return string.Compare(x.Author, y.Author,
StringComparison.OrdinalIgnoreCase);
    }
}
```

```
class BookYearComparer : IComparer {
    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return y.Year.CompareTo(x.Year); // Низходящо
    }
}

class BookPriceComparer : IComparer {
    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return x.Price.CompareTo(y.Price); // Възходящо
    }
}

class BookPagesComparer : IComparer {
    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return y.Pages.CompareTo(x.Pages); // Низходящо
    }
}

// Компаратор за множество критерии
class BookMultiComparer : IComparer {
    private string[] criteria;

    public BookMultiComparer(params string[] criteria) {
        this.criteria = criteria;
    }

    public int Compare(Book x, Book y) {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        foreach (var criterion in criteria) {
            int comparison = CompareByCriterion(x, y, criterion);
            if (comparison != 0) return comparison;
        }
```

```
        return 0;
    }

    private int CompareByCriterion(Book x, Book y, string criterion) {
        switch (criterion.ToLower()) {
            case "title":
                return string.Compare(x.Title, y.Title,
StringComparison.OrdinalIgnoreCase);
            case "author":
                return string.Compare(x.Author, y.Author,
StringComparison.OrdinalIgnoreCase);
            case "year":
                return y.Year.CompareTo(x.Year);
            case "price":
                return x.Price.CompareTo(y.Price);
            case "pages":
                return y.Pages.CompareTo(x.Pages);
            case "genre":
                return string.Compare(x.Genre, y.Genre,
StringComparison.OrdinalIgnoreCase);
            default:
                return 0;
        }
    }
}

// Използване
var library = new Library();

// Добавяне на книги
library.AddBook(new Book("C# Programming", "Иван Петров", 2023, "978-
1234567890", 50, 400, "Програмиране"));
library.AddBook(new Book("OOP Principles", "Мария Георгиева", 2022, "978-
0987654321", 45, 350, "Програмиране"));
library.AddBook(new Book("Database Design", "Петър Стоянов", 2023, "978-
1122334455", 55, 300, "Бази данни"));
library.AddBook(new Book("Web Development", "Анна Димитрова", 2022, "978-
5566778899", 60, 450, "Web"));
library.AddBook(new Book("Algorithms", "Иван Петров", 2021, "978-
9988776655", 40, 500, "Алгоритми"));

Console.WriteLine("=== Библиотечна система ===");
library.DisplayStatistics();

Console.WriteLine("\n=== Всички книги ===");
foreach (var book in library) {
```

```
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Книги по жанр 'Програмиране' ===");
foreach (var book in library.GetBooksByGenre("Програмиране")) {
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Книги от Иван Петров ===");
foreach (var book in library.GetBooksByAuthor("Иван Петров")) {
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Книги в ценови диапазон 40-50 лв ===");
foreach (var book in library.GetBooksInPriceRange(40, 50)) {
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Търсене по заглавие 'C#' ===");
foreach (var book in library.SearchByTitle("C#")) {
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Сортиране по заглавие ===");
library.SortBooks(new BookTitleComparer());
foreach (var book in library) {
    Console.WriteLine(book);
}

Console.WriteLine("\n=== Сортиране по автор, след това по заглавие ===");
library.SortBooks(new BookMultiComparer("author", "title"));
foreach (var book in library) {
    Console.WriteLine(book);
}
```

6. Практически задачи

Задачи за упражнение:

- **Създай клас Product** с итератор за различни критерии
- **Имплементирай IComparable** за сравняване на продукти

- **Направи компаратори** за сортиране по цена, име, категория
- **Създай итератор** за филтриране на продукти
- **Имплементирай търсене** с различни критерии

7. Заключение

Итераторите и компараторите са мощни инструменти за работа с обекти, които ни позволяват да създаваме гъвкави и ефективни системи за манипулиране на данни.

Ключови принципи:

- **Итератори** - последователен достъп до елементи
- **Компаратори** - различни критерии за сравняване
- **yield return** - лениво генериране на стойности
- **Гъвкавост** - лесно добавяне на нови критерии
- **Преизползване** - една логика за различни типове