Отчёт по лабораторной работе №6

Задание

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».

3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.

4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

5. Создать коллекцию класса List. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы: • public void Push(T element) – добавление в стек; • public T Pop() – чтение с удалением из стека.

8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

Текст программы

```
using System;
using System.Collections;
using System.Collections.Generic;

// Абстрактный класс Figure
public abstract class Figure : IComparable<Figure>
{
```

```
    public abstract double GetArea();

    public int CompareTo(Figure other)
    {
        if (other == null) return 1;
        return GetArea().CompareTo(other.GetArea());
    }

    public override string ToString()
    {
        return $"{GetType().Name} (Площадь = {GetArea():F2})";
    }
}
class FigureComparer : IComparer
{
    public int Compare(object x, object y)
    {
        if (x is Figure figure1 && y is Figure figure2)
        {
            return figure1.CompareTo(figure2);
        }

        throw new ArgumentException("Objects are not of type Figure");
    }
}

// Реализация классов фигур
public class Rectangle : Figure
{
    public double Width { get; set; }
    public double Height { get; set; }

    public Rectangle(double width, double height)
    {
```

```
        Width = width;
        Height = height;
    }

    public override double GetArea()
    {
        return Width * Height;
    }
}

public class Square : Figure
{
    public double Side { get; set; }

    public Square(double side)
    {
        Side = side;
    }

    public override double GetArea()
    {
        return Side * Side;
    }
}

public class Circle : Figure
{
    public double Radius { get; set; }

    public Circle(double radius)
    {
        Radius = radius;
    }
```

```
    public override double GetArea()
    {
        return Math.PI * Radius * Radius;
    }
}

// Реализация разреженной матрицы
public class SparseMatrix<T>
{
    private readonly Dictionary<(int x, int y, int z), T> _matrix = new();

    public T this[int x, int y, int z]
    {
        get => _matrix.TryGetValue((x, y, z), out var value) ? value : default;
        set => _matrix[(x, y, z)] = value;
    }

    public override string ToString()
    {
        var result = "";
        foreach (var key in _matrix.Keys)
        {
            result += $"[{key.x}, {key.y}, {key.z}] = {_matrix[key]}\n";
        }
        return result;
    }
}

public class SimpleList<T>
{
    protected class Node
    {
        public T Data { get; set; }
        public Node Next { get; set; }
```

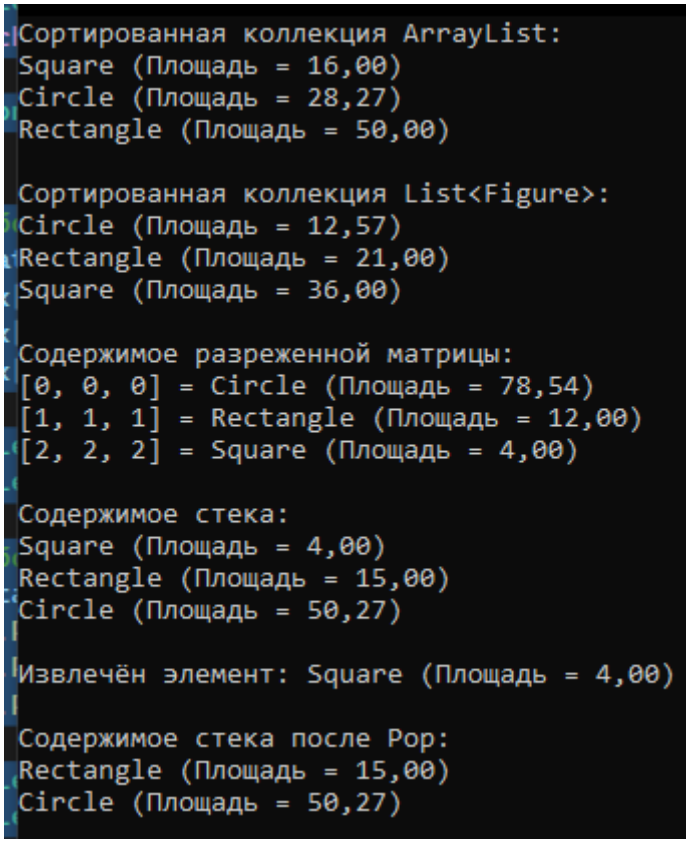
```
    }
    protected Node Head;
    public bool IsEmpty() => Head == null;
    public override string ToString()
    {
        var result = "";
        var current = Head;
        while (current != null)
        {
            result += $"{current.Data}\n";
            current = current.Next;
        }
        return result;
    }
}


public class SimpleStack<T> : SimpleList<T>
{
    // Добавление элемента в стек
    public void Push(T element)
    {
        var newNode = new Node { Data = element, Next = Head };
        Head = newNode;
    }

    // Удаление элемента из стека
    public T Pop()
    {
        if (IsEmpty()) throw new InvalidOperationException("Stack is empty");

        var value = Head.Data;
        Head = Head.Next; // Убираем первый элемент
        return value;
```

```
    }

    // Проверка на пустоту через базовый класс
    public override string ToString()
    {
        return base.ToString();
    }
}

// Основная программа
class Program
{
    static void Main(string[] args)
    {
        // Работа с ArrayList
        var figuresArrayList = new ArrayList
        {
            new Rectangle(10, 5),
            new Square(4),
            new Circle(3)
        };

        figuresArrayList.Sort(new FigureComparer());

        Console.WriteLine("Сортированная коллекция ArrayList:");
        foreach (Figure figure in figuresArrayList)
        {
            Console.WriteLine(figure);
        }

        // Работа с List<Figure>
        var figuresList = new List<Figure>
        {
            new Rectangle(7, 3),
```

```
            new Square(6),
            new Circle(2)
        };

        figuresList.Sort();

        Console.WriteLine("\nСортированная коллекция List<Figure>:");
        foreach (var figure in figuresList)
        {
            Console.WriteLine(figure);
        }

        // Работа с разреженной матрицей
        var matrix = new SparseMatrix<Figure>();
        matrix[0, 0, 0] = new Circle(5);
        matrix[1, 1, 1] = new Rectangle(4, 3);
        matrix[2, 2, 2] = new Square(2);

        Console.WriteLine("\nСодержимое разреженной матрицы:");
        Console.WriteLine(matrix);

        // Работа с SimpleStack
        var stack = new SimpleStack<Figure>();
        stack.Push(new Circle(4));
        stack.Push(new Rectangle(5, 3));
        stack.Push(new Square(2));

        Console.WriteLine("Содержимое стека:");
        Console.WriteLine(stack);

        Console.WriteLine("Извлечён элемент: " + stack.Pop());
        Console.WriteLine("\nСодержимое стека после Pop:");
        Console.WriteLine(stack);
    }
```

```
}
```

Пример выполнения программы



Вывод

Удалось выполнить поставленную задачу