Московский государственный технический университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по лабораторной работе №3

Выполнил:
Студент группы ИУ5-31Б
Куртинец Роман

Проверил:
Гапанюк Ю. Е.

2025 г.

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<Figure>. 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.

Листинг программы:

Circle.cs

```
class Circle : GeometrFigure
{
    public override double Radius { get; set; }

    public Circle(double rad = 0)
    {
        Check(rad);
        Radius = rad;
    }

    public override double Area() => Math.PI * Radius * Radius;
```

```
    public override string ToString() => $"Круг: Радиус = {Radius}, площадь =
{Area()}";
}
```

Rect.cs

```
class Rect : GeometrFigure
{

    public override double Length { get; set; }

    public override double Width { get; set; }

    public Rect(double length = 0, double width = 0)
    {
        Check(length);  Check(width);
        Length = length;
        Width = width;
    }

    public override double Area() => Length * Width;

    public override string ToString() => $"Прямоугольник: Длина = {Length},
высота = {Width}, площадь = {Area()}";

    public override void Print() => System.Console.WriteLine(ToString());
}
```

Square.cs

```
class Square : Rect
{
    public Square(double side = 0) : base(side, side) { }
    public override string ToString() => $"Квадрат: Сторона = {Length}, площадь
= {Area()}";
}
```

GeomFigure.cs

```
public abstract class GeometrFigure : IPrint
{
    public abstract double Area();
    public override string ToString() => "Такой фигуры нет в программе";

    public virtual double Length
    {
```

```
        get => 0;
        set { }
    }

    public virtual double Width
    {
        get => 0;
        set { }
    }

    public virtual double Radius
    {
        get => 0;
        set { }
    }

    public virtual void Print() => System.Console.WriteLine(ToString());

    protected void Check(double val) { if (val < 0) throw new
ArgumentException("Значение не может быть меньше 0"); }
}
```

SparseMatrixElement.cs

```
public class SparseMatrixElement<T> : IComparable<SparseMatrixElement<T>>
{
    public int xCoord { get; set; }
    public int yCoord { get; set; }
    public int zCoord { get; set; }
    public T value { get; set; }

    public SparseMatrixElement(int coordX, int coordY, int coordZ, T value)
    {
        xCoord = coordX;
        yCoord = coordY;
        zCoord = coordZ;
        this.value = value;
    }

    public int CompareTo(SparseMatrixElement<T> other)
    {
        if (other == null) return 1;
        if (xCoord != other.xCoord) return xCoord.CompareTo(other.xCoord);
        if (yCoord != other.yCoord) return yCoord.CompareTo(other.yCoord);
        return zCoord.CompareTo(other.zCoord);
    }
}
```

SparseTensor.cs

```
public class SparseTensor<T>
{
    private List<SparseMatrixElement<T>> data = new
List<SparseMatrixElement<T>>();
    private int sizeX, sizeY, sizeZ;

    public SparseTensor(int sizeX, int sizeY, int sizeZ)
    {
        if (sizeX <= 0 || sizeY <= 0 || sizeZ <= 0)
            throw new ArgumentException("Размеры должны быть положительными");
        this.sizeX = sizeX;
        this.sizeY = sizeY;
        this.sizeZ = sizeZ;
    }

    public void AddElem(int x, int y, int z, T value)
    {
        if (x < 0 || y < 0 || z < 0 || x >= sizeX || y >= sizeY || z >= sizeZ)
            throw new ArgumentException("Введены неверные координаты");
        var newElem = new SparseMatrixElement<T>(x, y, z, value);
        var index = data.BinarySearch(newElem);

        if (index >= 0) // нашли элемент
        {
            if (value.Equals(default(T)))
            {
                data.RemoveAt(index);
            }
            else
            {
                data.RemoveAt(index);
                data.Insert(index, newElem);
            }
        }
        else // не нашли элемент
        {
            if (!value.Equals(default(T)))
            {
                int insertIndex = ~index;
                data.Insert(insertIndex, newElem);
            }
        }
    }

    public T Get(int x, int y, int z)
    {
        if (x < 0 || y < 0 || z < 0 || x >= sizeX || y >= sizeY || z >= sizeZ)
            throw new ArgumentException("Введены неверные координаты");
```

```
        var searchElem = new SparseMatrixElement<T>(x, y, z, default(T));
        int index = data.BinarySearch(searchElem);
        return index >= 0 ? data[index].value : default(T);
    }

    public int NonZeroElems => data.Count;

    public void Print()
    {
        System.Console.WriteLine($"Sparse Tensor {sizeX}, {sizeY}, {sizeZ},
ненулевых элементов {NonZeroElems}");
        foreach (var elem in data)
        {
            System.Console.WriteLine($"[{elem.xCoord}, {elem.yCoord},
{elem.zCoord}] = {elem.value}");
        }
        if (data.Count == 0)
        {
            System.Console.WriteLine("Все элементы нулевые");
        }
    }

    public void FillMatrix(T area)
    {
        int x, y, z;
        System.Console.WriteLine("Введите координаты матрицы, куда вставим
площадь");
        (x, y, z) = (Convert.ToInt32(Console.ReadLine()),
Convert.ToInt32(Console.ReadLine()), Convert.ToInt32(Console.ReadLine()));
        AddElem(x, y, z, area);
    }
}
```

SimpleList.cs

```
public class SimpleList<T> : IPrint
{
    protected class Node
    {
        public T value { get; set; }
        public Node next { get; set; }

        public Node(T value)
        {
            this.value = value;
            next = null;
        }
    }
```

```
protected Node head;
protected int len;

public SimpleList()
{
    head = null;
    len = 0;
}

public virtual void Add(T value)
{
    Node newNode = new Node(value);
    if (head == null)
    {
        head = newNode;
    }
    else
    {
        Node curr = head;
        while (curr.next != null)
        {
            curr = curr.next;
        }
        curr.next = newNode;
    }
    ++len;
}

public bool Remove(T delValue)
{
    if (head == null) return false;

    if (head.value.Equals(delValue))
    {
        head = head.next;
        --len;
        return true;
    }

    Node curr = head;
    while (curr.next != null)
    {
        if (curr.next.value.Equals(delValue))
        {
            curr.next = curr.next.next;
            --len;
            return true;
        }
        curr = curr.next;
```

```
        }
        return false;
    }

    public T Get(int index)
    {
        if (index < 0 || index >= len) throw new IndexOutOfRangeException();

        Node curr = head;
        for (int i = 0; i < index; i++)
        {
            curr = curr.next;
        }
        return curr.value;
    }

    public void Print()
    {
        Node curr = head;
        System.Console.Write($"Stack: ");
        while (curr != null)
        {
            System.Console.Write($"{curr.value} ");
            curr = curr.next;
        }
        System.Console.Write('\n');
    }
}
```

SimpleStack.cs

```
public class SimpleStack<T> : SimpleList<T>
{
    public void Push(T value)
    {
        Node newNode = new Node(value);
        newNode.next = head;
        head = newNode;
        ++len;
    }

    public T Pop()
    {
        if (head == null) throw new InvalidOperationException("Стек пуст");

        T delValue = head.value;
        head = head.next;
        --len;
```

```
        return delValue;
    }

    public override void Add(T value)
    {
        Push(value);
    }
}
```

Program.cs

```
using System;

class Program
{
    static void Main()
    {
        string next = "y";
        while (next == "y")
        {
            System.Console.WriteLine("Выберите фигуру для подсчёта площади: 1 - 
Прямоугольник, 2 - Квадрат, 3 - Круг");
            if (int.TryParse(Console.ReadLine(), out int choose))
            {
                switch (choose)
                {
                    case 1:
                        System.Console.Write("Длина - ");
                        double length = Convert.ToDouble(Console.ReadLine());
                        System.Console.Write("Высота - ");
                        double width = Convert.ToDouble(Console.ReadLine());
                        Rect rect = new Rect(length, width);
                        rect.Print();
                        break;
                    case 2:
                        System.Console.Write("Сторона - ");
                        double side = Convert.ToDouble(Console.ReadLine());
                        Square square = new Square(side);
                        square.Print();
                        break;
                    case 3:
                        System.Console.Write("Радиус - ");
                        double radius = Convert.ToDouble(Console.ReadLine());
                        Circle circle = new Circle(radius);
                        circle.Print();
                        break;
                }
            }
```

```
            System.Console.WriteLine("Повторим? (y/n)");
            next = Console.ReadLine();
        }
    }
}
```

Результат выполнения:

```
Выберите фигуру для подсчёта площади: 1 - Прямоугольник, 2 - Квадрат, 3 - Круг
1
Длина - 234
Высота - 332.2
Прямоугольник: Длина = 234, высота = 332.2, площадь = 77734.8
Введите координаты матрицы, куда вставим площадь
1
1
1
Повторим? (y/n)
y
Выберите фигуру для подсчёта площади: 1 - Прямоугольник, 2 - Квадрат, 3 - Круг
3
Радиус - 345.321
Круг: Радиус = 345.321, площадь = 374624.22066321736
Введите координаты матрицы, куда вставим площадь
1
2
3
Повторим? (y/n)
y
Выберите фигуру для подсчёта площади: 1 - Прямоугольник, 2 - Квадрат, 3 - Круг
2
Сторона - 2
Квадрат: Сторона = 2, площадь = 4
Введите координаты матрицы, куда вставим площадь
2
4
1
Повторим? (y/n)
y
Выберите фигуру для подсчёта площади: 1 - Прямоугольник, 2 - Квадрат, 3 - Круг
3
Радиус - 3453.12312
Круг: Радиус = 3453.12312, площадь = 37460537.04091879
Введите координаты матрицы, куда вставим площадь
3
3
3
Повторим? (y/n)
```

```
n
Отсортированная коллекция ArrayList фигур:
Квадрат: Сторона = 2, площадь = 4
Прямоугольник: Длина = 234, высота = 332.2, площадь = 77734.8
Круг: Радиус = 345.321, площадь = 374624.22066321736
Круг: Радиус = 3453.12312, площадь = 37460537.04091879
Отсортированная коллекция List<Figure> фигур:
Квадрат: Сторона = 2, площадь = 4
Прямоугольник: Длина = 234, высота = 332.2, площадь = 77734.8
Круг: Радиус = 345.321, площадь = 374624.22066321736
Круг: Радиус = 3453.12312, площадь = 37460537.04091879
Sparse Tensor 5, 5, 5, ненулевых элементов 4
[1, 1, 1] = 77734.8
[1, 2, 3] = 374624.22066321736
[2, 4, 1] = 4
[3, 3, 3] = 37460537.04091879
Stack: 37460537.04091879 4 374624.22066321736 77734.8
```