#### Московский государственный технический университет имени Н.Э. Баумана

## Лабораторная работа по дисциплине БКИТ №3

Выполнила: Ершова А. ИУ5-32

### I. Описание задания

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.

### II. Код программы

﻿using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
  
namespace lab3  
{  
    class MainClass  
    {  
        static double InputValue(string prompt)  
        {  
            double a = 0;  
            do  
                Console.Write(prompt);  
            while (!double.TryParse(Console.ReadLine(), out a));  
            return a;  
        }  
  
  
        public static void Main(string[] args)  
        {  
            Console.WriteLine("Введите параметры прямоугольника");  
            double width = InputValue("Введите ширину прямоугольника:");  
            double height = InputValue("Введите высоту прямоугольника:");  
            Rectangle rect = new Rectangle(width, height);  
            rect.Print();  
            Console.WriteLine("Введите параметры квадрата");  
            double length = InputValue("Введите сторону квадрата:");  
            Square sq = new Square(length);  
            sq.Print();  
            Console.WriteLine("Введите параметры окружности");  
            double radius = InputValue("Введите радиус круга:");  
            Circle circ = new Circle(radius);  
            circ.Print();  
            Console.WriteLine("Работаем с ArrayList");  
            ArrayList figureArray = new ArrayList();  
            figureArray.Add(rect);  
            figureArray.Add(sq);  
            figureArray.Add(circ);  
            Console.WriteLine("Неотсортированный список:");  
            foreach (var figure in figureArray)  
            {  
                Console.WriteLine(figure);  
            }  
            figureArray.Sort();  
            Console.WriteLine("Отсортированный список:");  
            foreach (var figure in figureArray)  
            {  
                Console.WriteLine(figure);  
            }  
            Console.WriteLine("Работаем с List");  
            List<GeometricFigure> figureList = new List<GeometricFigure>();  
            figureList.Add(rect);  
            figureList.Add(sq);  
            figureList.Add(circ);  
            Console.WriteLine("Неотсортированный список:");  
            foreach (var figure in figureList)  
            {  
                Console.WriteLine(figure);  
            }  
            figureList.Sort();  
            Console.WriteLine("Отсортированный список:");  
            foreach (var figure in figureList)  
            {  
                Console.WriteLine(figure);  
            }  
  
            Console.WriteLine("\nМатрица");  
            Console.WriteLine("Введите рамер матрицы");  
            int a = (int)InputValue("Введите размер i:");  
            int b = (int)InputValue("Введите размер j:");  
            int c = (int)InputValue("Введите размер k:");  
            Matrix<GeometricFigure> matrix = new Matrix<GeometricFigure>(a, b, c, new FigureMatrixCheckEmpty());  
            Console.WriteLine("Заполнение матрицы\n");  
            matrix[0, 0, 0] = new Rectangle(10, 5);  
            matrix[0, 0, 1] = new Circle(10);  
            matrix[0, 1, 0] = new Square(5);  
            matrix[0, 1, 1] = new Rectangle(1, 10);  
            matrix[1, 0, 0] = new Circle(5);  
            matrix[1, 0, 1] = new Square(10);  
            matrix[1, 1, 0] = new Circle(100);  
            matrix[1, 1, 1] = new Circle(50);  
            Console.WriteLine("Матрица:\n");  
            Console.WriteLine(matrix.ToString());  
  
            SimpleStack<GeometricFigure> stack = new SimpleStack<GeometricFigure>();  
            Console.WriteLine("Добавим прямоугольник в стек");  
            double width1 = InputValue("Введите ширину прямоугольника:");  
            double height1 = InputValue("Введите высоту прямоугольника");  
            Rectangle rect1 = new Rectangle(width, height);  
            stack.Push(rect1);  
            Console.WriteLine("\n");  
            Console.WriteLine("Добавим квадрат в стек");  
            double length1 = InputValue("Введите сторону квадрата:");  
            Square square1 = new Square(length);  
            stack.Push(square1);  
            Console.WriteLine("\n");  
            Console.WriteLine("Добавим круг в стек\n");  
            double radius1 = InputValue("Введите радиус круга:");  
            Circle circle1 = new Circle(radius);  
            Console.WriteLine("\n");  
            stack.Push(circle1);  
            Console.WriteLine("Последний элемент в стеке:");  
            Console.WriteLine(stack);  
            Console.WriteLine("Удалим последний элемент в стеке:");  
            stack.Pop();  
            Console.WriteLine("Последний элемент в стеке:");  
            Console.WriteLine(stack);  
        }  
    }  
  
    interface IPrint  
    {  
        void Print();  
    }  
  
  
    abstract class GeometricFigure : IComparable  
    {  
        const double EPSILON = 0.00001;  
        public abstract double Area();  
  
        public int CompareTo(object obj)  
        {  
            GeometricFigure p = (GeometricFigure)obj;  
            if (this.Area() < p.Area())  
                return -1;  
            else if (Math.Abs(this.Area() - p.Area()) < EPSILON)  
                return 0;  
            else  
                return 1;  
        }  
    }  
  
    class Rectangle : GeometricFigure, IPrint  
    {  
        private double \_width;  
        private double \_height;  
  
        public Rectangle(double w, double h)  
        {  
            width = w;  
            height = h;  
        }  
  
        public double width  
        {  
            get { return \_width; }  
            private set { \_width = value; }  
        }  
  
        public double height  
        {  
            get { return \_height; }  
            private set { \_height = value; }  
        }  
  
        public override double Area()  
        {  
            return width \* height;  
        }  
  
        public override string ToString()  
        {  
            return string.Format("[Rectangle: width = {0}, height = {1}, area = {2}]", width, height, Area());  
        }  
  
        public void Print()  
        {  
            Console.WriteLine(this);  
        }  
    }  
  
    class Square : Rectangle, IPrint  
    {  
        public Square(double l) : base(l, l) { }  
  
        public override string ToString()  
        {  
            return string.Format("[Square: length = {0}, area = {1}]", height, Area());  
        }  
    }  
  
    class Circle : GeometricFigure, IPrint  
    {  
        private double \_radius;  
  
        public Circle(double r)  
        {  
            radius = r;  
        }  
  
        public double radius  
        {  
            get { return \_radius; }  
            private set { \_radius = value; }  
        }  
  
        public override double Area()  
        {  
            return Math.PI \* radius \* radius;  
        }  
  
        public override string ToString()  
        {  
            return string.Format("[Circle: radius = {0}, area = {1}]", radius, Area());  
        }  
  
        public void Print()  
        {  
            Console.WriteLine(this);  
        }  
    }  
  
    public interface IMatrixCheckEmpty<T>  
    {  
        T getEmptyElement();  
        bool checkEmptyElement(T element);  
    }  
    public class Matrix<T>  
    {  
        Dictionary<string, T> \_matrix = new Dictionary<string, T>();  
  
        int maxX;  
        int maxY;  
        int maxZ;  
  
        IMatrixCheckEmpty<T> checkEmpty;  
  
        public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam) {  
            this.maxX = px;  
            this.maxY = py;  
            this.maxZ = pz;  
            this.checkEmpty = checkEmptyParam;  
        }  
  
        public T this[int x, int y, int z]  
        {  
            set  
            {  
                CheckBounds(x, y, z);  
                string key = DictKey(x, y, z);  
                this.\_matrix.Add(key, value);  
            }  
            get  
            {  
                CheckBounds(x, y, z);  
                string key = DictKey(x, y, z);  
                if (this.\_matrix.ContainsKey(key))  
                {  
                    return this.\_matrix[key];  
                }  
                else  
                {  
                    return this.checkEmpty.getEmptyElement();  
                }  
            }  
        }  
  
        void CheckBounds (int x, int y, int z)   
        {  
            if (x < 0 || x >= this.maxX)  
            {  
                throw new ArgumentException("x", "x = " + x + "выходит за границы");  
            }  
            if (y < 0 || y >= this.maxY)  
            {  
                throw new ArgumentException("y", "y = " + y + "выходит за границы");  
            }  
            if (z < 0 || z >= this.maxZ)  
            {  
                throw new ArgumentException("z", "z = " + z + "выходит за границы");  
            }  
        }  
  
        string DictKey(int x, int y, int z)  
        {  
            return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();  
        }  
  
        public override string ToString()  
        {  
            StringBuilder b = new StringBuilder();  
  
            for (int i = 0; i < this.maxX; i++)  
            {  
                for (int j = 0; j < this.maxY; j++)  
                {  
                    for (int k = 0; k < this.maxZ; k++)  
                    {  
                        b.Append("i = " + i + " j = " + j + " k = " + k + " ");  
                        b.Append(this[i, j, k].ToString() + "\n");   
                    }  
                }  
            }  
  
            return b.ToString();  
        }  
    }  
  
    class FigureMatrixCheckEmpty: IMatrixCheckEmpty<GeometricFigure>  
    {  
        public GeometricFigure getEmptyElement()  
        {  
            return null;  
        }  
  
        public bool checkEmptyElement(GeometricFigure element)  
        {  
            bool Result = false;  
            if (element == null)  
            {  
                Result = true;  
            }  
            return Result;  
        }  
    }  
  
    public class SimpleListItem<T>  
    {  
        public T data { get; set; }  
  
        public SimpleListItem<T> next { get; set; }  
  
        public SimpleListItem(T param)  
        {  
            this.data = param;  
        }  
    }  
  
    public class SimpleList<T> : IEnumerable<T> where T : IComparable  
    {  
        protected SimpleListItem<T> first = null;  
  
        protected SimpleListItem<T> last = null;  
  
        public void Print()  
        {  
            Console.WriteLine(this.ToString());  
        }  
  
        public int Count  
        {  
            get { return \_count; }  
            protected set { \_count = value; }  
        }  
  
        int \_count;  
  
        public void Add(T element)  
        {  
            SimpleListItem<T> newItem = new SimpleListItem<T>(element);  
            this.Count++;  
            if (last == null)  
            {  
                this.first = newItem;  
                this.last = newItem;  
            }  
            else  
            {  
                this.last.next = newItem;  
                this.last = newItem;  
            }  
        }  
  
        public SimpleListItem<T> GetItem(int number)  
        {  
            if ((number < 0) || (number >= this.Count))  
            {  
                throw new Exception("Выход за границу индекса");  
            }  
            SimpleListItem<T> current = this.first;  
            int i = 0;  
            while (i < number)  
            {  
                current = current.next;  
                i++;  
            }  
            return current;  
        }  
  
        public T Get(int number)  
        {  
            return GetItem(number).data;  
        }  
  
        public IEnumerator<T> GetEnumerator()  
        {  
            SimpleListItem<T> current = this.first;  
  
            while (current != null)  
            {  
                yield return current.data;  
                current = current.next;  
            }  
        }  
  
        System.Collections.IEnumerator  
        System.Collections.IEnumerable.GetEnumerator()  
        {  
            return GetEnumerator();  
        }  
  
        public void Sort()  
        {  
            Sort(0, this.Count - 1);  
        }  
  
        private void Sort(int low, int high)  
        {  
            int i = low;  
            int j = high;  
            T x = Get((low + high) / 2);  
            do  
            {  
                while (Get(i).CompareTo(x) < 0) ++i;  
                while (Get(j).CompareTo(x) > 0) --j;  
                if (i <= j)  
                {  
                    Swap(i, j);  
                    i++;  
                    j--;  
                }  
            } while (i <= j);  
            if (low < j) Sort(low, j);  
            if (i < high) Sort(i, high);  
        }  
  
        private void Swap(int i, int j)  
        {  
            SimpleListItem<T> ci = GetItem(i);  
            SimpleListItem<T> cj = GetItem(j);  
            T temp = ci.data;  
            ci.data = cj.data;  
            cj.data = temp;  
        }  
    }  
  
    class SimpleStack<T> : SimpleList<T> where T : IComparable  
    {  
        public void Push(T element)  
        {  
            Add(element);  
        }  
  
        public T Pop()  
        {  
            T Result = default(T);  
            if (this.Count == 0) return Result;  
  
            if (this.Count == 1)  
            {  
                Result = this.first.data;  
                this.first = null;  
                this.last = null;  
            }  
  
            else  
            {  
                SimpleListItem<T> newLast = this.GetItem(this.Count - 2);  
                Result = newLast.next.data;  
                this.last = newLast;  
                newLast.next = null;  
            }  
            this.Count--;  
            return Result;  
        }  
  
        public SimpleListItem<T> LastElem()  
        {  
            return last;  
        }  
  
        public override string ToString()  
        {  
            if (last != null)  
                return last.data.ToString();  
            else  
            {  
                return "Стек пуст";  
            }  
        }  
    }  
}

### III. Примеры работы





