Московский государственный технический

университет им. Н.Э. Баумана.

Факультет «Информатика и системы управления»

Кафедра ИУ5. Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе № 3

«Работа с коллекциями»

Выполнил:

студент группы ИУ5-33

Терентьев Владислав

Москва, 2019 г.

1. Постановка задачи

Разработать программу, реализующую работу с коллекциями.

Объекты классов «Прямоугольник», «Квадрат», «Круг» использовать из проекта лабораторной работы №2.

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

Создать коллекцию класса ArrayList. Сохранить объекты (типы) Прямоугольник, Квадрат, Круг, в коллекцию. Вывести в цикле содержимое площади элементов коллекции.

Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

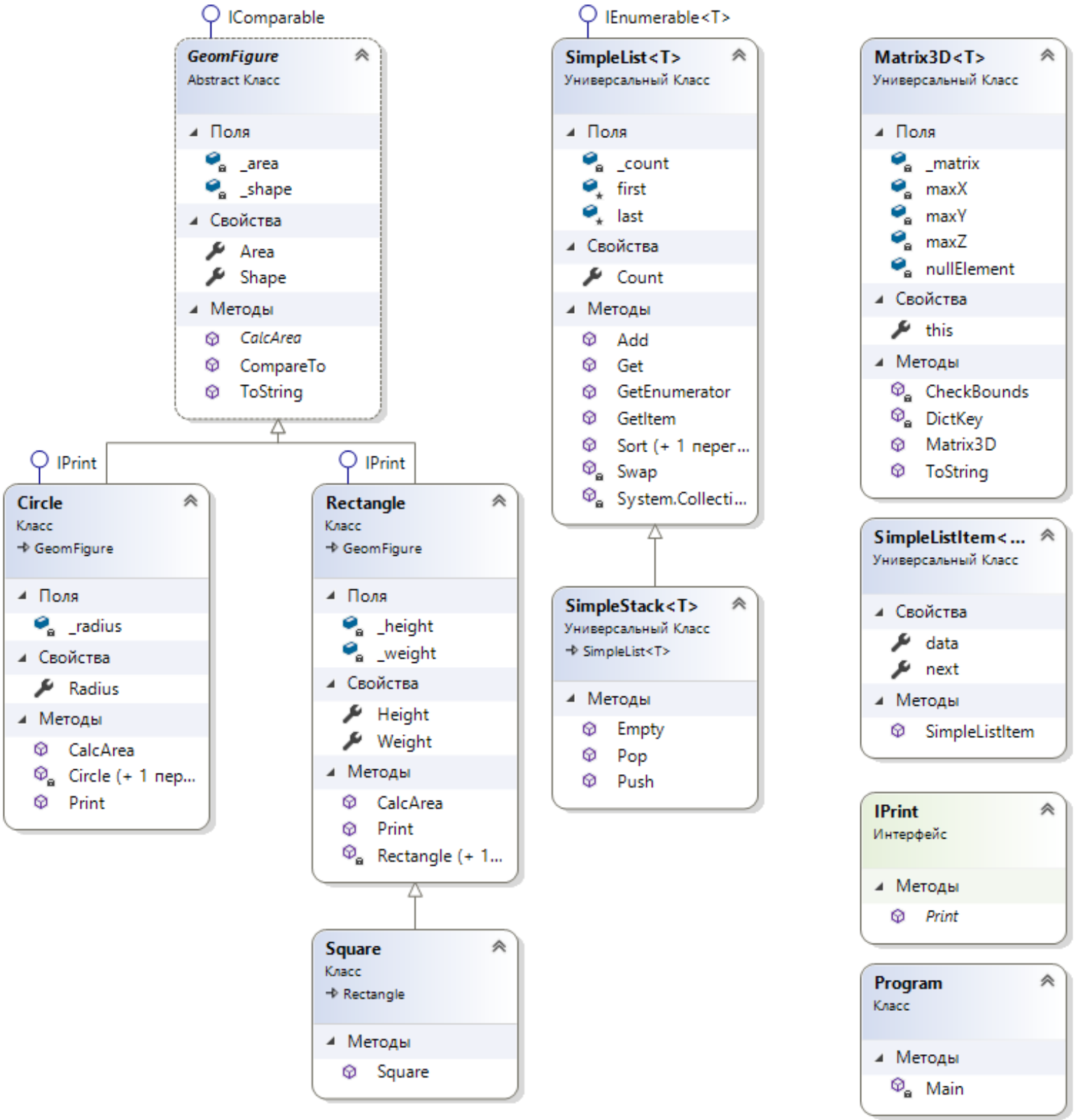
Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

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.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

2. Диаграмма классов



3. Текст программы

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace laba3
{

    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Терентьев Владислав ИУ5-33";
            Rectangle rectangle = new Rectangle(5, 4);
            Square square = new Square(5);
            Circle circle = new Circle(5);

            Console.WriteLine("ArrayList");
            ArrayList collection = new ArrayList();
            collection.Add(circle);
            collection.Add(rectangle);
            collection.Add(square);
            foreach (object o in collection)
            {
                Console.WriteLine(o);
            }

            Console.WriteLine("\nList<GeomFigure>");
            List<GeomFigure> coll2 = new List<GeomFigure>();
            coll2.Add(circle);
            coll2.Add(rectangle);
            coll2.Add(square);
            foreach (object o in coll2)
            {
                Console.WriteLine(o);
            }
            Console.WriteLine("\nList<GeomFigure> - сортировка");
            coll2.Sort();
            foreach (object o in coll2)
            {
                Console.WriteLine(o);
            }

            //
            Console.WriteLine("\nМатрица");
            Matrix3D<GeomFigure> cube = new Matrix3D<GeomFigure>(3, 3, 3, null);
            cube[0, 0, 2] = rectangle;
            cube[1, 1, 1] = square;
            cube[2, 2, 0] = circle;
            Console.WriteLine(cube.ToString());
            //

            Console.WriteLine("\nСписок");
            SimpleList<GeomFigure> list = new SimpleList<GeomFigure>();
            list.Add(square);
            list.Add(rectangle);
            list.Add(circle);
            foreach (var o in list)
            {
                Console.WriteLine(o);
            }
            list.Sort();
            Console.WriteLine("\nСортировка списка");
            foreach (var o in list)
            {
                Console.WriteLine(o);
            }

            Console.WriteLine("\nСтек");
```

```
            SimpleStack<GeomFigure> stack = new SimpleStack<GeomFigure>();
            stack.Push(rectangle);
            stack.Push(square);
            stack.Push(circle);
            while (stack.Count > 0)
            {
                GeomFigure tmp = stack.Pop();
                Console.WriteLine(tmp);
            }
            Console.ReadKey();
        }
    }

    abstract class GeomFigure : IComparable
    {
        string _shape;
        double _area;

        public string Shape
        {
            get { return _shape; }
            protected set { _shape = value; }
        }

        public double Area
        {
            get { return _area; }
            protected set { _area = value; }
        }

        public abstract double CalcArea();


        public int CompareTo(object o)
        {
            GeomFigure a = (GeomFigure)o;
            if (_area < a._area) return -1;
            else if (_area == a._area) return 0;
            else return 1;
        }

        public override string ToString() { return _shape + " площадью " +
_area.ToString("0.00"); }
    }

    interface IPrint { void Print(); }

    class Rectangle : GeomFigure, IPrint
    {
        double _height = 0;
        double _weight = 0;

        Rectangle() { Shape = "Прямоугольник"; }

        public override double CalcArea() { return _height * _weight; }

        public double Height
        {
            get { return _height; }
            private set
            {
                if (value < 0) { throw new Exception("Высота не может быть
отрицательной"); }
                else { _height = value; }
            }
```

```
        }

        public double Weight
        {
            get { return _weight; }
            private set
            {
                if (value < 0) { throw new Exception("Ширина не может быть
отрицательной"); }
                else { _weight = value; }
            }
        }

        public Rectangle(double h, double w) : this()
        {
            Height = h;
            Weight = w;
            Area = CalcArea();
        }

        //public override string ToString() { return base.ToString() + ", высотой " +
_height + " и шириной " + _weight; }

        public void Print() { Console.WriteLine(ToString()); }
    }

    class Square : Rectangle
    {
        public Square(double leng) : base(leng, leng) { Shape = "Квадрат"; }

        //public override string ToString() { return base.ToString() + " (стороной " +
Height + ")"; }
    }

    class Circle : GeomFigure, IPrint
    {
        double _radius = 0;

        Circle() { Shape = "Окружность"; }

        public override double CalcArea() { return Math.PI * _radius * _radius; }

        public double Radius
        {
            get { return _radius; }
            private set
            {
                if (value < 0) { throw new Exception("Радиус не может быть
отрицательным"); }
                else { _radius = value; }
            }
        }

        public Circle(double r) : this()
        {
            Radius = r;
            Area = CalcArea();
        }

        //public override string ToString() { return base.ToString() + " и радиусом " +
_radius; }

        public void Print() { Console.WriteLine(ToString()); }
    }
```

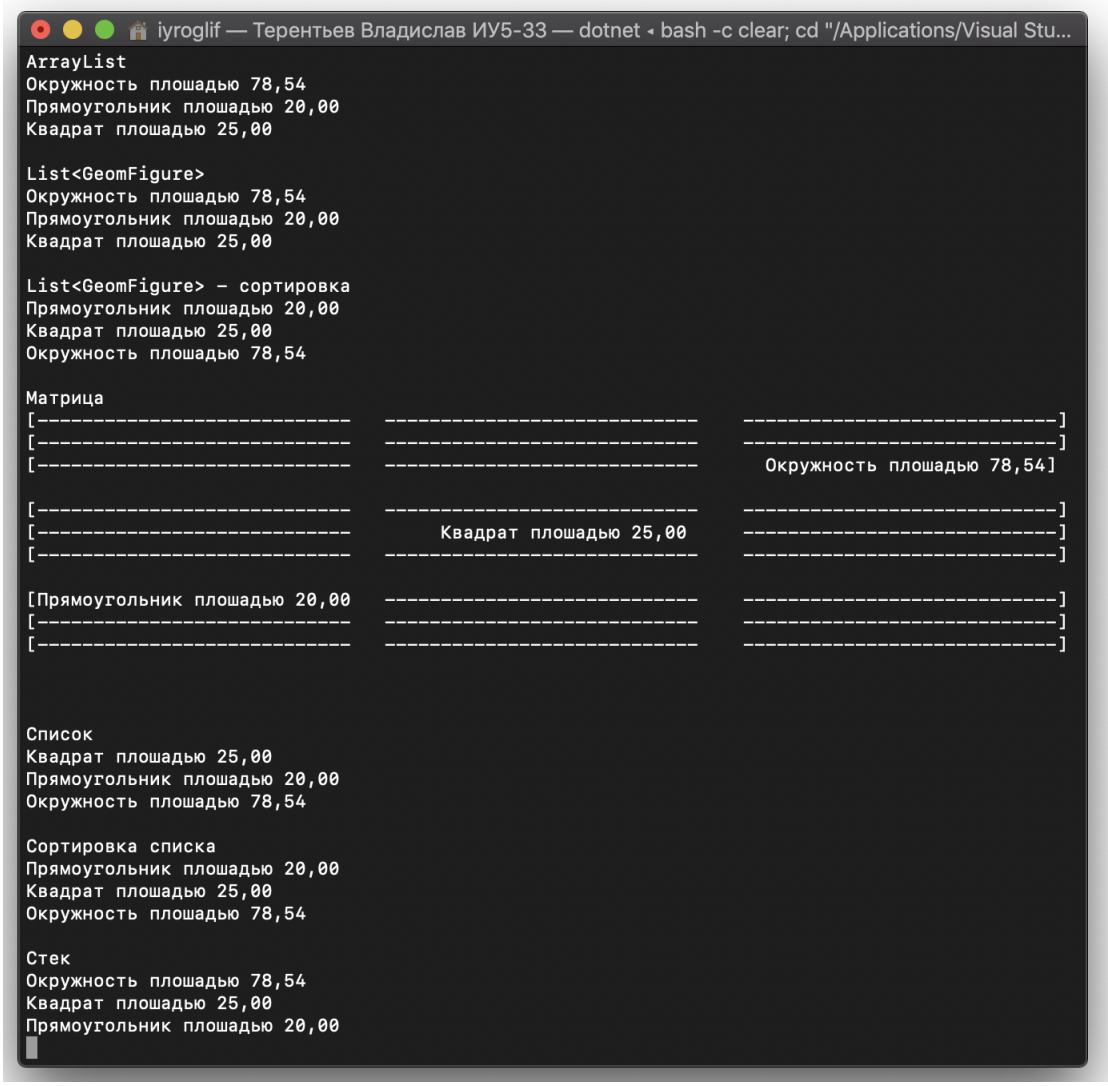
```
    public class Matrix3D<T>
    {
        Dictionary<string, T> _matrix = new Dictionary<string, T>();
        int maxX;
        int maxY;
        int maxZ;
        T nullElement;

        public Matrix3D(int px, int py, int pz, T nullElementParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.nullElement = nullElementParam;
        }

        public T this[int x, int y, int z]
        {
            get
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                if (this._matrix.ContainsKey(key)) { return this._matrix[key]; }
                else { return this.nullElement; }
            }
            set
            {
                CheckBounds(x, y, z);
                string key = DictKey(x, y, z);
                this._matrix.Add(key, value);
            }
        }

        void CheckBounds(int x, int y, int z)
        {
            if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за
границы");
            if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за
границы");
            if (z < 0 || z >= this.maxZ) throw new Exception("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 k = 0; k < this.maxZ; k++)
            {
                for (int j = 0; j < this.maxY; j++)
                {
                    b.Append("[");
                    for (int i = 0; i < this.maxX; i++)
                    {
                        if (i > 0) b.Append("\t");
                        if (this[i, j, k] == null) b.Append("----------------------------
");
                        else b.Append($"{this[i, j, k].ToString(), 27}");
                    }
                    b.Append("]\n");
                }
                b.Append("\n");
            }
```

```
            return b.ToString();
        }
    }

    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 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 bool Empty()
    {
        if (last == null) return true;
        else return false;
    }

    public void Push(T element)
    {
        SimpleListItem<T> newItem = new SimpleListItem<T>(element);
        this.Count++;
```

```
            if (last == null)
            {
                this.first = newItem;
                this.last = newItem;
            }
            else
            {
                newItem.next = last;
                last = newItem;
            }
        }

        public T Pop()
        {
            if (Empty()) { throw new Exception("Удаление элемента из пустого стека"); }
            else
            {
                Count--;
                T temp = last.data;
                if (last.next == null) { last = null; first = null; }
                else { last = last.next; }
                return temp;
            }
        }
    }
}
```

4. Анализ результатов



5. Ссылка на репозиторий

https://github.com/iYroglif/newlabs