Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5.

Курс «Базовые компоненты интернет-технологий»

Отчет по ЛР3.

Выполнил:

студент группы ИУ5-31Б

Вардумян Арсен

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Юрий Евгеньевич

Подпись и дата:

г. Москва, 2020 г.

Постановка задачи

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.

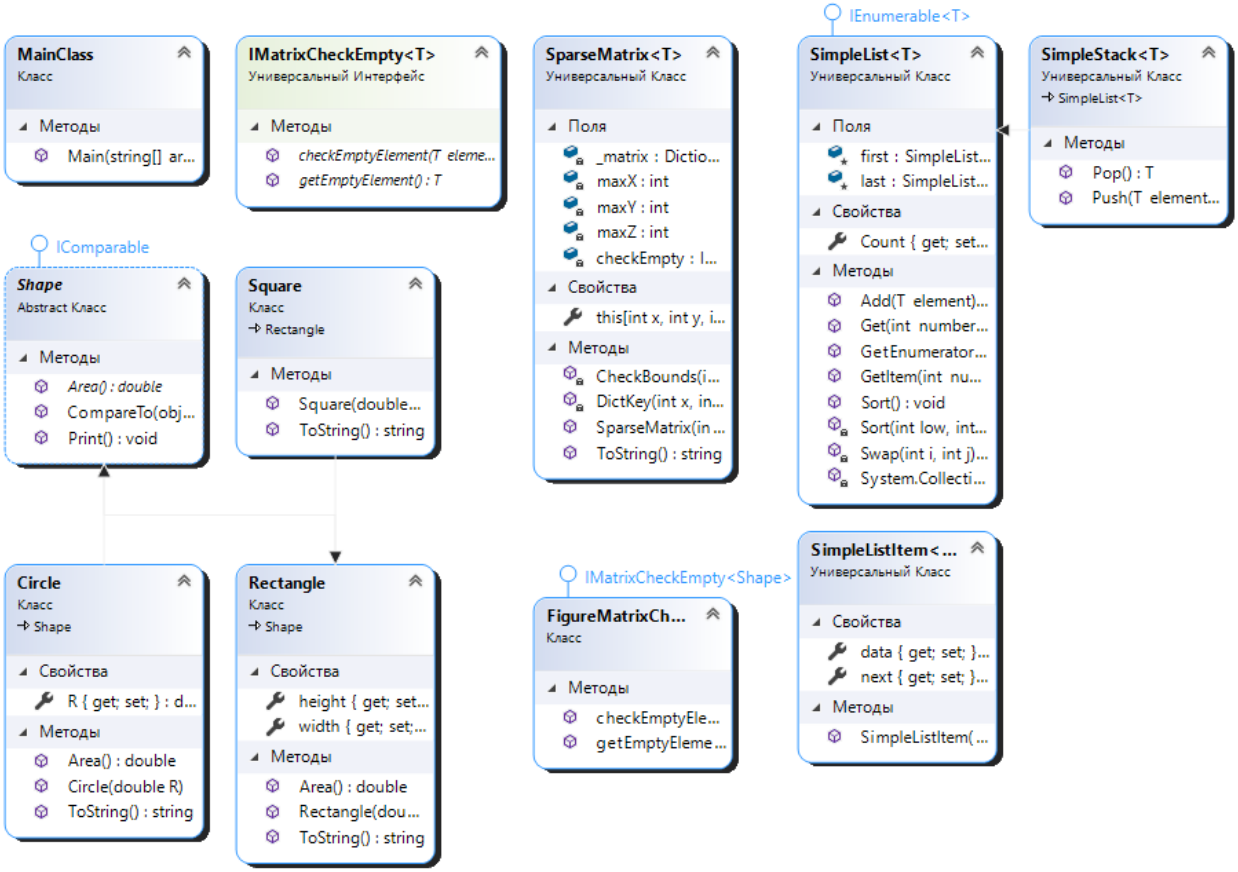
6.
Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

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.

Разработка интерфейса класса



Листинг программы

//Program.cs

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

namespace Lab3
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.OutputEncoding = System.Text.Encoding.UTF8;

            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("ArrayList");
            Console.ForegroundColor = ConsoleColor.Black;

            var arr = new ArrayList
            {
                new Rectangle(6, 4),
                new Rectangle(5, 2),
                new Circle(10),
                new Square(7),
                new Rectangle(13, 7),
                new Circle(11)
```

```
};

arr.Sort();

foreach (Shape item in arr)
{
    item.Print();
}

Console.WriteLine();


Console.ForegroundColor = ConsoleColor.Red;
Console.WriteLine("\nShapesList");
Console.ForegroundColor = ConsoleColor.Black;

var ShapesList = new List<Shape>
{
    new Rectangle(3, 4),
    new Square(9),
    new Circle(10),
    new Square(7),
    new Circle(2)
};

ShapesList.Sort();

foreach (Shape item in ShapesList)
{
    item.Print();
}

Console.ForegroundColor = ConsoleColor.Red;
Console.WriteLine("\nМатрица");
Console.ForegroundColor = ConsoleColor.Black;

SparseMatrix<Shape> matrix = new SparseMatrix<Shape>(3, 3, 3,
new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = ShapesList[0];
matrix[1, 1, 1] = ShapesList[1];
matrix[2, 2, 2] = ShapesList[2];

Console.WriteLine(matrix.ToString());

Console.ForegroundColor = ConsoleColor.Red;
Console.WriteLine("\nСтек фигур");
Console.ForegroundColor = ConsoleColor.Black;

SimpleStack<Shape> ShapeStack = new SimpleStack<Shape>();
ShapeStack.Push(ShapesList[0]);
ShapeStack.Push(ShapesList[3]);
ShapeStack.Push(ShapesList[2]);
ShapeStack.Push(ShapesList[2]);
ShapeStack.Push(ShapesList[1]);
ShapeStack.Push(ShapesList[0]);

while (ShapeStack.Count > 0)
{
    Console.WriteLine(ShapeStack.Pop());
}
```

```
        }
    }
}
```

//Shape.cs

```
using System;
namespace Lab3
{
    public abstract class Shape : IComparable
    {
        public abstract double Area();

        public int CompareTo(object obj)
        {
            Shape x = obj as Shape;

            if (x != null)
                return Area().CompareTo(x.Area());
            else
                throw new Exception("Невозможно сравнить два объекта");
        }

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

    public class Rectangle : Shape
    {
        public double height { get; private set; }

        public double width { get; private set; }

        public Rectangle(double height, double width)
        {
            this.height = height;
            this.width = width;
        }

        public override double Area()
        {
            return height * width;
        }

        public override string ToString()
        {
            return $"Прямоугольник с высотой {height}, шириной {width} и площадью
{this.Area()}";
        }
    }

    public class Square : Rectangle
    {
        public Square(double side) : base(side, side)
        {

        }

        public override string ToString()
```

```
        {
            return $"Квадрат со стороной {height} и площадью {this.Area()}";
        }
    }

    public class Circle : Shape
    {
        public double R { get; private set; }

        public Circle(double R)
        {
            this.R = R;
        }

        public override double Area()
        {
            return Math.PI * R * R;
        }

        public override string ToString()
        {
            return $"Круг с радиусом {R} и площадью {this.Area()}";
        }
    }

    class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Shape>
    {
        public Shape getEmptyElement()
        {
            return null;
        }

        public bool checkEmptyElement(Shape element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }
    }
}
```

//SparseMatrix.cs

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

namespace Lab3
{
    public class SparseMatrix<T>
    {
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        int maxX;

        int maxY;
```

```
        int maxZ;

        IMatrixCheckEmpty<T> checkEmpty;

        public SparseMatrix(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 ArgumentOutOfRangeException("x",
                    "x=" + x + " выходит за границы");
            }
            if (y < 0 || y >= this.maxY)
            {
                throw new ArgumentOutOfRangeException("y",
                    "y=" + y + " выходит за границы");
            }
            if (z < 0 || z >= this.maxZ)
            {
                throw new ArgumentOutOfRangeException("y",
                    "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 j = 0; j < this.maxY; j++)
            {
                b.Append("[");
                for (int i = 0; i < this.maxX; i++)
                {
                    b.Append("[");
                    if (i > 0)
                    {
                        b.Append("\t");
                    }
                    for (int k = 0; k < this.maxZ; k++)
                    {
                        if (k > 0)
                        {
                            b.Append("\t");
                        }

                        if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))
                        {
                            b.Append(this[i, j, k].ToString());
                        }
                        else
                        {
                            b.Append(" - ");
                        }
                    }
                    b.Append("]");
                }
                b.Append("]\n");
            }
            return b.ToString();
        }
    }

    public interface IMatrixCheckEmpty<T>
    {
        T getEmptyElement();

        bool checkEmptyElement(T element);
    }
}
```

//SimpleList.cs

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

namespace Lab3
{
    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; protected set; }

    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;
        }
    }
}
```

//SimpleStack.cs

```
using System;
namespace Lab3
{
    public class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        public void Push(T element)
        {
            this.Add(element);
        }

        public T Pop()
        {
```

```
            T Result = default;

            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;
        }
    }
}
```

Анализ результатов

Терминал – Lab3

```
ShapesList
Прямоугольник с высотой 3, шириной 4 и площадью 12
Круг с радиусом 2 и площадью 12.5663706143592
Квадрат со стороной 7 и площадью 49
Квадрат со стороной 9 и площадью 81
Круг с радиусом 10 и площадью 314.159265358979

Матрица
[[Прямоугольник с высотой 3, шириной 4 и площадью 12     –       – ][    –       –       – ][    –       –       – ]]
[[ –     –       – ][    –      Круг с радиусом 2 и площадью 12.5663706143592    – ][    –       –       – ]]
[[ –     –       – ][    –       –       – ][    –       –      Квадрат со стороной 7 и площадью 49]]


Стек фигур
Прямоугольник с высотой 3, шириной 4 и площадью 12
Круг с радиусом 2 и площадью 12.5663706143592
Квадрат со стороной 7 и площадью 49
Квадрат со стороной 7 и площадью 49
Квадрат со стороной 9 и площадью 81
Прямоугольник с высотой 3, шириной 4 и площадью 12
```

Терминал – Lab3

```
ShapesList
Квадрат со стороной 2 и площадью 4
Прямоугольник с высотой 12, шириной 3 и площадью 36
Круг с радиусом 6.6 и площадью 136.847775990371
Квадрат со стороной 17 и площадью 289
Круг с радиусом 20 и площадью 1256.63706143592

Матрица
[[Квадрат со стороной 2 и площадью 4     –       – ][    –       –       – ][    –       –       – ]]
[[ –     –       – ][    –      Прямоугольник с высотой 12, шириной 3 и площадью 36      – ][    –       –       – ]]
[[ –     –       – ][    –       –       – ][    –       –      Круг с радиусом 6.6 и площадью 136.847775990371]]


Стек фигур
Квадрат со стороной 2 и площадью 4
Прямоугольник с высотой 12, шириной 3 и площадью 36
Круг с радиусом 6.6 и площадью 136.847775990371
Круг с радиусом 6.6 и площадью 136.847775990371
Квадрат со стороной 17 и площадью 289
Квадрат со стороной 2 и площадью 4
```