Московский государственный технический
университет им. Н.Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:
студент группы РТ5-31Б

Ходосов Михаил
Подпись и дата:

Проверил:
доцент каф. ИУ5

Гапанюк Ю.Е.
Подпись и дата:

Москва, 2020 г.

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.

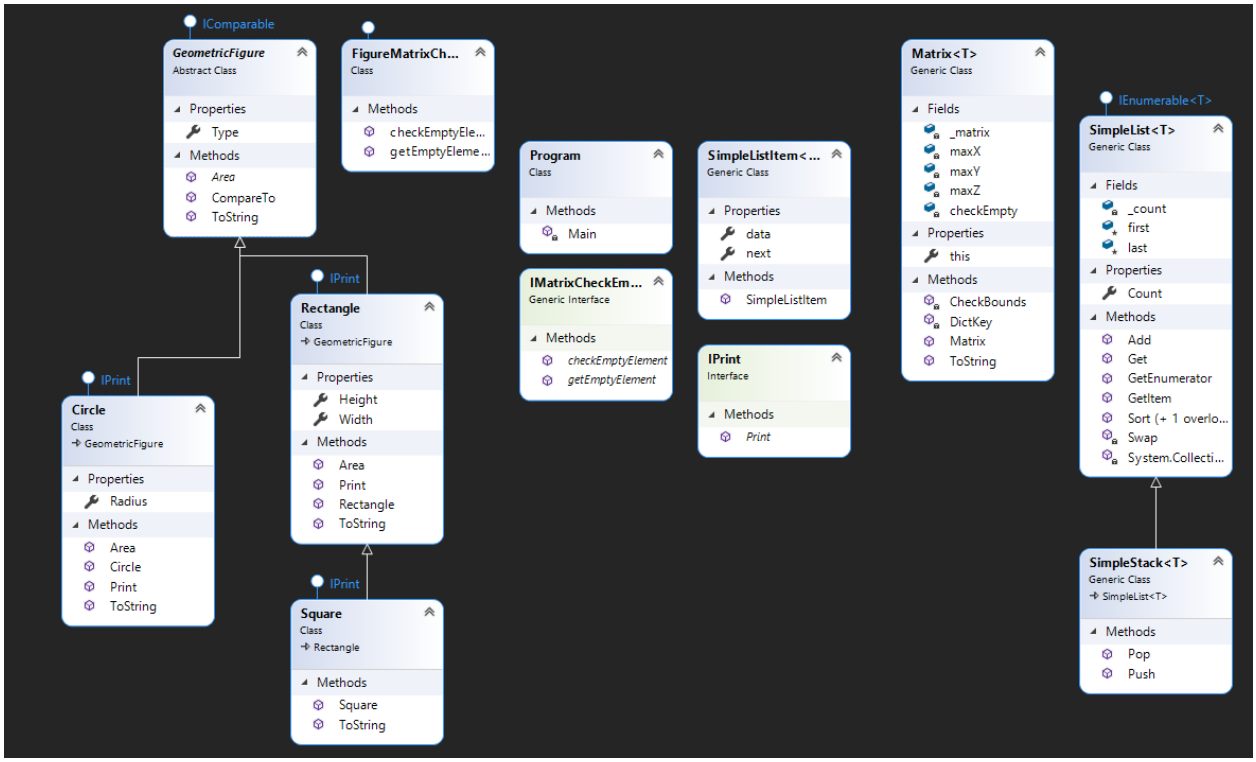
6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

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.

Диаграмма классов



Текст программы

Figures.cs

```
using System;

namespace Figures
{
    abstract class GeometricFigure : IComparable
    {
        public string Type { get; protected set; }

        public abstract double Area();

        public override string ToString()
        {
            return this.Type + " площадью " + this.Area().ToString();
        }
```

```
        public int CompareTo(object obj)
        {
            GeometricFigure right = (GeometricFigure)obj;

            if (this.Area() < right.Area())
                return -1;
            else if (this.Area() == right.Area())
                return 0;
            else
                return 1;
        }
    }

    interface IPrint
    {
        void Print();
    }

    class Rectangle : GeometricFigure, IPrint
    {

        public double Height { get; protected set; }
        public double Width { get; protected set; }

        public Rectangle(double ph, double pw)
        {
            this.Height = ph;
            this.Width = pw;
            this.Type = "Прямоугольник";
        }

        public override double Area()
        {
            return (this.Height * this.Width);
        }

        public override string ToString()
        {
            return this.Type + " шириной " + this.Width.ToString() + ", высотой " +
this.Height.ToString() + ", площадью " + this.Area().ToString();
        }

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

    class Square : Rectangle, IPrint
    {
        public Square(double size) : base(size, size)
        {
            this.Type = "Квадрат";
        }

        public override string ToString()
        {
            return this.Type + " со стороной " + this.Width.ToString() + ", площадью " +
this.Area().ToString();
        }
    }

    class Circle : GeometricFigure, IPrint
    {
        public double Radius { get; protected set; }
```

```
        public Circle(double pr)
        {
            this.Radius = pr;
            this.Type = "Круг";
        }

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

        public override string ToString()
        {
            return this.Type + " радиусом " + this.Radius.ToString() + ", площадью " +
this.Area().ToString();
        }

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

Matrix.cs

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

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

        bool checkEmptyElement(T element);
    }

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

        public bool checkEmptyElement(GeometricFigure element)
        {
            return element == null ? true : false;
        }
    }

    public class Matrix<T>
    {
        /// <summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// <summary>
        /// Количество элементов по горизонтали (максимальное количество столбцов)
        /// </summary>
```

```
        int maxX;

        /// <summary>
        /// Количество элементов по вертикали (максимальное количество строк)
        /// </summary>
        int maxY;

        /// <summary>
        /// Количество элементов по фронтали (максимальное количество в третьем
измерении)
        /// </summary>
        int maxZ;

        /// <summary>
        /// Реализация интерфейса для проверки пустого элемента
        /// </summary>
        IMatrixCheckEmpty<T> checkEmpty;

        /// <summary>
        /// Конструктор
        /// </summary>
        public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam)
        {
            this.maxX = px;
            this.maxY = py;
            this.maxZ = pz;
            this.checkEmpty = checkEmptyParam;
        }

        /// <summary>
        /// Индексатор для доступа к данных
        /// </summary>
        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();
                }
            }
        }

        /// <summary>
        /// Проверка границ
        /// </summary>
        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("z", "z=" + z + " выходит за
границы");
            }
        }

        /// <summary>
        /// Формирование ключа
        /// </summary>
        string DictKey(int x, int y, int z)
        {
            return x.ToString() + "_" + y.ToString() + "_" + z.ToString();
        }

        /// <summary>
        /// Приведение к строке
        /// </summary>
        /// <returns></returns>
        public override string ToString()
        {
            //Класс StringBuilder используется для построения длинных строк
            //Это увеличивает производительность по сравнению с созданием и склеиванием
            //большого количества обычных строк

            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++)
                    {
                        //Добавление разделителя-табуляции
                        if (i > 0)
                        {
                            b.Append("\t");
                        }
                        //Если текущий элемент не пустой
                        if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))
                        {
                            //Добавить приведенный к строке текущий элемент
                            b.Append(this[i, j, k].ToString());
                        }
                        else
                        {
                            //Иначе добавить признак пустого значения
                            b.Append(" - ");
                        }
                    }
                    b.Append("]\n");
                }
                b.Append("\n");
            }
            return b.ToString();
        }
    }
}
```

SimpleList.cs

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

namespace Figures
{
    /// <summary>
    /// Список
    /// </summary>
    public class SimpleList<T> : IEnumerable<T>
        where T : IComparable
    {
        /// <summary>
        /// Первый элемент списка
        /// </summary>
        protected SimpleListItem<T> first = null;

        /// <summary>
        /// Последний элемент списка
        /// </summary>
        protected SimpleListItem<T> last = null;

        /// <summary>
        /// Количество элементов
        /// </summary>
        public int Count
        {
            get { return _count; }
            protected set { _count = value; }
        }
        int _count;

        /// <summary>
        /// Добавление элемента
        /// </summary>
        public void Add(T element)
        {
            SimpleListItem<T> newItem = new SimpleListItem<T>(element);
            this.Count++;

            //Добавление первого элемента
            if (last == null)
            {
                this.first = newItem;
                this.last = newItem;
            }
            //Добавление следующих элементов
            else
            {
                //Присоединение элемента к цепочке
                this.last.next = newItem;
                //Просоединенный элемент считается последним
                this.last = newItem;
            }
        }

        /// <summary>
        /// Чтение контейнера с заданным номером
        /// </summary>
        public SimpleListItem<T> GetItem(int number)
        {
            if ((number < 0) || (number >= this.Count))
            {
                //Можно создать собственный класс исключения
```

```
                throw new Exception("Выход за границу индекса");
            }

            SimpleListItem<T> current = this.first;
            int i = 0;

            //Пропускаем нужное количество элементов
            while (i < number)
            {
                //Переход к следующему элементу
                current = current.next;
                //Увеличение счетчика
                i++;
            }

            return current;
        }

        /// <summary>
        /// Чтение элемента с заданным номером
        /// </summary>
        public T Get(int number)
        {
            return GetItem(number).data;
        }

        /// <summary>
        /// Для перебора коллекции
        /// </summary>
        public IEnumerator<T> GetEnumerator()
        {
            SimpleListItem<T> current = this.first;

            //Перебор элементов
            while (current != null)
            {
                //Возврат текущего значения
                yield return current.data;
                //Переход к следующему элементу
                current = current.next;
            }
        }

        //Реализация обобщенного IEnumerator<T> требует реализации необобщенного
интерфейса
        //Данный метод добавляется автоматически при реализации интерфейса
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }

        /// <summary>
        /// Сортировка
        /// </summary>
        public void Sort()
        {
            Sort(0, this.Count - 1);
        }

        /// <summary>
        /// Алгоритм быстрой сортировки
        /// </summary>
        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);
        }

        /// <summary>
        /// Вспомогательный метод для обмена элементов при сортировке
        /// </summary>
        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;
        }
    }
}
```

SimpleListItem.cs

```
using System;

namespace Figures
{
    /// <summary>
    /// Элемент списка
    /// </summary>
    public class SimpleListItem<T>
    {
        /// <summary>
        /// Данные
        /// </summary>
        public T data { get; set; }

        /// <summary>
        /// Следующий элемент
        /// </summary>
        public SimpleListItem<T> next { get; set; }

        ///конструктор
        public SimpleListItem(T param)
        {
            this.data = param;
        }
    }
}
```

SimpleStack.cs

```
using System;

namespace Figures
{
    /// <summary>
    /// Класс стек
    /// </summary>
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        /// <summary>
        /// Добавление в стек
        /// </summary>
        public void Push(T element)
        {
            //Добавление в конец списка уже реализовано
            Add(element);
        }

        /// <summary>
        /// Удаление и чтение из стека
        /// </summary>
        public T Pop()
        {
            //default(T) - значение для типа T по умолчанию
            T Result = default(T);
            //Если стек пуст, возвращается значение по умолчанию для типа
            if (this.Count == 0) return Result;
            //Если элемент единственный
            if (this.Count == 1)
            {
                //то из него читаются данные
                Result = this.first.data;
                //обнуляются указатели начала и конца списка
                this.first = null;
                this.last = null;
            }
            //В списке более одного элемента
            else
            {
                //Поиск предпоследнего элемента
                SimpleListItem<T> newLast = this.GetItem(this.Count - 2);
                //Чтение значения из последнего элемента
                Result = newLast.next.data;
                //предпоследний элемент считается последним
                this.last = newLast;
                //последний элемент удаляется из списка
                newLast.next = null;
            }
            //Уменьшение количества элементов в списке
            this.Count--;
            //Возврат результата
            return Result;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using Figures;
using Matrix;
```

```
namespace lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            //Создание объектов классов фигур
            Rectangle rectangle = new Rectangle(3, 4);
            Square square = new Square(5);
            Circle circle = new Circle(2.7282);


            ArrayList array = new ArrayList
            {
                rectangle,
                square,
                circle
            };

            Console.WriteLine("Array list перед сортировкой");
            foreach (var elem in array)
                Console.WriteLine(elem);


            array.Sort();

            Console.WriteLine("\nArray list после сортировки");
            foreach (var elem in array)
                Console.WriteLine(elem);


            List<GeometricFigure> list = new List<GeometricFigure>
            {
                rectangle,
                square,
                circle
            };

            Console.WriteLine("\n\nList перед сортировкой");
            foreach (var elem in array)
                Console.WriteLine(elem);

            list.Sort();

            Console.WriteLine("\nList после сортировки");
            foreach (var elem in array)
                Console.WriteLine(elem);

            Console.WriteLine("\n\nМатрица");
            Matrix<GeometricFigure> matrix = new Matrix<GeometricFigure>(3, 3, 3, new
FigureMatrixCheckEmpty());
            matrix[0, 0, 0] = rectangle;
            matrix[1, 1, 1] = square;
            matrix[2, 2, 2] = circle;
            Console.WriteLine(matrix.ToString());

            try
            {
                GeometricFigure tmp = matrix[66, 1777, 11];
            }
            catch (ArgumentOutOfRangeException err)
            {
                Console.WriteLine(err.Message);
            }
```

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

            SimpleStack<GeometricFigure> stack = new SimpleStack<GeometricFigure>();
            stack.Push(rectangle);
            stack.Push(square);
            stack.Push(circle);

            while (stack.Count > 0)
            {
                GeometricFigure f = stack.Pop();
                Console.WriteLine(f);
            }
        }
    }
}
```

Результат работы программы