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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:
студент группы ИУ5-32
Кудрявцев Сергей
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.
Подпись и дата:

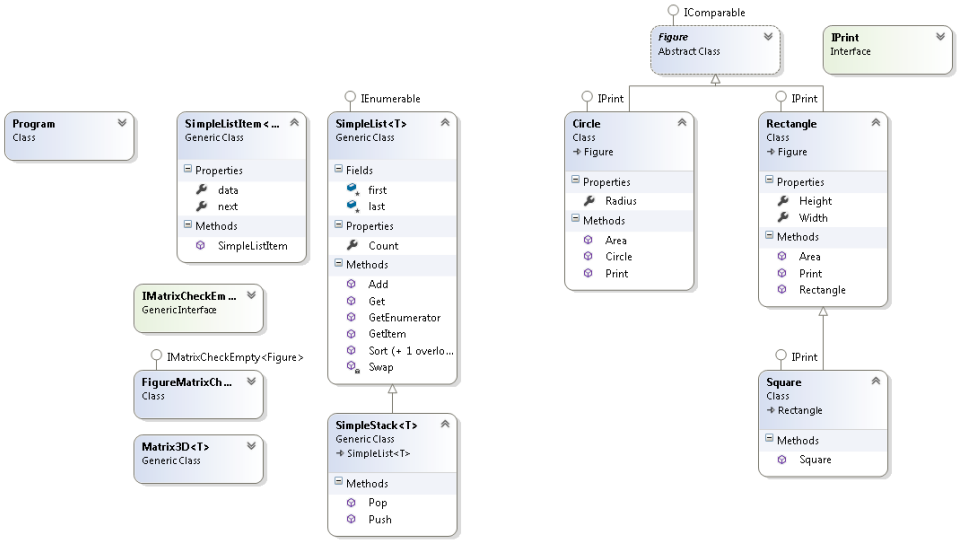
г. Москва, 2018 г.

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 (разобранного в пособии). Необходимо добавить в класс методы:
    a. public void Push(T element) – добавление в стек;
    b. public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

Диаграмма классов:



Текст программы:

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Collections;
using Lab2;

namespace Lab3
{
    internal class Program
    {
        public static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(5, 4);
            Square square = new Square(5);
            Circle circle = new Circle(5);

            Console.WriteLine("ArrayList");
            ArrayList arrayList = new ArrayList();
            arrayList.Add(circle);
            arrayList.Add(rect);
            arrayList.Add(square);

            foreach (var fig in arrayList)
            {
                Console.WriteLine(fig.ToString());
            }

            arrayList.Sort();

            Console.WriteLine("\nArrrayList после сортировки");

            foreach (var fig in arrayList)
            {
                Console.WriteLine(fig.ToString());
            }

            Console.WriteLine("\nList");
            List<Figure> list = new List<Figure>();
            list.Add(circle);
            list.Add(rect);
            list.Add(square);

            foreach (var fig in list)
            {
                Console.WriteLine(fig.ToString());
            }

            list.Sort();

            Console.WriteLine("\nList после сортировки");

            foreach (var fig in list)
            {
                Console.WriteLine(fig.ToString());
            }

            Console.WriteLine("\nМатрица");
            Matrix3D<Figure> cube = new Matrix3D<Figure>(3, 3, 3, new FigureMatrixCheckEmpty());
            cube[0, 0, 0] = rect;
            cube[1, 1, 1] = square;
            cube[2, 2, 2] = circle;
            Console.WriteLine(cube.ToString());
            Console.WriteLine("\nСписок");

            SimpleList<Figure> simpleList = new SimpleList<Figure>();
            simpleList.Add(square);
            simpleList.Add(rect);
            simpleList.Add(circle);
```

```
            foreach (var x in simpleList) Console.WriteLine(x);
            simpleList.Sort();
            Console.WriteLine("\nСортировка списка");
            foreach (var x in simpleList) Console.WriteLine(x);


            Console.WriteLine("\nСтек");
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(rect);
            stack.Push(square);
            stack.Push(circle);
            while (stack.Count > 0)
            {
                Figure f = stack.Pop();
                Console.WriteLine(f);
            }

        }
    }
}
```

IPrint.cs:

```
namespace Lab2
{
    public interface IPrint
    {
        void Print();
    }
}
```

Figure.cs:

```
using System;

namespace Lab2
{
    public abstract class Figure : 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)
        {
            Figure p = (Figure) obj;

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

Circle.sc:

```
using System;

namespace Lab2
{
    public class Circle : Figure, IPrint
    {
```

```
        public Circle(double radius)
        {
            Type = "Круг";
            Radius = radius;
        }

        public double Radius { get; private set; }

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

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

Rectangle.cs:

```
using System;

namespace Lab2
{
    public class Rectangle : Figure, IPrint
    {
        public Rectangle(double height, double width)
        {
            Type = "Прямугольник";
            Height = height;
            Width = width;
        }

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

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

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

Square.cs:

```
using System;

namespace Lab2
{
    public class Square : Rectangle, IPrint
    {
        public Square(double side) : base(side, side)
        {
            Type = "Квадрат";
        }
    }
}
```

SimpleList.cs:

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

```
namespace Lab3
{
    public class SimpleList<T> : IEnumerable
        where T : IComparable
    {
        protected SimpleListItem<T> first = null;
        protected SimpleListItem<T> last = null;

        public int Count { get; protected set; } = 0;

        public void Add(T data)
        {
            if (first == null)
            {
                first = new SimpleListItem<T>(data);
                last = first;
            }
            else
            {
                last.next = new SimpleListItem<T>(data);
                last = last.next;
            }
            Count++;
        }

        public SimpleListItem<T> GetItem(int n)
        {
            if (n < 0 || n >= Count)
            {
                throw new IndexOutOfRangeException($"Current index: {n} is out of List's range");
            }

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

            while (i < n)
            {
                current = current.next;
                i++;
            }

            return current;
        }

        public T Get(int n)
        {
            return this.GetItem(n).data;
        }

        public IEnumerator GetEnumerator()
        {
            var current = first;
            while (current != null)
            {
                yield return current.data;
                current = current.next;
            }
        }

        public void Sort()
        {
            this.Sort(0, 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;
        }
    }
}
```

SimleListItem.cs:

```
namespace Lab3
{
    public class SimpleListItem<T>
    {
        public T data { get; set; }

        public SimpleListItem<T> next { get; set; }

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

SimpleStack.cs:

```
using System;

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

        public T Pop()
        {
            var data = default(T);

            if (Count == 0)
            {
                return data;
            }
            else if (Count == 1)
            {
                data = last.data;
```

```
                first = last = null;
            }
            else
            {
                var newLast = this.GetItem(Count - 2);
                data = last.data;
                newLast.next = null;
                last = newLast;
            }

            Count--;
            return data;
        }

    }
}
```

IMatrixCheckEmpty.cs:

```
namespace Lab3
{
    /// <summary>
    /// Проверка пустого элемента матрицы
    /// </summary>
    public interface IMatrixCheckEmpty<T>
    {
        /// <summary>
        /// Возвращает пустой элемент
        /// </summary>
        T getEmptyElement();
        /// <summary>
        /// Проверка что элемент является пустым
        /// </summary>
        bool checkEmptyElement(T element);
    }
}
```

FigureMatrixCheckEmpty.cs:

```
using Lab2;

namespace Lab3
{
    public class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>
    {
        /// <summary>
        /// В качестве пустого элемента возвращается null
        /// </summary>
        public Figure getEmptyElement()
        {
            return null;
        }
        /// <summary>
        /// Проверка что переданный параметр равен null
        /// </summary>
        public bool checkEmptyElement(Figure element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }
    }
}
```

SparseMatrix.cs:

```
using System;
```

```
using System.Collections.Generic;
using System.Runtime.Serialization.Formatters;
using System.Text;
using System.Linq;
namespace Lab3
{
    public class Matrix3D<T>
    {
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        int maxX;
        int maxY;
        int maxZ;
        IMatrixCheckEmpty<T> checkEmpty;

        public Matrix3D(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]
        {
            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();
                }
            }
            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()
        {
            //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++)
            {
                b.Append($"\nZ: {k}\n");
                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 (checkEmpty.checkEmptyElement(this[i, j, k]) != true)
                        {
                            b.Append(this[i, j, k].ToString());
                        }
                        else
                        {
                            b.Append(" - ");
                        }
                    }

                    b.Append("]\n");
                }
            }

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

Примеры:

```
Круг площадью 78.5398163397448
Прямугольник площадью 20
Квадрат площадью 25

ArrrayList после сортировки
Прямугольник площадью 20
Квадрат площадью 25
Круг площадью 78.5398163397448

List
Круг площадью 78.5398163397448
Прямугольник площадью 20
Квадрат площадью 25

List после сортировки
Прямугольник площадью 20
Квадрат площадью 25
Круг площадью 78.5398163397448

Матрица

Z: 0
[Прямугольник площадью 20        -       - ]
[ -      -       - ]
[ -      -       - ]

Z: 1
[ -      -       - ]
[ -     Квадрат площадью 25      - ]
[ -      -       - ]

Z: 2
[ -      -       - ]
[ -      -       - ]
[ -      -      Круг площадью 78.5398163397448]
```

```
Список
Квадрат площадью 25
Прямугольник площадью 20
Круг площадью 78.5398163397448

Сортировка списка
Прямугольник площадью 20
Квадрат площадью 25
Круг площадью 78.5398163397448

Стек
Круг площадью 78.5398163397448
Квадрат площадью 25
Прямугольник площадью 20
```