Федеральное государственное бюджетное образовательное учреждение

высшего образования «Московский государственный технический университет имени Н.Э. Баумана»
(национальный исследовательский университет)
(МГТУ им. Н.Э. Баумана)

Факультет Информатика и системы управления

Кафедра Системы обработки информации и управления

Лабораторная работа №3

Студент Родионов Д.А.

Группа ИУ5-35Б

Название дисциплины Базовые компоненты интернет-технологий

Преподаватель Гапанюк Ю.Е _____
Фамилия И.О. подпись

Москва 2020

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 Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») 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 (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ»). 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.

Текст программы:

Circle.cs

```
using System;


namespace Lab3
{
    class Circle : Figure, IPrint
    {
        public Circle(double radius = 0)
        {
```

```
            Radius = radius;
        }
        public double Radius { get; set; }

        public override string FigureName => "Круг";

        public override double Area() => Math.PI * Radius * Radius;

        public void Print() => Console.WriteLine(this.ToString());

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

    }
  }
```

Figure.cs

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

namespace Lab3
{
    abstract class Figure : IComparable
    {
        public string Type { get; protected set; }
        public abstract string FigureName { get; }
        public abstract double Area();
        public int CompareTo(object obj)
        {
            Figure f = (Figure)obj;
            if (this.Area() < f.Area()) return -1;
            else if (this.Area() == f.Area()) return 0;
            else return 1;
        }

    }
  }
```

IPrint.cs

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

namespace Lab3
{
    interface IPrint
    {
        void Print();
    }
}
```

Rectangle.cs

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

```
namespace Lab3
{
    class Rectangle : Figure, IPrint
    {
        public Rectangle(double height = 0, double width = 0)
        {
            Height = height;
            Width = width;
        }
        public double Height { get; set; }
        public double Width { get; set; }

        public override string FigureName => "Прямоугольник";

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

        public void Print() => Console.WriteLine(this.ToString());

        public override string ToString()
        {
            return $"{this.FigureName} с площадью {this.Area()} и высотой
{this.Height}, а шириной {this.Width}";
        }
    }
}
```

Square.cs

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

namespace Lab3
{
    class Square : Rectangle, IPrint
    {
        public Square(double side = 0)
        {
            Side = side;
        }
        public double Side { get; set; }

        public override string FigureName => "Квадрат";

        public override double Area()
        {
            return Side * Side;
        }

        public void Print() => Console.WriteLine(this.ToString());

        public override string ToString()
        {
            return $"{this.FigureName} с площадью {this.Area()} и стороной {Side}";
        }
    }
  }
```

Matrix.cs

```
using System;
```

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

namespace Lab3
{
    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 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"); }
        }
        string DictKey(int x, int y, int z) { return x.ToString() + "_" + y.ToString()
+ "_" + z.ToString(); }
        // i- слой(номер таблицы), j - строка, k - столбец
        public override string ToString()
        {
            StringBuilder b = new StringBuilder();
            for (int i = 0; i<this.maxZ; i++)
            {
                for (int j = 0; j < this.maxY; j++)
                {
                    b.Append("[");
                    for (int k = 0; k < this.maxX; 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("]\n");
                }
                b.Append("\n***\n\n");
            }
            return b.ToString();
        }
    }

}
```

SimpleList.cs

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

namespace Lab3
{
    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;
            }
        }
        // Чтение контейнера с заданным номером
        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;
        }
        // Чтение элемента с заданным номером
```

```
        public T Get(int number) { return GetItem(number).data; }
        // Для перебора коллекции
        public IEnumerator<T> GetEnumerator()
        {
            SimpleListItem<T> current = this.first;
            while (current != null)
            {
                yield return current.data;
                current = current.next;
            }
        }
        //Реализация обобщенного IEnumerator<T> требует реализации необобщенного
интерфейса
        System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{ return GetEnumerator(); }
        public void Sort() { Sort(0, this.Count - 1); }
        // Алгоритм быстрой сортировки
        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;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab3
{
    class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        public void Push(T element) { Add(element); }
        public T Pop()
        {
            //default(T) - значение по умолчанию (null для ссылочных типов, 0 для
числовых)
            T Result = default(T);
            if (this.Count == 0) return Result;
```

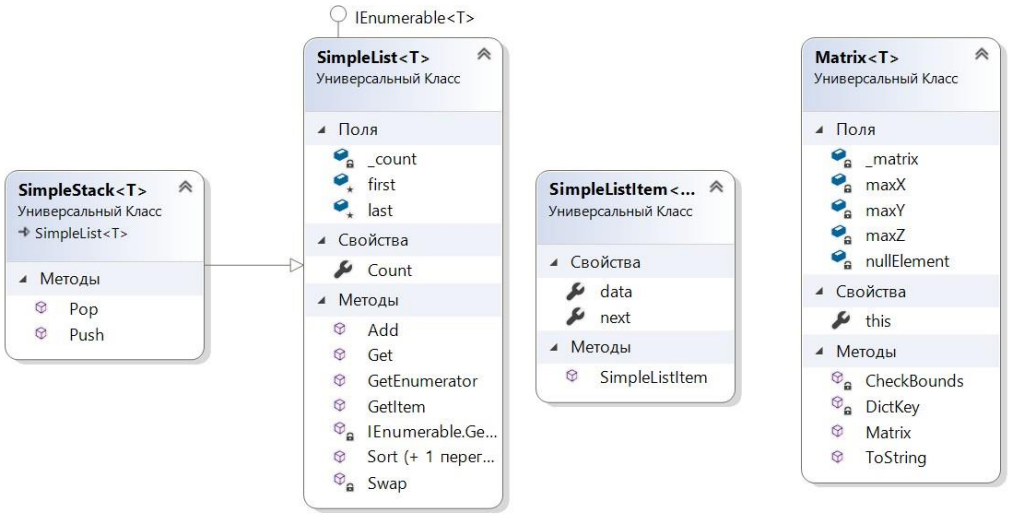
```
            if (this.Count == 1)
            {
                Result = this.first.data;
                this.first = null;
                this.last = null;
            }
            else
            {
                SimpleListItem<T> newLast = this.GetItem(this.Count - 2);
                Result = newLast.next.data;
                this.last = newLast;
                newLast.next = null;
            }
            this.Count--;
            return Result;
        }
    }

  }
```

Program.cs

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

namespace Lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            Circle cir = new Circle(10);
            Rectangle rec = new Rectangle(1, 3);
            Square sc = new Square(2);
            Circle cir2 = new Circle(100);

            //*** Test 1: ArrayList ***
            ArrayList al = new ArrayList();
            al.Add(cir);
            al.Add(rec);
            al.Add(sc);
            al.Add(cir2);

            Console.WriteLine("Test 1: ArrayList");
            Console.WriteLine("\nДо  сортировки:");
            foreach (Figure f in al) Console.WriteLine(f);
            al.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (Figure f in al) Console.WriteLine(f);

            //*** Test 2: List<T> ***
            List<Figure> list = new List<Figure>();
            list.Add(sc);
            list.Add(rec);
            list.Add(cir);
            list.Add(cir2);
```

```
            Console.WriteLine("\n\nTest 2: List<T>");
            Console.WriteLine("\nДо сортировки:");
            foreach (Figure f in list) Console.WriteLine(f);
            list.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (Figure f in list) Console.WriteLine(f);

            //*** Test 3: Matrix<T> ***
            Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 3, new
FigureMatrixCheckEmpty());
            matrix[0, 0, 0] = rec;
            matrix[1, 1, 1] = sc;
            matrix[2, 2, 2] = cir;
            matrix[2, 0, 2] = cir2;
            Console.WriteLine("\n\nTest 3: Matrix<T>\n");
            Console.WriteLine(matrix.ToString());

            //*** Test 4: SimpleList<T> ***
            SimpleList<Figure> simple_list = new SimpleList<Figure>();
            simple_list.Add(cir);
            simple_list.Add(rec);
            simple_list.Add(cir2);
            simple_list.Add(sc);
            Console.WriteLine("Test 4: SimpleList<T>");
            Console.WriteLine("\nДо сортировки:");
            foreach (var x in simple_list) Console.WriteLine(x);
            simple_list.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in simple_list) Console.WriteLine(x);

            //*** Test 5: SimpleStack<T> ***
            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            stack.Push(rec);
            stack.Push(sc);
            stack.Push(cir2);
            stack.Push(cir);
            Console.WriteLine("\nTest 5: SimpleStack<T>\n");
            while (stack.Count > 0)
            {
                Figure f = stack.Pop();
                Console.WriteLine(f);
            }

            Console.ReadKey();
        }
    }
}
```

Диаграмма классов:



Результаты программы: