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

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:

студент группы ИУ5-31

Стрихар Павел

Подпись и дата:

Проверил:

преподаватель каф. ИУ5

Гапанюк Ю. Е.

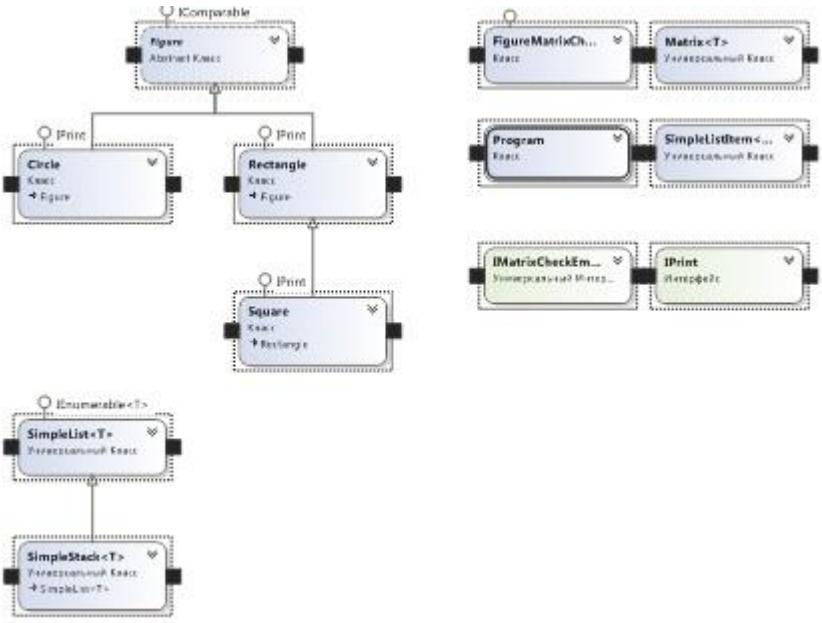
Подпись и дата:

Москва, 2020 г.

Описание задания:

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:



Код программы:

Program.cs

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

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

    public abstract class Figure : IComparable
    {
        public abstract double getSquare();

        public int CompareTo(object obj)
        {
            Figure o = obj as Figure;

            if (o != null)
            {
                return this.getSquare().CompareTo(o.getSquare());
            }
            else
                throw new Exception("Данный класс не соответствует классу Figure");
        }
    }

    class Rectangle : Figure, IPrint
    {
        protected double length, width;

        public Rectangle(double length, double width)
        {
            this.length = length;
            this.width = width;
        }
        public override double getSquare()
        {
            return length * width;
        }
        public override string ToString()
        {
            return "\nПрямоугольник\nВысота: " + this.length + "\nШирина: " + this.width +
"\nПлощадь: " + this.getSquare() + "\n";
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
```

```
    }
}

class Square : Rectangle, IPrint
{
    public Square(double length) : base(length, length) { }
    public override string ToString()
    {
        return "\nКвадрат\nСторона: " + this.length + "\nПлощадь: " + this.getSquare() + "\n";
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}

class Circle : Figure, IPrint
{
    private double r;

    public Circle(double r)
    {
        this.r = r;
    }
    public override double getSquare()
    {
        return Math.PI * r * r;
    }
    public override string ToString()
    {
        return "\nКруг\nРадиус: " + this.r + "\nПлощадь: " + this.getSquare() + "\n";
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}

/// <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);
}

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;
    }
}

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 в глубину (максимальное количество строк)
    /// </summary>
    int maxZ;

    IMatrixCheckEmpty<T> checkEmpty;

    /// <summary>
    /// Конструктор
    /// </summary>
    public Matrix(IMatrixCheckEmpty<T> checkEmptyParam, int px, int py, int pz = 0)
    {
        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=" + y + " 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++)
        {
            b.Append($"Matrix layer #\t{k}:\n");
            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\n");
        }
        return b.ToString();
    }
}

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;
    }
}

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>
    /// <param name="element"></param>
    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 обощенного 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);
```

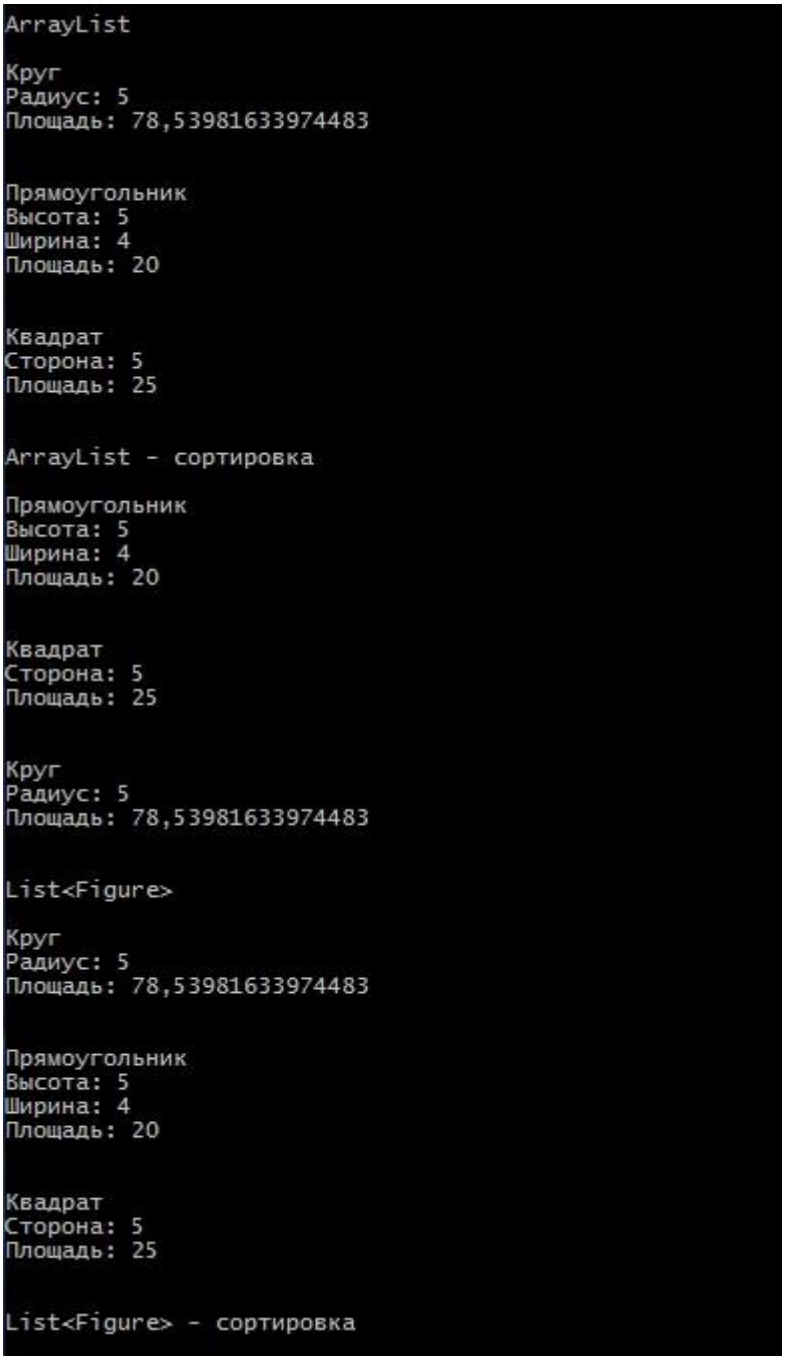
```
    }
    /// <summary>
    /// Реализация алгоритма быстрой сортировки
    /// </summary>
    /// <param name="low"></param>
    /// <param name="high"></param>
    private void Sort(int low, int high)
    {
        int i = low;
        int j = high;
        T x = Get((low + high) / 2);
        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;
    }
}

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

    public T Pop()
    {
        if (this.Count == 0)
            return default(T);

        if (this.Count == 1)
        {
            this.first = null;
            T pop = this.last.data;
            this.last = null;
            this.Count = 0;
```

```
            return pop;
        }
        else
        {
            T pop = this.last.data;
            SimpleListItem<T> last = this.GetItem(this.Count - 2);
            this.last = last;
            last.next = null;
            this.Count--;
            return pop;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        Rectangle rect = new Rectangle(5, 4);

        Square square = new Square(5);

        Circle circle = new Circle(5);

        Console.WriteLine("\nArrayList");

        ArrayList al = new ArrayList();

        al.Add(circle);

        al.Add(rect);

        al.Add(square);

        foreach (var x in al) Console.WriteLine(x);

        Console.WriteLine("\nArrayList - сортировка");

        al.Sort();

        foreach (var x in al) Console.WriteLine(x);

        Console.WriteLine("\nList<Figure>");

        List<Figure> fl = new List<Figure>();

        fl.Add(circle);

        fl.Add(rect);

        fl.Add(square);

        foreach (var x in fl) Console.WriteLine(x);

        Console.WriteLine("\nList<Figure> - сортировка");
```

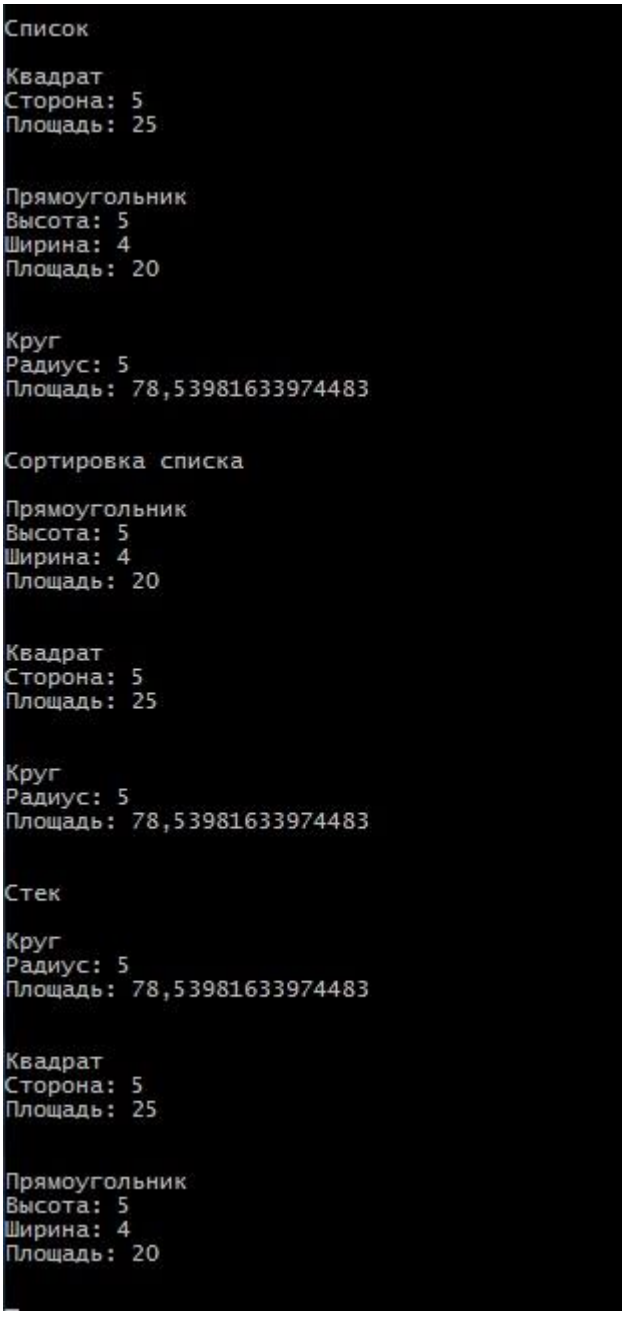
```
        fl.Sort();
        foreach (var x in fl) Console.WriteLine(x);
        Console.WriteLine("\nМатрица");
        Matrix<Figure> cube = new Matrix<Figure>(new FigureMatrixCheckEmpty(), 3, 3, 3);
        cube[0, 0, 0] = rect;
        cube[1, 1, 1] = square;
        cube[2, 2, 2] = circle;
        Console.WriteLine(cube.ToString());
        Console.WriteLine("\nСписок");
        SimpleList<Figure> list = new SimpleList<Figure>();
        list.Add(square);
        list.Add(rect);
        list.Add(circle);
        foreach (var x in list) Console.WriteLine(x);
        list.Sort();
        Console.WriteLine("\nСортировка списка");
        foreach (var x in list) 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);
        }
        Console.ReadLine();
    }
}
```

}

Пример выполнения программы:



```
List<Figure> - сортировка

Прямоугольник
Высота: 5
Ширина: 4
Площадь: 20


Квадрат
Сторона: 5
Площадь: 25


Круг
Радиус: 5
Площадь: 78,53981633974483


Матрица
Matrix layer #  0:
[
Прямоугольник
Высота: 5
Ширина: 4
Площадь: 20
          -       - ]
[ -       -       - ]
[ -       -       - ]


Matrix layer #  1:
[ -       -       - ]
[ -
Квадрат
Сторона: 5
Площадь: 25
          - ]
[ -       -       - ]


Matrix layer #  2:
[ -       -       - ]
[ -       -       - ]
[ -       -
Круг
Радиус: 5
Площадь: 78,53981633974483
]
```


Список
Квадрат
Сторона: 5
Площадь: 25
Прямоугольник
Высота: 5
Ширина: 4
Площадь: 20
Круг
Радиус: 5
Площадь: 78,53981633974483
Сортировка списка
Прямоугольник
Высота: 5
Ширина: 4
Площадь: 20
Квадрат
Сторона: 5
Площадь: 25
Круг
Радиус: 5
Площадь: 78,53981633974483
Стек
Круг
Радиус: 5
Площадь: 78,53981633974483
Квадрат
Сторона: 5
Площадь: 25
Прямоугольник
Высота: 5
Ширина: 4
Площадь: 20