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

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №3
«Трек курса «Функциональное программирование»»

Выполнил:
студент группы ИУ5-34
Драгун И.А.
Подпись и дата:

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

Москва, 2020 г.

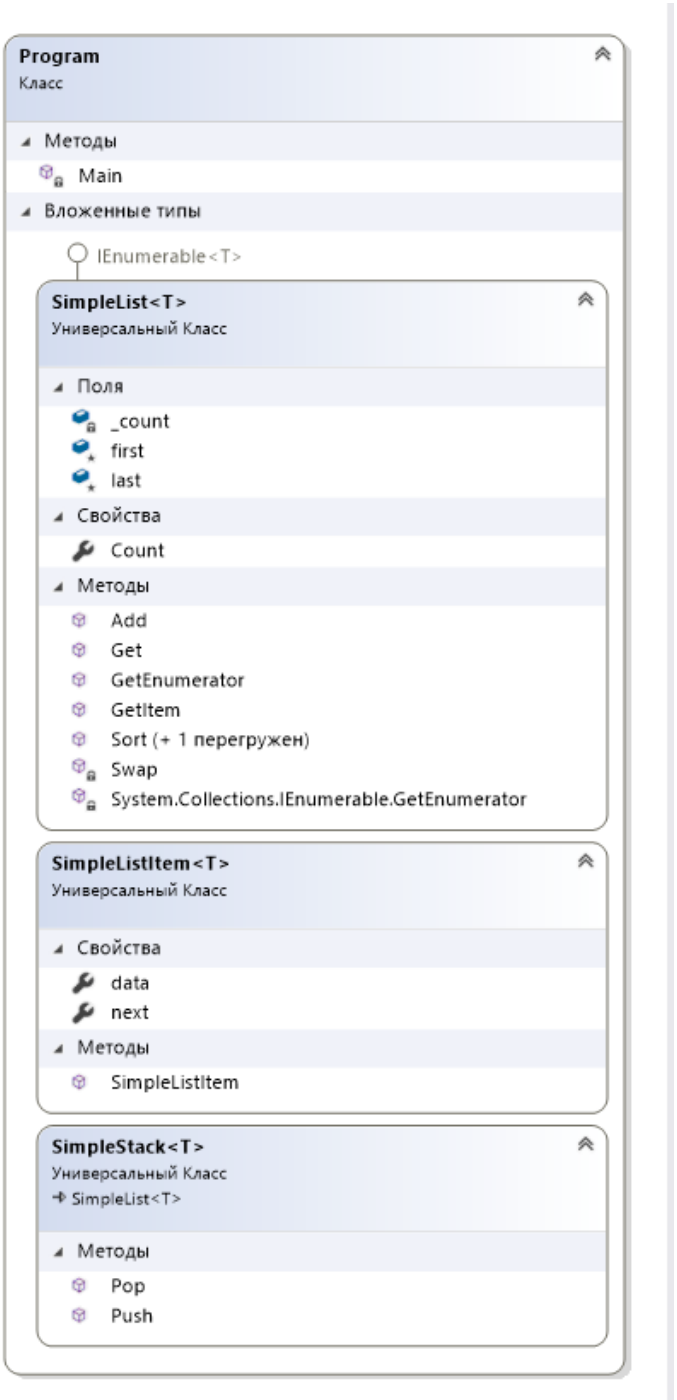
Описание задания

Разработать программу, реализующую работу с коллекциями.

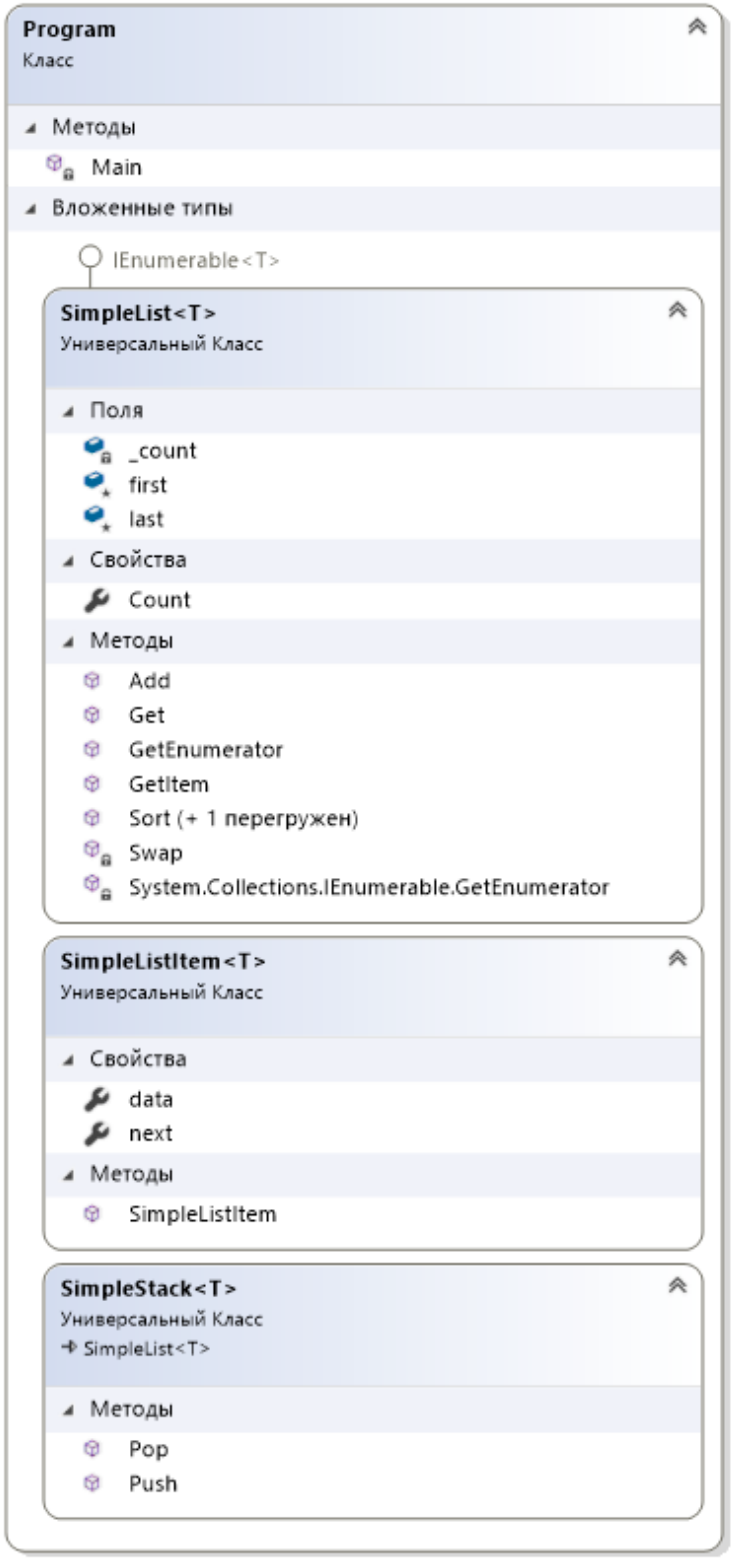
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится 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.

Диаграммы классов

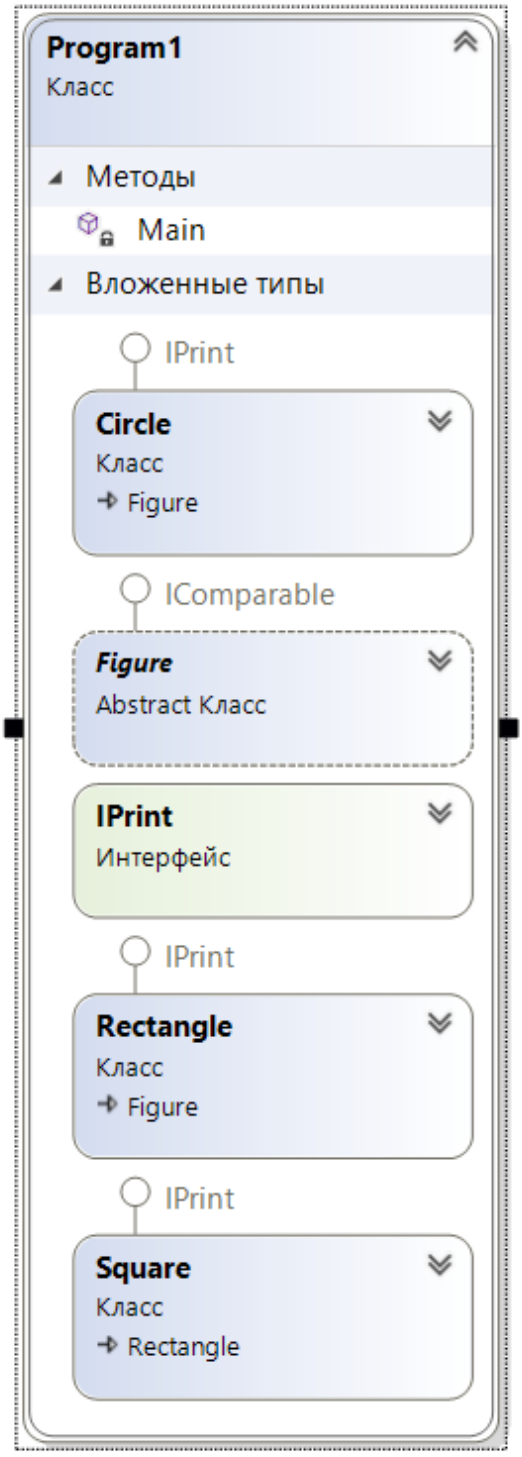
1. Для проекта lab3_1:



2. Для проекта SparseMatrix:



3. Для проекта lab3_2:



Текст программы

1. Для проекта lab3_1:

```
using System;
using System.Collections.Generic;
using static SparseMatrix.Program1;

namespace SparseMatrix
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            SimpleList<Figure> list = new SimpleList<Figure>();
            Program1.Figure rect = new Program1.Rectangle(3, 9);
            Program1.Figure square = new Program1.Square(7);
            Program1.Figure circle = new Program1.Circle(6);
            list.Add(circle);
            list.Add(rect);
            list.Add(square);
            Console.WriteLine("\nПеред сортировкой:");
            foreach (var x in list) Console.WriteLine(x);
            //сортировка
            list.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in list) Console.WriteLine(x);


            SimpleStack<Figure> stack = new SimpleStack<Figure>();
            //добавление данных в стек
            stack.Push(rect);
            stack.Push(square);
            stack.Push(circle);
            //чтение данных из стека
            Console.WriteLine(" ");
            Console.WriteLine("Чтение днных из стека после добавоения новых элементов");

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


        }


        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>//верширшина стека-конец списка
 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;
        //Присоединенный 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;
            }
        }

        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;
            }
        }

    }

}
```

2. Для проекта lab3_2:

```
using System;
using System.Collections.Generic;
using System.Text;
using static SparseMatrix.Program1;

namespace SparseMatrix
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("\nМатрица");
            Matrix<Program1.Figure> matrix = new Matrix<Program1.Figure>(3, 3,3, new
FigureMatrixCheckEmpty());
            Program1.Figure rect = new Program1.Rectangle(3, 9);
            Program1.Figure square = new Program1.Square(7);
            Program1.Figure circle = new Program1.Circle(6);
            matrix[0, 0, 0] = rect;
            matrix[1, 1, 1] = square;
            matrix[2, 2, 2] = circle;
            Console.WriteLine(matrix.ToString());
        }

        public class Matrix<T>
        {
            Dictionary<string, T> _matrix = new Dictionary<string, T>();
         int maxX;
         int maxY;
         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)//внедрение зависимостей через контруктор
            {
                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()
        {
            StringBuilder b = new StringBuilder();


            for ( int z = 0; z < this.maxZ; z++)
            {
                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, z]))
                        {
                            //Добавить приведенный к строке текущий элемент
                            b.Append(this[i, j, z].ToString());
                        }
```

```
                        else
                        {
                            //Иначе добавить признак пустого значения
                            b.Append(" - ");
                        }
                    }// i
                    b.Append("]\n");

                }//j

            }//z
            return b.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;
        }
    }//тут*/

}

}
```

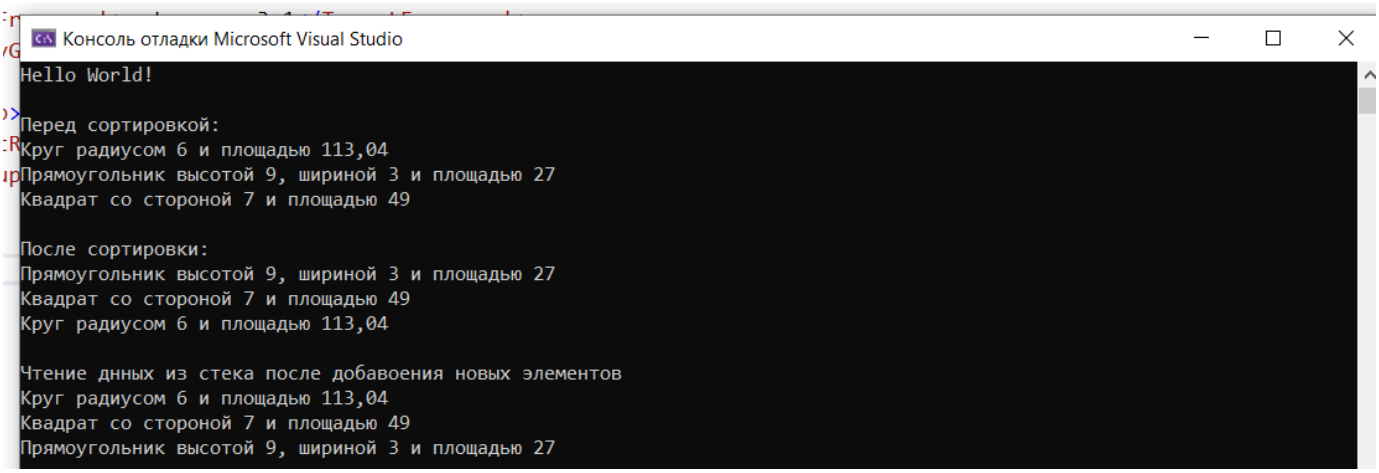
3. Для проекта Лаб 3 Светашева lab3_3:

```
class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Rectangle Pr = new Rectangle(2, 3);
            Square S = new Square(4);
            Circle C = new Circle(2);
            Console.WriteLine("Сортировка коллекции класса ArrayList");
            ArrayList arrayListF = new ArrayList(3) { Pr, S, C };
            Console.WriteLine("\nПеред сортировкой:");
            foreach (var x in arrayListF) Console.WriteLine(x);
            arrayListF.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in arrayListF) Console.WriteLine(x);
            List<GeomFig> listF = new List<GeomFig>();
            listF.Add(Pr);
            listF.Add(S);
            listF.Add(C);
            Console.WriteLine("Сортировка коллекции класса List<GeomFig>");
            Console.WriteLine("\nПеред сортировкой:");
            foreach (var x in listF) Console.WriteLine(x);
            //сортировка
            listF.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in listF) Console.WriteLine(x);

            Console.WriteLine("*****************");



        }

    }

    interface IPrint
    {
        void Print();
    }

    abstract class GeomFig: IComparable
    {
        public abstract double CalculateArea();
        public string Type { get; set; }
        public override string ToString()
        {
            return this.Type + "площадью" + this.CalculateArea().ToString();
        }

        public int CompareTo(object obj)
        {
            //Приведение параметра к типу "фигура"
            GeomFig p = (GeomFig)obj;
            //Сравнение
            if (this.CalculateArea() < p.CalculateArea()) return -1;
            else if (this.CalculateArea() == p.CalculateArea()) return 0;
            else return 1; //(this.Area() > p.Area())
        }

    }
    class Rectangle : GeomFig, IPrint
```

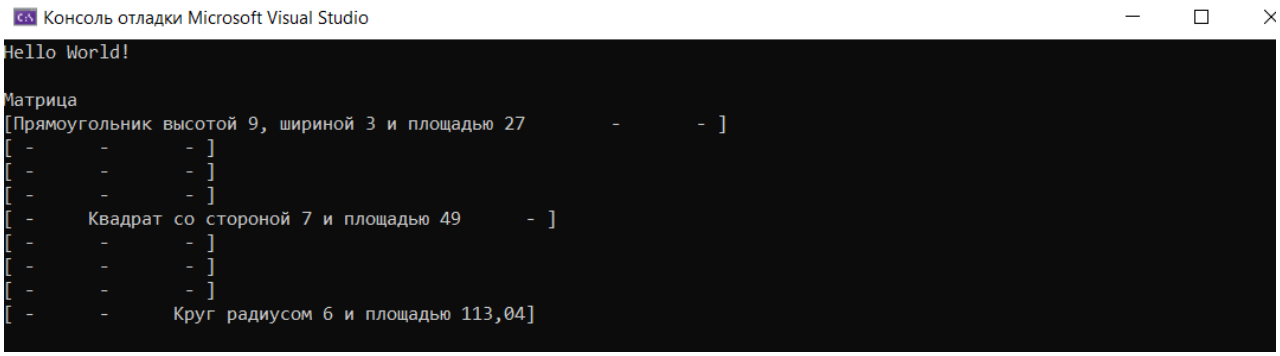
```
    {
        public double width;
        public double height;
        public Rectangle(double w, double h)
        {
            width = w;
            height = h;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
        public override string ToString()
        {
            return "Ширина прямоугольника = " + this.width + " Высота прямоугольника = "
+ this.height + "; Площадь прямоугольника = " + this.CalculateArea().ToString();
        }
        public override double CalculateArea()
        {
            return this.width * this.height;
        }
    }
    class Square : Rectangle, IPrint
    {
        public Square(double l) : base(l, l)
        {
            width = l;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
        public override string ToString()
        {
            return "Длина стороны квадрата = " + this.width + "; Площадь квадрата = " +
this.CalculateArea().ToString();
        }
    }
    class Circle : GeomFig, IPrint
    {
        public double radius;
        public Circle(double r)
        {
            radius = r;
        }
        public void Print()
        {
            Console.WriteLine(this.ToString());
        }
        public override string ToString()
        {
            return "Радиус круга = " + this.radius + "; Площадь круга = " +
this.CalculateArea().ToString();
        }
        public override double CalculateArea()
        {
            return this.radius * this.radius * Math.PI;
        }
    }
                                            }
```

Экранные формы с примерами выполнения программы

1. Для проекта lab3_1:



2. Для проекта SparseMatrix:



3. Для проекта lab3_2: