Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3
“ Разработка программы, реализующую работу с коллекциями”

Выполнил:
студент группы ИУ5-33Б
Ларионова Амина
Подпись и дата:
15.11.20

Проверил:
Подпись и дата:

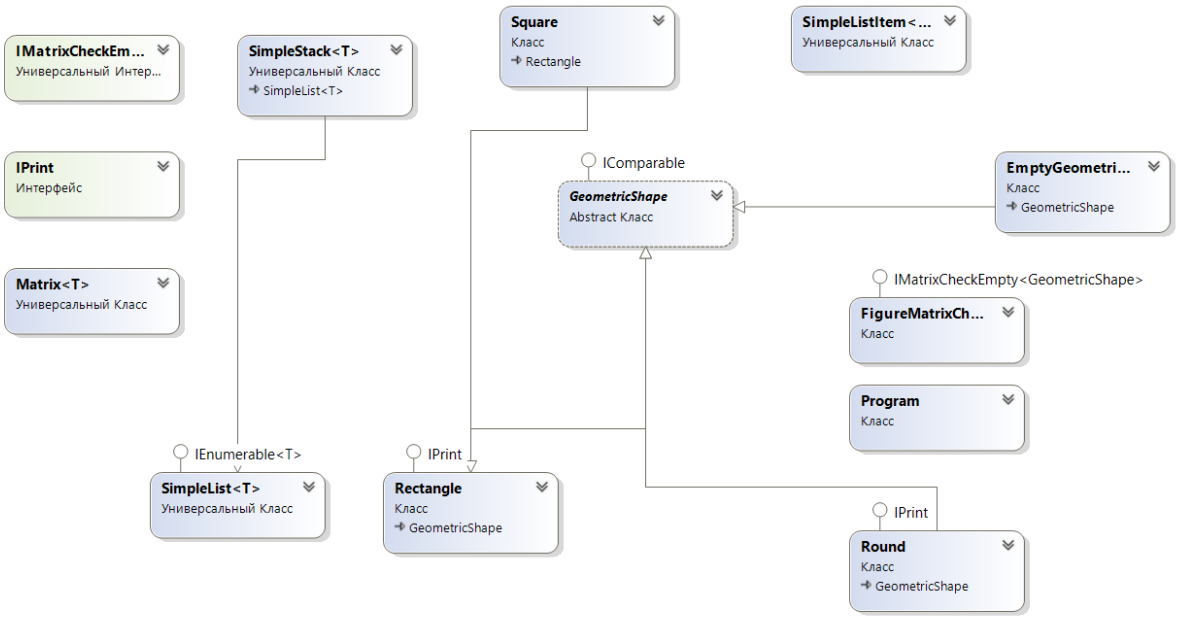
Москва, 2020 г.

1) **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.

2) **Диаграмма классов**



3) Текст программы

Program.cs

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

namespace Лаб_3_2_к
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Ларионова Амина Павловна ИУ5-33Б\n");
            Rectangle Rec = new Rectangle(8, 20);
            Square Sq = new Square(26);
            Round R = new Round(18);
            //////
            Console.WriteLine("\nArrayList");
            ArrayList al = new ArrayList();
            al.Add(R);
            al.Add(Rec);
            al.Add(Sq);

            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<GeometricShape>");
            List<GeometricShape> fl = new List<GeometricShape>();
            fl.Add(R);
            fl.Add(Rec);
```

```
            fl.Add(Sq);

            Console.WriteLine("\nПеред сортировкой:");
            foreach (var x in fl) Console.WriteLine(x);

            //сортировка
            fl.Sort();

            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in fl) Console.WriteLine(x);

            ////////////////////////////////

            Console.WriteLine("\nМатрица");
            Matrix<GeometricShape> matrix = new Matrix<GeometricShape>(3, 3, new
FigureMatrixCheckEmpty());
            matrix[0, 0] = Rec;
            matrix[1, 1] = Sq;
            matrix[2, 2] = R;
            Console.WriteLine(matrix.ToString());

            //Выход за границы индекса и обработка исключения
            try
            {
                GeometricShape temp = matrix[123, 123];
            }
            catch (ArgumentOutOfRangeException e)
            {
                Console.WriteLine(e.Message);
            }

            ////////////////////////////////
            Console.WriteLine("\nСписок");
            SimpleList<GeometricShape> list = new SimpleList<GeometricShape>();
            list.Add(R);
            list.Add(Rec);
            list.Add(Sq);
            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);

            ////////////////////////////////
            Console.WriteLine("\nСтек");

            SimpleStack<GeometricShape> stack = new SimpleStack<GeometricShape>();
            //добавление данных в стек
            stack.Push(Rec);
            stack.Push(Sq);
            stack.Push(R);
            //чтение данных из стека
            while (stack.Count > 0)
            {
                GeometricShape f = stack.Pop();
                Console.WriteLine(f);
            }

            Console.ReadLine();

        Console.ReadKey();

        }
    }
```

```
}
```

Iprint.cs

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

namespace Лаб_3_2_к
{
    interface IPrint
    {
        void Print();
    }
       }
```

FigureMartixCheckEmpty.cs

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

namespace Лаб_3_2_к
{
    class FigureMatrixCheckEmpty:IMatrixCheckEmpty<GeometricShape>//создаем класс
реализующий интерфейс IMatrixCheckEmpty
    {

       // В качестве пустого элемента возвращается null
        public GeometricShape getEmptyElement()
        {
            return null;
        }
        /// Проверка что переданный параметр равен null
        public bool checkEmptyElement(GeometricShape element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }

    }
}
```

EmptyGeometricShape.cs

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

```
namespace Лаб_3_2_к
{
    class EmptyGeometricShape:GeometricShape
    {
        public override double Area()
        {
            return 0;
        }
    }
}
```

GeometricShape.cs

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

namespace Лаб_3_2_к
{
    abstract class GeometricShape : IComparable
    {
        public virtual double Area() { return 0; }

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

        public string Type // Тип фигуры
        {
            get
            {
                return this._Type;
            }
            protected set
            {
                this._Type = value;
            }
        }
        string _Type;
        public override string ToString() // Приведение к строке, переопределение метода
Object
        {
            return this.Type + " площадью " + this.Area().ToString();
        }
    }
}
```

IMatrixCheckEmpty.cs

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

namespace Лаб_3_2_к
```

```
{

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

Matrix.cs

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

namespace Лаб_3_2_к
{
    public class Matrix<T>
    {

        /// Словарь для хранения значений
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// Количество элементов по горизонтали (максимальное количество столбцов)
        int maxX;

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

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

        /// Конструктор
        public Matrix(int px, int py, IMatrixCheckEmpty<T> checkEmptyParam)//пеоедаем
размер матрицы и объект класса,реализ. интерфейс IMatrixCheckEmpty<T>
        {
            this.maxX = px;
            this.maxY = py;
            this.checkEmpty = checkEmptyParam;
        }

        /// Индексатор для доступа к данных
        public T this[int x, int y]
        {
            set//запись элемента в матрицу
            {
                CheckBounds(x, y);//проверка границ
                string key = DictKey(x, y);//вычисляется ключ
                this._matrix.Add(key, value);//происходит запись значения value в словарь
_matrix
            }
            get//чтение элемента из матрицы
            {
                CheckBounds(x, y);
                string key = DictKey(x, y);
                //проверка существования ключа в словаре
                if (this._matrix.ContainsKey(key))//если элемент с вычесленным ключом
существует
                {
```

```
                    return this._matrix[key];//возвращается значение элемента
                }
                else//если не существует
                {
                    return this.checkEmpty.getEmptyElement();//возвращается пустое
значение
                }
            }
        }

        /// Проверка границ
        void CheckBounds(int x, int y)
        {
            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");
            }
        }

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

        /// Приведение к строке
        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 j = 0; j < this.maxY; j++)
            {
                //метод Append() позволяетдописать в конец внутренней строки
StringBuilder практически любые данные
                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]))
                    {
                        //Добавить приведенный к строке текущий элемент
                        b.Append(this[i, j].ToString());
                    }
                    else
                    {
                        //Иначе добавить признак пустого значения
                        b.Append(" - ");
                    }
                }
                b.Append("]\n");
```

```
            }
            return b.ToString();


        }
    }
}
```

Rectangle.cs

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


namespace Лаб_3_2_к
{
    class Rectangle : GeometricShape, IPrint
    {
        private float width, height;//ширина и высота
        public Rectangle(float width, float height)
        {
            this.width = width;
            this.height = height;
        }
        public Rectangle() { }
        public void SetValues(float width, float height)
        {
            this.width = width;
            this.height = height;
        }
        public double GetValueW()
        {
            return width;
        }
        public double GetValueH()
        {
            return height;
        }
        public override double Area()
        {
            return width * height;
        }
        public override string ToString()
        {
            return "Ширина= " + width.ToString() + " Высота= " + height.ToString() + "
Площадь= " + Area().ToString();
        }
        public void Print()
        {
            Console.WriteLine(ToString());
        }
    }
}
```

Round.cs

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

```
using System.Threading.Tasks;
using static System.Math;

namespace Лаб_3_2_к
{
    class Round : GeometricShape, IPrint
    {
        int radius;
        public Round(int radius)
        {
            this.radius = radius;
        }
        public override double Area()
        {
            return Pow(radius, 2) * PI;
        }
        public override string ToString()
        {
            return "Радиус= " + radius.ToString() + " Площадь= " + Area().ToString();
        }
        public void Print()
        {
            Console.WriteLine(ToString());
        }
    }
}
```

SimpleList.cs

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

namespace Лаб_3_2_к
{

        /// <summary>
        /// Список
        /// </summary>
        public class SimpleList<T> : IEnumerable<T>
            where T : IComparable
        {

        // Первый элемент списка
            protected SimpleListItem<T> first = null;//содержит ссылку на контейнер для
первого элемнта

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

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

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

```
            //добавление контейнера к цепочке контейнеров
                //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;
                }
            }

            /// Чтение(получение) контейнера с заданным номером
            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;
            }


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


        /// Для перебора коллекции
        ///данный метод осуществляет перебор всех элементов списка и их возврат с помощью
конструкции «yield return»
            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();
            }


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


        /// Алгоритм быстрой сортировки
        //метод Sort с параметрами применяется для внутренних рекурсивных вызовов
            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);
            }


            /// Вспомогательный метод для обмена элементов при сортировке
            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;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Лаб_3_2_к
{
    /// Элемент списка
    public class SimpleListItem<T>
```

```
    {
        /// Данные
        public T data { get; set; }

        /// Следующий элемент
        public SimpleListItem<T> next { get; set; }//next-аналог указателя на след
элемент

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

SimpleStac.cs

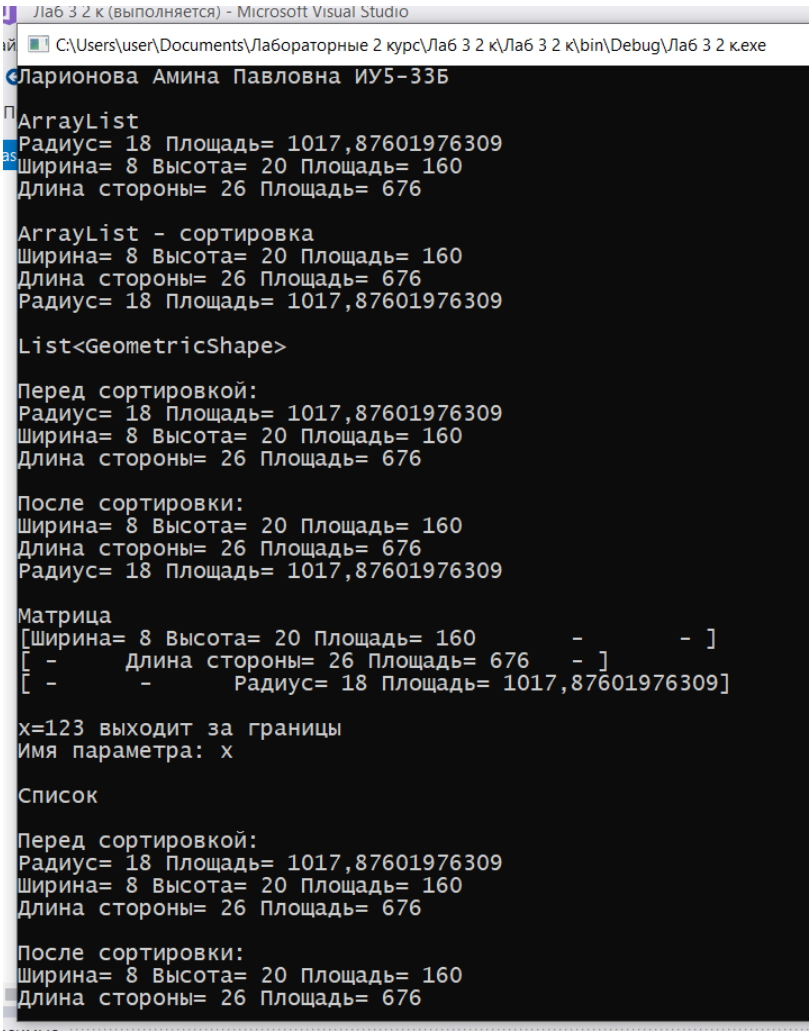
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Лаб_3_2_к
{

        /// Класс стек
        class SimpleStack<T> : SimpleList<T> where T : IComparable
        {
        //в данной реализации верщина стека-конец списка

            /// Добавление в стек
            public void Push(T element)
            {
                //Добавление в конец списка уже реализовано
                Add(element);
            }

            /// Удаление и чтение из стека
            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;
```

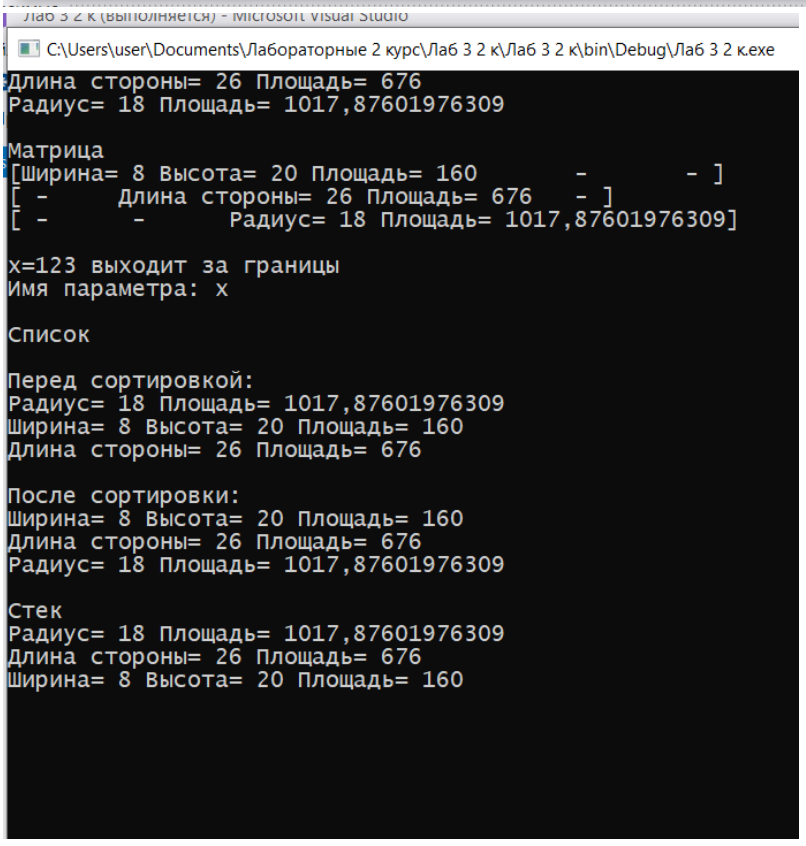
```
                    //последний элемент удаляется из списка
                    newLast.next = null;
                }
                //Уменьшение количества элементов в списке
                this.Count--;
                //Возврат результата
                return Result;
            }
        }

}
```

Square.cs

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


namespace Лаб_3_2_к
{
    class Square:Rectangle
    {
        int sidelength;
        public Square(int sidelength)
        {
            this.sidelength = sidelength;
        }
        public override double Area()
        {
            return sidelength * sidelength;
        }
        public override string ToString()
        {
            return "Длина стороны= " + sidelength.ToString() + " Площадь= " +
Area().ToString();
        }
        public void Set(int sidelength)
        {
            this.sidelength = sidelength;
        }
        public double Get()
        {
            return sidelength;
        }
        public new void Print()
        {
            Console.WriteLine(ToString());
        }
    }
}
```

4) Экранные формы с примерами выполнения программы


Лаб 3 2 к (выполняется) - Microsoft Visual Studio
C:\Users\user\Documents\Лабораторные 2 курс\Лаб 3 2 к\Лаб 3 2 к\bin\Debug\Лаб 3 2 к.exe
Ларионова Амина Павловна ИУ5-33Б
ArrayList
Радиус= 18 Площадь= 1017,87601976309
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
ArrayList - сортировка
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
Радиус= 18 Площадь= 1017,87601976309
List<GeometricShape>
Перед сортировкой:
Радиус= 18 Площадь= 1017,87601976309
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
После сортировки:
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
Радиус= 18 Площадь= 1017,87601976309
Матрица
[Ширина= 8 Высота= 20 Площадь= 160 - -]
[- Длина стороны= 26 Площадь= 676 -]
[- - Радиус= 18 Площадь= 1017,87601976309]
x=123 выходит за границы
Имя параметра: x
Список
Перед сортировкой:
Радиус= 18 Площадь= 1017,87601976309
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
После сортировки:
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676


Лаб 3 2 к (выполняется) - Microsoft Visual Studio
C:\Users\user\Documents\Лабораторные 2 курс\Лаб 3 2 к\Лаб 3 2 к\bin\Debug\Лаб 3 2 к.exe
Длина стороны= 26 Площадь= 676
Радиус= 18 Площадь= 1017,87601976309
Матрица
[Ширина= 8 Высота= 20 Площадь= 160 - -]
[- Длина стороны= 26 Площадь= 676 -]
[- - Радиус= 18 Площадь= 1017,87601976309]
x=123 выходит за границы
Имя параметра: x
Список
Перед сортировкой:
Радиус= 18 Площадь= 1017,87601976309
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
После сортировки:
Ширина= 8 Высота= 20 Площадь= 160
Длина стороны= 26 Площадь= 676
Радиус= 18 Площадь= 1017,87601976309
Стек
Радиус= 18 Площадь= 1017,87601976309
Длина стороны= 26 Площадь= 676
Ширина= 8 Высота= 20 Площадь= 160