



**Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

**Базовые компоненты интернет технологий
Отчет по лабораторной работе №3**

Студент: Дубянский А.И.
Группа: ИУ5Ц-51Б

Преподаватель: Гапанюк Ю. Е.

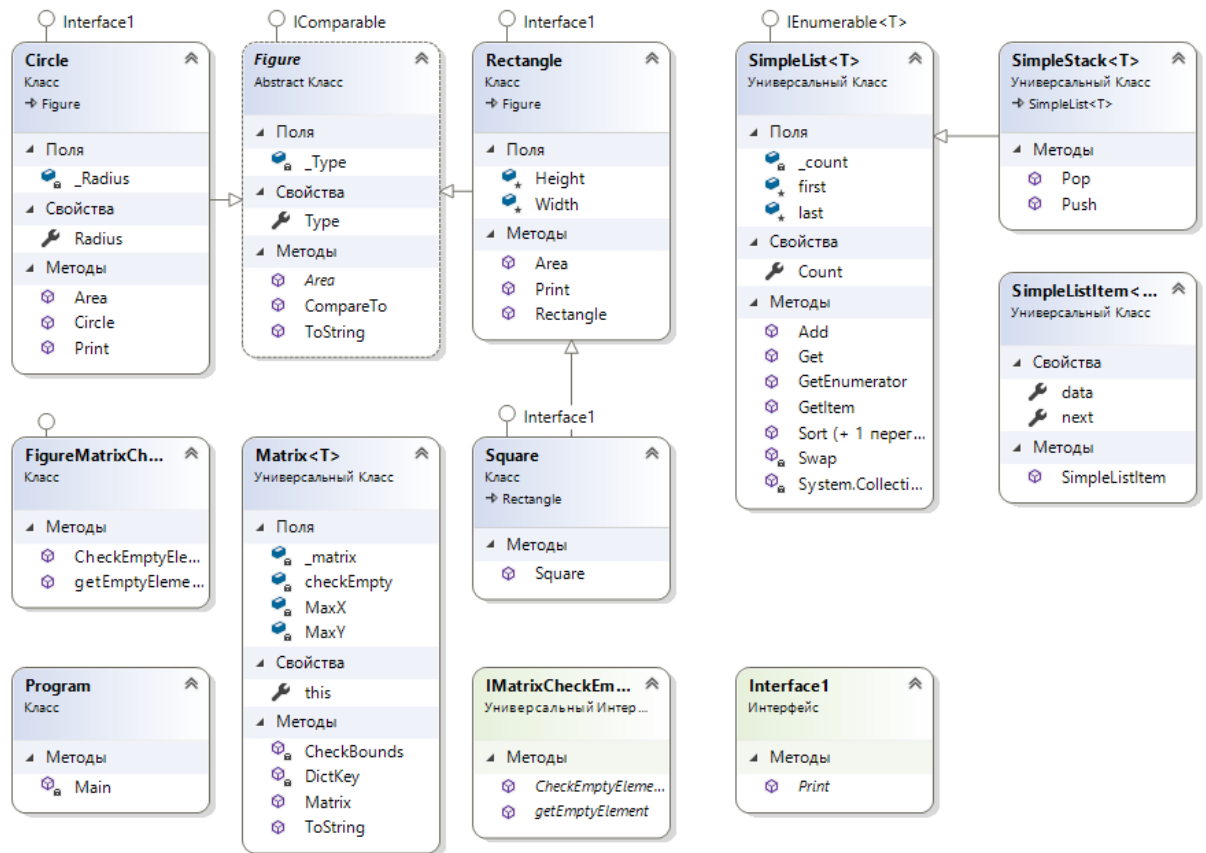
2019 г.

Лабораторная работа №3

Разработать программу, реализующую работу с коллекциями.

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Объекты классов «Прямоугольник», «Квадрат», «Круг» использовать из проекта лабораторной работы №2.
3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса `Comparable`. Сортировка производится по площади фигуры.
4. Создать коллекцию класса `ArrayList`. Сохранить объекты (типы) Прямоугольник, Квадрат, Круг, в коллекцию. Вывести в цикле содержимое площади элементов 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



Текст программы

Program.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//using LAB_3_Bulygina;

namespace Lab_3
{
    class Program
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Лабораторная работа №3");
            Console.Title = "Дубянский А. И., ИУ5Ц-51Б";

            // Объект класса Rectangle
            Rectangle rect = new Rectangle(2, 4);
            rect.Print();

            // Объект класса
            Square square = new Square(5);
            square.Print();

            // Объект класса
            Circle circle = new Circle(3);
            circle.Print();

            //коллекция класса ArrayList
            ArrayList figures = new ArrayList();
            figures.Add(circle);
            figures.Add(rect);
            figures.Add(square);
            Console.WriteLine("\nДо сортировки для ArrayList");
            foreach (var i in figures)
            {
                Console.WriteLine(i);
            }
            figures.Sort();
            Console.WriteLine("\nПосле сортировки для ArrayList");
            foreach (var i in figures)
            {
                Console.WriteLine(i);
            }

            //коллекция класса List<Figure>
            List<Figure> figures1 = new List<Figure>();
            figures1.Add(circle); //добавление в коллекцию
            figures1.Add(rect);
            figures1.Add(square);
            Console.WriteLine("\n\nДо сортировки для List<Figure>:");
            foreach (var i in figures1)
            {
                Console.WriteLine(i);
            }
        }
    }
}
```

```

        Console.WriteLine("\nПосле сортировки для List<Figure>:");
        figures1.Sort();
        foreach (var i in figures1)
        {
            Console.WriteLine(i);
        }

        //создание разреженной матрицы
        Console.WriteLine("\n\nМатрица:");
        Matrix<Figure> matrix = new Matrix<Figure>(3, 3, new
FigureMatrixCheckEmpty());
        matrix[0, 0] = rect;
        matrix[1, 1] = square;
        matrix[2, 2] = circle;
        Console.WriteLine(matrix.ToString());

        //использование коллекции SimpleList
        SimpleList<Figure> list = new SimpleList<Figure>();
        list.Add(circle);
        list.Add(rect);
        list.Add(square);
        Console.WriteLine("\n\nПеред сортировкой (SimpleList):");
        foreach (var a in list)
        {
            Console.WriteLine(a);
        }
        list.Sort();
        Console.WriteLine("\n\nПосле сортировки (SimpleList):");
        foreach (var a in list)
        {
            Console.WriteLine(a);
        }

        //использование собственного стека
        SimpleStack<Figure> stack = new SimpleStack<Figure>();
        stack.Push(circle);
        stack.Push(rect);
        stack.Push(square);
        Console.WriteLine("\n\nИспользование стека:");
        while (stack.Count > 0)
        {
            Figure f = stack.Pop();
            Console.WriteLine(f);
        }

        Console.ReadKey();
    }
}
}

```

Figure.cs

```

namespace Lab_3
{
    abstract partial class Figure : IComparable
    {
        private string _Type;
        /// <summary>
        /// Название фигуры
        /// </summary>
        public string Type
        {
            get { return this._Type; }
            set { this._Type = value; }
        }
    }
}

```

```

    /// <summary>
    /// Вычисление площади
    /// </summary>
    /// <returns></returns>
    abstract public double Area();

    public override string ToString()
    {
        //Console.WriteLine(this.Type + ":");
        return this.Type + " с 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 -1;
        }
        else if (this.Area() == p.Area())
        {
            return 0;
        }
        else
        {
            throw new NotImplementedException();
        }
    }
}
}

```

Square.cs

```

namespace Lab_3
{
    class Square : Rectangle, Interface1
    {
        /// Конструктор
        /// <param name="width">длина стороны</param>
        public Square(double width) :
            base(width, width)
        { this.Type = "Квадрат"; }
    }
}

```

Rectangle.cs

```

namespace Lab_3
{
    /// Класс Прямоугольник
    class Rectangle : Figure, Interface1
    {
        /// Ширина прямоугольника
        protected double Width;

        /// Высота прямоугольника
        protected double Height;
    }
}

```

```

    /// Конструктор
    /// <param name="width">Ширина</param>
    /// <param name="height">Высота</param>
    public Rectangle(double width, double height)
    {
        this.Width = width;
        this.Height = height;
        this.Type = "Прямоугольник";
    }

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

    public void Print()
    {
        Console.WriteLine("_____");
        Console.WriteLine("Высота четырехугольника: " + this.Height);
        Console.WriteLine("Ширина четырехугольника: " + this.Width);
        Console.WriteLine(this.ToString());
    }
}
}

```

Circle.cs

```

namespace Lab_3
{
    /// Класс Круг
    class Circle : Figure, Interface1
    {
        /// Радиус
        //protected double Radius;
        private double _Radius;
        public double Radius
        {
            get { return _Radius; }
            set { this._Radius = value; }
        }

        /// Конструктор
        public Circle(double radius)
        {
            this.Radius = radius;
            this.Type = "Круг";
        }

        /// Площадь Круга
        public override double Area()
        {
            return this.Radius * this.Radius * Math.PI;
        }

        ///// Переопределенный метод преобразования в строку
        //public override string ToString()
        //{
        //    return "Круг: радиус = " + this.Radius + "; площадь = " + this.Area();
        //}

        /// Метод вывода на консоль
        public void Print()
        {
            Console.WriteLine("_____");
        }
    }
}

```

```

        Console.WriteLine("Радиус круга: " + this.Radius);
        Console.WriteLine(this.ToString());
    }
}

```

Interface1.cs

```

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

namespace Lab_3
{
    interface Interface1
    {
        void Print();
    }
}

```

Matrix.cs

```

namespace Lab_3
{
    partial class Matrix<T>
    {
        ///<summary>
        /// Словарь для хранения значений
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        ///<summary>
        ///Макс. количество столбцов
        /// </summary>
        int MaxX;

        ///<summary>
        ///Макс.количество строк
        ///</summary>
        int MaxY;

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

        ///<summary>
        ///Конструктор
        /// </summary>
        public Matrix(int x, int y, IMatrixCheckEmpty<T> param)
        {
            this.MaxX = x;
            this.MaxY = y;
            this.checkEmpty = param;
        }

        ///<summary>
        ///Индексатор для доступа к данным
        /// </summary>
        public T this[int x, int y]
        {
            set
            {
                CheckBounds(x, y);
                string key = DictKey(x, y);
            }
        }
    }
}

```



```

        this._matrix.Add(key, value);
    }
    get
    {
        CheckBounds(x, y);
        string key = DictKey(x, y);
        if (this._matrix.ContainsKey(key))
        {
            return this._matrix[key];
        }
        else
        {
            return this.checkEmpty.getEmptyElement();
        }
    }
}

///

```

```

        {
            b.Append(" - ");
        }
        b.Append("]\n");
    }
    return b.ToString();
}
}
}

```

SimpleList.cs

```

namespace Lab_3
{
    ///<summary>
    ///Список
    /// </summary>
    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>
        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;
}
}
}

```

SimpleStack.cs

```

namespace Lab_3
{
    /// <summary>
    /// класс стек
    /// </summary>

    partial class SimpleStack<T> : SimpleList<T> where T : IComparable
    {
        /// <summary>
        /// добавление в стек
        /// </summary>
        public void Push(T element)
        {
            Add(element);
        }

        /// <summary>
        /// удаление и чтение из стека
        /// </summary>
        public T Pop()
        {
            //default - значение по умолчанию
            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
    {
        //поиск предпоследнего элемента
        SimpleListItem<T> newLast = this.GetItem(this.Count - 2);
        //чтение из последнего элемента
        Result = newLast.next.data;
        //предпоследний элемент считается последним
        this.last = newLast;
        //последний элемент удаляется
        newLast.next = null;
    }
    //уменьшение количества элементов в списке
    this.Count--;
    //возврат результата
    return Result;
}
}
}

```

SimpleListItem.cs

```

namespace Lab_3
{
    /// <summary>
    /// Элемент списка
    /// </summary>
    partial class SimpleListItem<T>
    {
        ///<summary>
        ///Данные
        ///</summary>
        public T data { get; set; }
        ///<summary>
        ///Следующий элемент
        /// </summary>
        public SimpleListItem<T> next { get; set; }

        /// <summary>
        /// конструктор
        /// </summary>
        public SimpleListItem(T param)
        {
            this.data = param;
        }
    }
}

```

FigureMatrixCheckEmpty.cs

```

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

namespace Lab_3
{
    partial class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>
    {
        //реализация первого метода интерфейса
        public Figure getEmptyElement()
        {
            return null;
        }
    }
}

```

```

        public bool CheckEmptyElement(Figure element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }
    }
}

```

IMatrixCheckEmpty.cs

```

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

namespace Lab_3
{
    //методы данного интерфейса используются при создании разреженной матрицы
    public interface IMatrixCheckEmpty<T>
    {
        //возвращает пустой элемент
        T getEmptyElement();
        //проверка, что элемент является пустым
        bool CheckEmptyElement(T element);
    }
}

```

Тест программы

```
Дубянский А. И., ИУ5Ц-51Б
Лабораторная работа №3

Высота четырехугольника: 4
Ширина четырехугольника: 2
Прямоугольник с площадью 8

Высота четырехугольника: 5
Ширина четырехугольника: 5
Квадрат с площадью 25

Радиус круга: 3
Круг с площадью 28,2743338823081
```

Сортировка через необобщенную коллекцию ArrayList с помощью стандартного метода Sort():

```
До сортировки для ArrayList
Круг с площадью 28,2743338823081
Прямоугольник с площадью 8
Квадрат с площадью 25

После сортировки для ArrayList
Прямоугольник с площадью 8
Квадрат с площадью 25
Круг с площадью 28,2743338823081
```

Сортировка с помощью IComparable. Обобщенная коллекция List

```
До сортировки для List<Figure>:
Круг с площадью 28,2743338823081
Прямоугольник с площадью 8
Квадрат с площадью 25

После сортировки для List<Figure>:
Прямоугольник с площадью 8
Квадрат с площадью 25
Круг с площадью 28,2743338823081
```

Матрица

```
Матрица:
[Прямоугольник с площадью 8      -      - ]
[ -      Квадрат с площадью 25    - ]
[ -      -      Круг с площадью 28,2743338823081]
```

Результат работы собственно-реализованной коллекции SimpleList и стека SimpleStack:

```
Перед сортировкой (SimpleList):  
Круг с площадью 28,2743338823081  
Прямоугольник с площадью 8  
Квадрат с площадью 25  
  
После сортировки (SimpleList):  
Прямоугольник с площадью 8  
Квадрат с площадью 25  
Круг с площадью 28,2743338823081  
  
Использование стека:  
Квадрат с площадью 25  
Прямоугольник с площадью 8  
Круг с площадью 28,2743338823081
```

Ссылка на репозиторий исходных кодов GitHub

https://github.com/VolandAID/Dub_Lab_3