

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Э. БАУМАНА

РАДИОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
курса «Базовые компоненты интернет-технологий»
«РАБОТА С КОЛЛЕКЦИЯМИ В C#»

Выполнил студент:

Елисеев Глеб Борисович
группа: РТ5-31

Проверил:

к.т.н., доцент
Гапанюк Юрий Евгеньевич

Москва, 2017 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 `Comparable`. 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 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` (разобранного в пособии). 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.

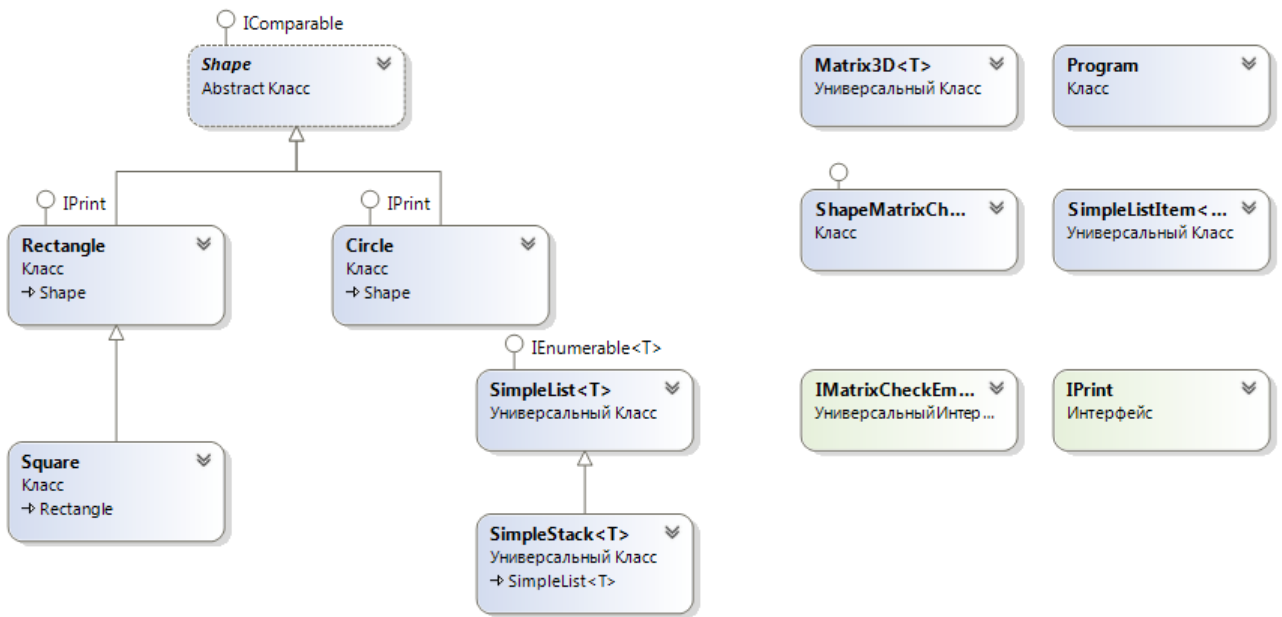


Рисунок 1 — Диаграмма классов.

Исходный код 1 — /source/Program.cs

```

using System;
using System.Collections;
using System.Collections.Generic;
using Shapes;
using SparseMatrix;
using ShapesCollections;

namespace lab3
{
    class Program
    {
        static void Main(string[] args)
        {
            #region Объекты классов Rectangle, Square и Circle
            Rectangle rect = new Rectangle(12, 22);
            Square square = new Square(24);
            Circle circle = new Circle(11);
            #endregion

            #region Коллекция класса ArrayList
            Console.WriteLine("Отсортированный необобщенный список");
        }
    }
}

```

```

ArrayList shapesArrList = new ArrayList();
shapesArrList.Add(rect);
shapesArrList.Add(square);
shapesArrList.Add(circle);

shapesArrList.Sort();

foreach (var shape in shapesArrList)
    Console.WriteLine(shape);
#endregion

#region Коллекция класса List<Shape>
Console.WriteLine("\nОтсортированный обобщенный список");

List<Shape> shapeList = new List<Shape>();
shapeList.Add(rect);
shapeList.Add(square);
shapeList.Add(circle);

shapeList.Sort();

foreach (var shape in shapeList)
    Console.WriteLine(shape);
#endregion

#region Пример работы разреженной матрицы Matrix3D<Shape>
Console.WriteLine("\nМатрица");

Matrix3D<Shape> matrix = new Matrix3D<Shape>(3, 3, 3, new
    ↪ ShapeMatrixCheckEmpty());
matrix[0, 0, 0] = rect;
matrix[1, 1, 1] = square;
matrix[2, 2, 2] = circle;

Console.WriteLine(matrix);
#endregion

#region Пример работы класса SimpleStack
Console.WriteLine("Стек");

SimpleStack<Shape> stack = new SimpleStack<Shape>();

```

```

        stack.Push(rect);
        stack.Push(square);
        stack.Push(circle);

        while (stack.Count > 0)
        {
            Shape shape = stack.Pop();
            Console.WriteLine(shape);
        }
        #endregion

        Console.ReadKey();
    }
}

```

Исходный код 2 — /source/Shapes/Shapes.cs

```

using System;

namespace Shapes
{
    /// <summary>
    /// Класс Фигура.
    /// </summary>
    abstract class Shape : IComparable
    {
        /// <summary>
        /// Тип фигуры.
        /// </summary>
        public string Type { get; protected set; }

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

        /// <summary>
        /// Сравнение элементов.
        /// </summary>

```

```

    /// <param name="obj">правый параметр сравнения</param>
    /// <returns> -1 - если левый параметр меньше правого
    ///             0 - параметры равны
    ///             1 - если левый параметр больше правого
    /// </returns>
    public int CompareTo(object obj)
    {
        Shape p = (Shape)obj;

        if (this.Area() < p.Area()) return -1;
        else if (this.Area() == p.Area()) return 0;
        else return 1;
    }

    /// <summary>
    /// Приведение к строке, переопределение метода Object.
    /// </summary>
    public override string ToString()
    {
        return "Площадь " + Type + "a = " + string.Format("{0:F2}",
            ↪ Area()) + " кв. ед.";
    }
}
}

```

Исходный код 3 — /source/Shapes/IPrint.cs

```

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

```

Исходный код 4 — /source/Shapes/Circle.cs

```
using System;

namespace Shapes
{
    /// <summary>
    /// Класс Круг.
    /// </summary>
    class Circle : Shape, IPrint
    {
        /// <summary>
        /// Радиус круга.
        /// </summary>
        private double radius;

        /// <summary>
        /// Основной конструктор.
        /// </summary>
        /// <param name="radius">Радиус круга</param>
        public Circle(double radius)
        {
            Type = "Круг";
            this.radius = radius;
        }

        /// <summary>
        /// Вычисление площади.
        /// </summary>
        public override double Area()
        {
            double area = Math.PI * Math.Pow(radius, 2);

            return area;
        }

        /// <summary>
        /// Вывод информации о фигуре в консоль.
        /// </summary>
        public void Print()
```

```

        {
            Console.WriteLine(ToString());
        }
    }
}

```

Исходный код 5 — /source/Shapes/Rectangle.cs

```

using System;

namespace Shapes
{
    /// <summary>
    /// Класс Прямоугольник.
    /// </summary>
    class Rectangle : Shape, IPrint
    {
        /// <summary>
        /// Высота.
        /// </summary>
        private double height;

        /// <summary>
        /// Ширина.
        /// </summary>
        private double width;

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

        /// <summary>
        /// Вычисление площади.
        /// </summary>
    }
}

```



```

public override double Area()
{
    double area = height * width;

    return area;
}

/// <summary>
/// Вывод информации о фигуре в консоль.
/// </summary>
public void Print()
{
    Console.WriteLine(ToString());
}
}
}

```

Исходный код 6 — /source/Shapes/Square.cs

```

namespace Shapes
{
    /// <summary>
    /// Класс Квадрат.
    /// </summary>
    class Square : Rectangle
    {
        /// <summary>
        /// Основной конструктор.
        /// </summary>
        /// <param name="size">Длина стороны квадрата</param>
        public Square(double size) : base(size, size)
        {
            Type = "Квадрат";
        }
    }
}

```

Исходный код 7 — /source/Shapes/ShapeMatrixCheckEmpty.cs

```

using SparseMatrix;

namespace Shapes
{
    class ShapeMatrixCheckEmpty : IMatrixCheckEmpty<Shape>
    {
        /// <summary>
        /// В качестве пустого элемента возвращается null.
        /// </summary>
        public Shape getEmptyElement() => null;

        /// <summary>
        /// Проверка что переданный параметр равен null.
        /// </summary>
        public bool checkEmptyElement(Shape element)
        {
            bool Result = false;
            if (element == null)
            {
                Result = true;
            }
            return Result;
        }
    }
}

```

Исходный код 8 — /source/SparseMatrix/IMatrixCheckEmpty.cs

```

namespace SparseMatrix
{
    /// <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);
}
}

```

Исходный код 9 — /source/SparseMatrix/SparseMatrix.cs

```

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

namespace SparseMatrix
{
    /// <summary>
    /// Класс Разреженная матрица.
    /// </summary>
    public class Matrix3D<T>
    {
        /// <summary>
        /// Словарь для хранения значений.
        /// </summary>
        Dictionary<string, T> _matrix = new Dictionary<string, T>();

        /// <summary>
        /// Максимальное число элементов по оси X.
        /// </summary>
        int maxX;

        /// <summary>
        /// Максимальное число элементов по оси Y.
        /// </summary>
        int maxY;

        /// <summary>
        /// Максимальное число элементов по оси Z.

```

```

/// </summary>
int maxZ;

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

/// <summary>
/// Основной конструктор.
/// </summary>
public Matrix3D(int x_size, int y_size, int z_size,
    ↪ IMatrixCheckEmpty<T> checkEmptyParam)
{
    this.maxX = x_size;
    this.maxY = y_size;
    this.maxZ = z_size;
    this.checkEmpty = checkEmptyParam;
}

/// <summary>
/// Индексатор для доступа к данным.
/// </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) => x.ToString() + "_" +
    ↪ y.ToString() + "_" + z.ToString();

/// <summary>
/// Приведение к строке.
/// </summary>
/// <returns></returns>
public override string ToString()
{
    StringBuilder b = new StringBuilder();

    for (int i = 0; i < this.maxX; i++)
    {
        b.Append("[");
    }

```

```

        for (int j = 0; j < this.maxY; j++)
        {
            b.Append("[");

            for (int z = 0; z < this.maxZ; z++)
            {
                if (!this.checkEmpty.checkEmptyElement(this[i, j, z]))
                    b.Append(this[i, j, z].ToString());
                else
                    b.Append(" - ");
            }

            if (j < this.maxY - 1)
                b.Append("], ");
            else
                b.Append("]");

        }

        b.Append("]\n");
    }

    return b.ToString();
}

}
}

```

Исходный код 10 — /source/SimpleListStack/SimpleListItem.cs

```

namespace ShapesCollections
{
    /// <summary>
    /// Элемент списка.
    /// </summary>
    public class SimpleListItem<T>
    {
        /// <summary>
        /// Данные.
        /// </summary>
        public T data { get; set; }
    }
}

```

```

    /// <summary>
    /// Следующий элемент.
    /// </summary>
    public SimpleListItem<T> next { get; set; }

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

```

Исходный код 11 — /source/SimpleListStack/SimpleList.cs

```

using System;
using System.Collections.Generic;

namespace ShapesCollections
{
    /// <summary>
    /// Класс Список.
    /// </summary>
    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>
    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;
        }
    }

    /// <summary>
    /// Чтение контейнера с заданным номером.
    /// </summary>
    public SimpleListItem<T> GetItem(int number)
    {
        if ((number < 0) || (number >= this.Count))
        {
            throw new Exception("Выход за границу индекса");
        }

        SimpleListItem<T> current = this.first;
        for (var i = 0; i < number; i++)
            current = current.next;

        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;

    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;
    }
}
}

```

Исходный код 12 — /source/SimpleListStack/SimpleStack.cs

```

using System;

namespace ShapesCollections
{
    /// <summary>
    /// Класс Стек.
    /// </summary>
    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()
        {
            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;
}
}
}

```

```

C:\Users\User\source\repos\lab3\lab3\bin\Debug\lab3.exe
Отсортированный необобщенный список
Площадь Прямоугольника = 264,00 кв. ед.
Площадь Круга = 380,13 кв. ед.
Площадь Квадрата = 576,00 кв. ед.

Отсортированный обобщенный список
Площадь Прямоугольника = 264,00 кв. ед.
Площадь Круга = 380,13 кв. ед.
Площадь Квадрата = 576,00 кв. ед.

Матрица
[[Площадь Прямоугольника = 264,00 кв. ед. - - ], [ - - - ], [ - - - ]]
[[ - - - ], [ - Площадь Квадрата = 576,00 кв. ед. - ], [ - - - ]]
[[ - - - ], [ - - - ], [ - - Площадь Круга = 380,13 кв. ед.]]

Стек
Площадь Круга = 380,13 кв. ед.
Площадь Квадрата = 576,00 кв. ед.
Площадь Прямоугольника = 264,00 кв. ед.
-

```

Рисунок 2 — Результат выполнения программы.