

Московский государственный технический
университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторная работа №3

Выполнила: Цветкова Алена

Группа: ИУ5-31

Дата:

Проверил: Гапанюк Ю.Е.

Дата:

Условие задачи

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 `Matrix` (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») 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` (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ»). 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.

Текст программы

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace лаба3
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle(5, 4);
            Square square = new Square(5);
            Round round = new Round(5);
            ArrayList al = new ArrayList();
            al.Add(333);
            al.Add(123.123);
            al.Add("строка");
            al.Add(rect);
            foreach (object o in al)
            {
                string type = o.GetType().Name;
                if (type == "Int32")
                {
                    Console.WriteLine("Целое число: " + o.ToString());
                }
                else if (type == "String")
                {
                    Console.WriteLine("Строка: " + o.ToString());
                }
                else
                {
                    Console.WriteLine("Другой тип: " + o.ToString());
                }
            }

            List<Figure> fl = new List<Figure>();
            fl.Add(rect);
            fl.Add(round);
            fl.Add(square);
            fl.Add(round);
            Console.WriteLine("\nПеред сортировкой:");
            foreach (var x in fl) Console.WriteLine(x);
            fl.Sort();
            Console.WriteLine("\nПосле сортировки:");
            foreach (var x in fl) Console.WriteLine(x);
            Matrix<int> x1 = new Matrix<int>(4, 3, 3, -1);
            x1[0, 0, 0] = 333;
        }
    }
}
```

```

        x1[1, 1, 1] = 334;
        try
        {
            x1[100, 100, 100] = 100;
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
        Console.WriteLine(x1);
        SimpleStack<Figure> stack = new SimpleStack<Figure>();
        //добавление данных в стек
        stack.Push(rect);
        stack.Push(square);
        stack.Push(round);
        //чтение данных из стека
        while (stack.Count > 0)
        {
            Figure f = stack.Pop();
            Console.WriteLine(f);
        }
        Console.ReadLine();
    }
}
abstract class Figure : IComparable
{
    public string Type { get; set; }
    public abstract double Area();
    public int CompareTo(object obj)
    {
        Figure p = (Figure)obj;
        if (this.Area() < p.Area()) return -1;
        else if (this.Area() == p.Area()) return 0;
        else return 1;
    }
}
class Rectangle : Figure, IPrint
{
    private double _property1 = 0;
    public double height
    {
        get
        {
            return _property1;
        }
        set
        {
            _property1 = value;
        }
    }
    private double _property2 = 0;

```

```

    public double width
    {
        get
        {
            return _property2;
        }
        set
        {
            _property2 = value;
        }
    }
    public Rectangle(double w, double h)
    {
        this.height = h; this.width = w; this.Type =
"Прямоугольник";
    }
    public override double Area()
    {
        return (this.height * this.width);
    }
    public override string ToString()
    {
        return this.Type + " со сторонами (" + this.width + "; " +
this.height + ") и площадью " + this.Area().ToString();
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}
class Square : Rectangle, IPrint
{
    public Square(double w) : base(w, w) { this.Type = "Квадрат"; }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}
class Round : Figure, IPrint
{
    private double _property1 = 0;
    public double radius
    {
        get
        {
            return _property1;
        }
        set
        {
            _property1 = value;
        }
    }
}

```

```

    }
    public Round(double r) { this.radius = r; this.Type =
"Окружность"; }
    public override double Area()
    {
        return (3.14159265 * this.radius * this.radius);
    }
    public override string ToString()
    {
        return this.Type + " с радиусом (" + this.radius + ") и
площадью " +
        this.Area().ToString();
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
}
interface IPrint
{
    void Print();
}
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;
    int maxZ;
    /// <summary>
    /// Пустой элемент, который возвращается если элемент с нужными
координатами не был задан
    /// </summary>
    T nullElement;
    /// <summary>
    /// Конструктор
    /// </summary>
    public Matrix(int px, int py, int pz, T nullElementParam)
    {
        this.maxX = px;
        this.maxY = py;

```

```

        this.maxZ = pz;
        this.nullElement = nullElementParam;
    }
    /// <summary>
    /// Индексатор для доступа к данным
    /// </summary>
    public T this[int x, int y, int z]
    {
        get
        {
            CheckBounds(x, y, z);
            string key = DictKey(x, y, z);
            if (this._matrix.ContainsKey(key))
            {
                return this._matrix[key];
            }
            else
            {
                return this.nullElement;
            }
        }
        set
        {
            CheckBounds(x, y, z);
            string key = DictKey(x, y, z);
            this._matrix.Add(key, value);
        }
    }
    /// <summary>
    /// Проверка границ
    /// </summary>
    void CheckBounds(int x, int y, int z)
    {
        if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + "
выходит за границы");
        if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + "
выходит за границы");
        if (z < 0 || z >= this.maxZ) throw new Exception("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()
    {
        //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("z = " + k + "\n");
            for (int j = 0; j < this.maxY; j++)
            {
                b.Append("[");
                for (int i = 0; i < this.maxX; i++)
                {
                    if (i > 0) b.Append("\t");
                    b.Append(this[i, j, k].ToString());
                }
                b.Append("]\n");
            }
            b.Append("\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;
    }
}
/// <summary>
/// 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>
/// <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;
        //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>
    /// <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 element) { Add(element); }
    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;
}
}
}

```

Проверка

```

Выбрать C:\Users\cveto\OneDrive\документы\visual studio 2017\Projects\лаба3\лаба3\bin\Debug\лаба3.exe
Целое число: 333
Другой тип: 123,123
Строка: строка
Другой тип: Прямоугольник со сторонами (5; 4) и площадью 20

Перед сортировкой:
Прямоугольник со сторонами (5; 4) и площадью 20
Окружность с радиусом (5) и площадью 78,53981625
Квадрат со сторонами (5; 5) и площадью 25
Окружность с радиусом (5) и площадью 78,53981625

После сортировки:
Прямоугольник со сторонами (5; 4) и площадью 20
Квадрат со сторонами (5; 5) и площадью 25
Окружность с радиусом (5) и площадью 78,53981625
Окружность с радиусом (5) и площадью 78,53981625
x=100 выходит за границы
z = 0
[333  -1    -1    -1]
[-1   -1    -1    -1]
[-1   -1    -1    -1]

z = 1
[-1   -1    -1    -1]
[-1   334  -1    -1]
[-1   -1    -1    -1]

z = 2
[-1   -1    -1    -1]
[-1   -1    -1    -1]
[-1   -1   336  -1]

Окружность с радиусом (5) и площадью 78,53981625
Квадрат со сторонами (5; 5) и площадью 25
Прямоугольник со сторонами (5; 4) и площадью 20

```