

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

Факультет «Информатика и управление»

Кафедра ИУ5.

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

Выполнил:
студент группы ИУ5-31Б

Гришин Станислав

Подпись и дата:
21.12.2020

Проверил:
преподаватель каф.
ИУ5

Гапанюк Ю.Е.

Подпись и дата:
21.12.2020

г. Москва, 2020 г.

Постановка 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 (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.

Разработка интерфейса класса

Program

Класс

Методы

Main

Вложенные типы

IPrint

Circle

Класс

→ Figure

IPrintable

Figure

Abstract Класс

IPrintable

IMatrixCheckEmpty<Figure>

FigureMatrixCheckEmpty

Класс

IMatrixCheckEmpty<T>

Универсальный Интерфейс

IPrint

Интерфейс

Matrix<T>

Универсальный Класс

IPrint

Quadrat

Класс

→ Rectangle

IPrint

Rectangle

Класс

→ Figure

IPrintable

IEnumerable<T>

SimpleList<T>

Универсальный Класс

SimpleListItem<T>

Универсальный Класс

SimpleStack<T>

Универсальный Класс

→ SimpleList<T>

Листинг программы

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

namespace Laba3
{
    class Program
    {
        abstract class Figure: IComparable
        {
            abstract public 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 Circle : Figure, IPrint
        {
            private int pradius { get; set; }
            public Circle (int radius)
            {
                this.pradius = radius;
            }
        }
    }
}
```

```

public override double Area()
{
    double S;

    S = Math.PI*pradius*pradius;

    return S;
}

public void Print()
{
    Console.WriteLine(this.ToString());
}

public override string ToString()
{
    return "Радиус круга: " + this.pradius + " " + "Площадь круга: " + this.Area();
}
}

class Quadrate : Rectangle, IPrint
{
    private int pwidth { get; set; }

    public Quadrate (int width):base(width,width)
    {
        this.pwidth = width;
    }

    public override double Area()
    {
        double S;

        S = pwidth* pwidth;

        return S;
    }

    public override string ToString()
    {
        return "Сторона квадрата: " + this.pwidth + " " + "Площадь квадрата: " + this.Area();
    }
}

```

```

    }
}

class Rectangle : Figure, IPrint
{
    private int pwidth { get; set; }
    private int pheight { get; set; }
    public Rectangle(int width,int height)
    {
        this.pwidth = width;
        this.pheight = height;
    }
    public override double Area()
    {
        double S;
        S = pwidth * pheight;
        return S;
    }
    public void Print()
    {
        Console.WriteLine(this.ToString());
    }
    public override string ToString()
    {
        return "Ширина и высота прямоугольника: " + this.pwidth + " " + this.pheight + " " + "Площадь
прямоугольника: " + this.Area();
    }
}

interface IPrint
{
    void Print();
}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

```

```

{
    public Figure getEmptyElement()
    {
        return null;
    }

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

public class Matrix<T>
{
    Dictionary<string, T> _matrix = new Dictionary<string, T>();
    int maxX;
    int maxY;
    int maxZ;
    IMatrixCheckEmpty<T> checkEmpty;
    public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam)
    {
        this.maxX = px;
        this.maxY = py;
        this.maxZ = pz;
        this.checkEmpty = checkEmptyParam;
    }

    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();
}
}

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");
}

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

public override string ToString()
{

```



```

        StringBuilder b = new StringBuilder();

        for (int k = 0; k < this.maxZ; k++)
        {
            b.Append("{\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");

                    if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))
                    {
                        b.Append(this[i, j, k].ToString());
                    }

                    else
                    {
                        b.Append(" - ");
                    }
                }

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

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

        return b.ToString();
    }

}

public interface IMatrixCheckEmpty<T>
{
    T getEmptyElement();
}

```

```

    bool checkEmptyElement(T element);
}

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>
    where T : IComparable
{
    protected SimpleListItem<T> first = null;

    protected SimpleListItem<T> last = null;

    public int Count
    {
        get { return _count; }

        protected set { _count = value; }
    }

    int _count;

    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;
    }
}

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

    SimpleListItem<T> current = this.first;

    int i = 0;

    while (i < number)
    {
        current = current.next;

        i++;
    }

    return current;
}

public T Get(int number)
{
    return GetItem(number).data;
}

public IEnumerator<T> GetEnumerator()
{
    SimpleListItem<T> current = this.first;

    while (current != null)
    {
        yield return current.data;

        current = current.next;
    }
}

```

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

```
{
    return GetEnumerator();
}

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

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);
}
```

```

    }

    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
{
    public void Push(T element)
    {
        Add(element);
    }

    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;
}
}

```

```

static void Main(string[] args)
{
    int a, b;

    Console.WriteLine("Введите ширину прямоугольника:");
    a = Convert.ToInt32(Console.ReadLine());

    Console.WriteLine("Введите высоту прямоугольника:");
    b = Convert.ToInt32(Console.ReadLine());

    Rectangle A = new Rectangle(a, b);

    Console.WriteLine("Введите сторону квадрата:");
    a = Convert.ToInt32(Console.ReadLine());

    Quadrate B = new Quadrate(a);

    Console.WriteLine("Введите радиус круга:");
    a = Convert.ToInt32(Console.ReadLine());

    Circle C = new Circle(a);

    ArrayList L = new ArrayList();

    L.Add(A);

    L.Add(B);

    L.Add(C);

    Console.WriteLine("\nНеобобщенный список:");

    Console.WriteLine("\nПеред сортировкой:");

    foreach (var x in L)
    {
        Console.WriteLine(x);
    }

    L.Sort();
}

```

```

Console.WriteLine("\nПосле сортировки:");

foreach (var x in L)
{
    Console.WriteLine(x);
}

List<Figure> M = new List<Figure>()
{
    A,
    B,
    C
};

Console.WriteLine("\nОбобщенный список:");

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in M)
{
    Console.WriteLine(x);
}

M.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in L)
{
    Console.WriteLine(x);
}

Console.WriteLine("\nМатрица:");

Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 3, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = A;

matrix[1, 1, 1] = B;

matrix[2, 2, 2] = C;

Console.WriteLine(matrix.ToString());

SimpleList<Figure> list = new SimpleList<Figure>();

list.Add(A);

```

```

list.Add(B);

list.Add(C);

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in list)

    Console.WriteLine(x);

list.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in list)

    Console.WriteLine(x);

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

stack.Push(A);

stack.Push(B);

stack.Push(C);

while (stack.Count > 0)

{

    Figure f = stack.Pop();

    Console.WriteLine(f);

}

}

}

}

```

Анализ результатов


```

Введите ширину прямоугольника:
5
Введите высоту прямоугольника:
4
Введите сторону квадрата:
6
Введите радиус круга:
3

Необобщенный список:

Перед сортировкой:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Сторона квадрата: 6 Площадь квадрата: 36
Радиус круга: 3 Площадь круга: 28,274333882308138

После сортировки:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Радиус круга: 3 Площадь круга: 28,274333882308138
Сторона квадрата: 6 Площадь квадрата: 36

Обобщенный список:

Перед сортировкой:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Сторона квадрата: 6 Площадь квадрата: 36
Радиус круга: 3 Площадь круга: 28,274333882308138

После сортировки:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Радиус круга: 3 Площадь круга: 28,274333882308138
Сторона квадрата: 6 Площадь квадрата: 36

```

```

Матрица:
{
[Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20 - - ]
[ - - - ]
[ - - - ]
}

{
[ - - - ]
[ - Сторона квадрата: 6 Площадь квадрата: 36 - ]
[ - - - ]
}

{
[ - - - ]
[ - - - ]
[ - - Радиус круга: 3 Площадь круга: 28,274333882308138]
}

Перед сортировкой:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Сторона квадрата: 6 Площадь квадрата: 36
Радиус круга: 3 Площадь круга: 28,274333882308138

После сортировки:
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20
Радиус круга: 3 Площадь круга: 28,274333882308138
Сторона квадрата: 6 Площадь квадрата: 36
Радиус круга: 3 Площадь круга: 28,274333882308138
Сторона квадрата: 6 Площадь квадрата: 36
Ширина и высота прямоугольника: 5 4 Площадь прямоугольника: 20

```