**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»

отчёт

по лабораторной работе №3.

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студентка группы ИУ5-32 |  | преподаватель каф. ИУ5 |
| Честнова Е. А. |  | Гапанюк Ю. Е. |
|  |  |  |

Москва, 2020 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 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.

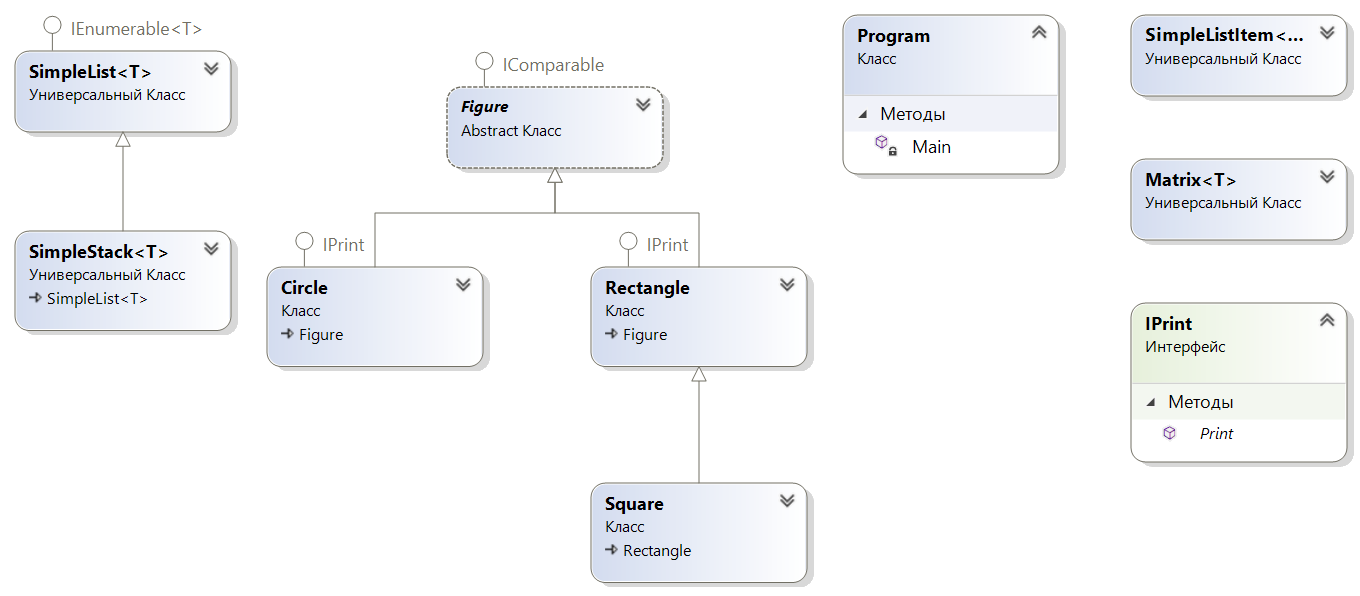
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:

· 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.

Диаграмма классов



Текст программы

using FigureCollections;

using System;

using System.Collections;

using System.Collections.Generic;

namespace lab2Csh

{

class FigureCheck : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

if (element == null) return true;

return false;

}

}

abstract class Figure: IComparable

{

public string Type { get; set; }

public abstract double Area();

public override string ToString()

{

return this.Type + " площадью " + this.Area().ToString();

}

public int CompareTo(object fig)

{

Figure a = (Figure)fig;

if (this.Area() < a.Area()) return -1;

else if (this.Area() == a.Area()) return 0;

else return 1;

}

}

interface IPrint

{

void Print();

}

class Rectangle : Figure, IPrint {

protected int \_width;

protected int \_height;

public int width {

get { return \_width; }

set {

if (value > 0) { \_width = value; }

else { \_width = 0; }

}

}

public int height {

get { return \_height; }

set

{

if (value > 0) { \_height = value; }

else { \_height = 0; }

}

}

public Rectangle(int x, int y) {

height = x;

width = y;

this.Type = "Прямоугольник " ;

}

public override double Area()

{

return width \* height;

}

public void Print() {

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle {

public Square(int a) : base (a, a) {

this.Type = "Квадрат ";

}

}

class Circle : Figure, IPrint {

private int \_radius;

public int radius {

get { return \_radius; }

set

{

if (value > 0) { \_radius = value; }

else { \_radius = 0; }

}

}

public Circle(int a) {

radius = a;

Type = "Круг ";

}

public override double Area() {

return Math.Round(radius \* radius \* Math.PI, 3);

}

public void Print() {

Console.WriteLine(this.ToString());

}

}

class Program

{

static void Main(string[] args) {

Rectangle first = new Rectangle(3, 2);

Square second = new Square(4);

Circle third = new Circle(2);

ArrayList arr1 = new ArrayList();

arr1.Add(first);

arr1.Add(second);

arr1.Add(third);

Console.WriteLine("Необобщенная коллекция");

foreach (object obj in arr1) Console.WriteLine(obj.ToString());

arr1.Sort();

foreach (object obj in arr1) Console.WriteLine(obj.ToString());

List <Figure> arr2 = new List<Figure>();

arr2.Add(first);

arr2.Add(second);

arr2.Add(third);

Console.WriteLine("Обобщенная коллекция");

foreach (Figure fig in arr2) Console.WriteLine(fig.ToString());

arr2.Sort();

foreach (Figure fig in arr2) Console.WriteLine(fig.ToString());

Console.WriteLine("\nРабота с разреженной матрицей на 3 измерения");

Matrix<Figure> figs = new Matrix<Figure>(3, 3, 3, new FigureCheck());

figs[0, 0, 0] = first;

figs[1, 1, 1] = second;

figs[2, 2, 2] = third;

Console.WriteLine(figs.ToString());

Console.WriteLine("\n\nРабота со стеком");

SimpleStack<Figure> figs2 = new SimpleStack<Figure>();

figs2.Push(first);

figs2.Push(second);

figs2.Push(third);

foreach (var o in figs2) Console.WriteLine(figs2.Pop());

figs2.Push(first);

figs2.Push(second);

figs2.Push(third);

figs2.Sort();

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

foreach (var o in figs2) Console.WriteLine(figs2.Pop());

}

}

public class Matrix<T>

{

Dictionary<string, T> \_matrix = new Dictionary<string, T>();

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> сheckEmpty;

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

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.сheckEmpty.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++)

{

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.сheckEmpty.checkEmptyElement(this[i, j,k]))

{

b.Append(this[i, j,k].ToString());

}

else

{

b.Append(" - ");

}

}

b.Append("] ");

}

b.Append("\n");

}

return b.ToString();

}

}

public interface IMatrixCheckEmpty<T>

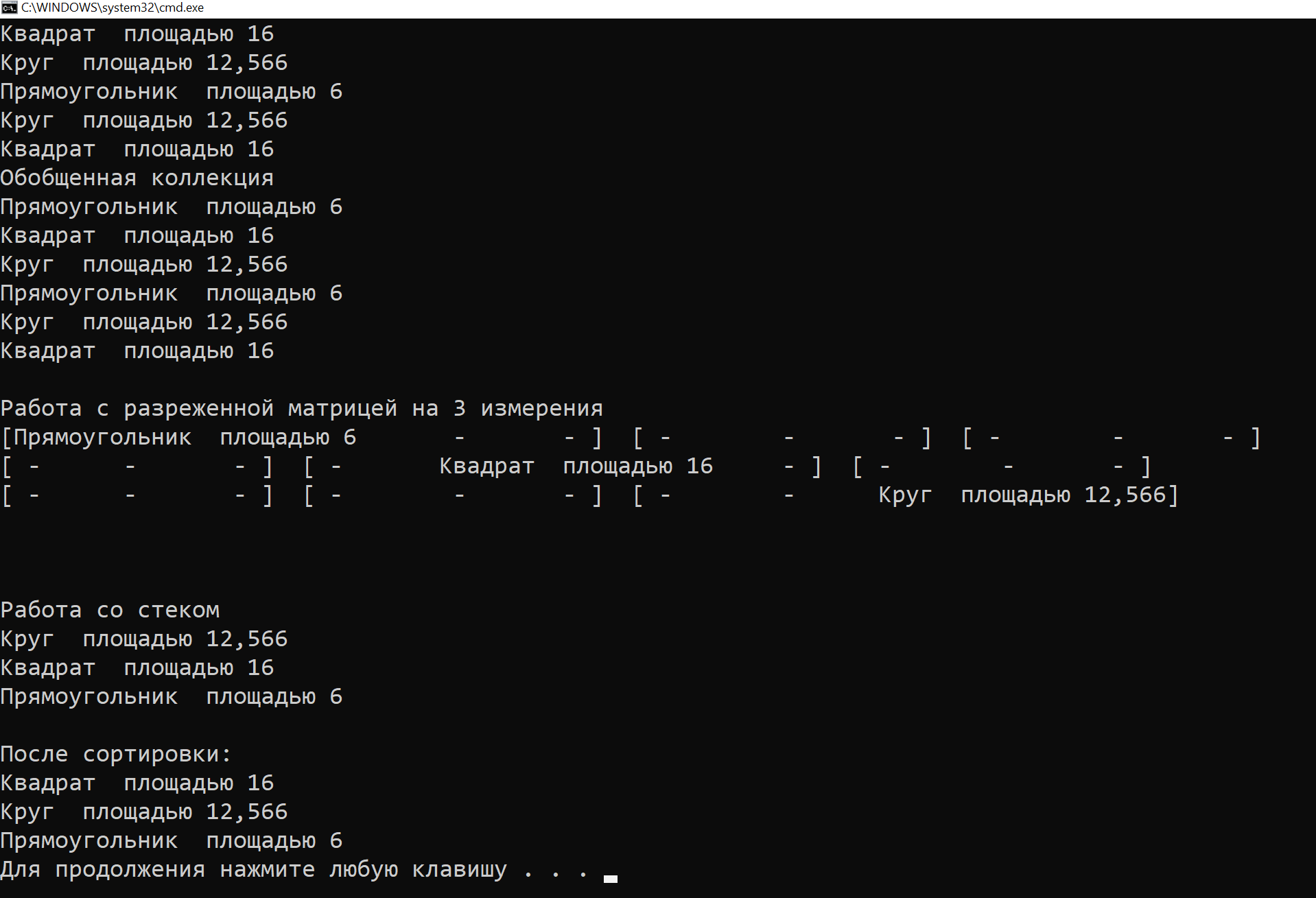
{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

Анализ результатов