МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

ИМ. Н. Э. БАУМАНА

Факультет "Информатика и системы управления"

### Кафедра ИУ5 “Системы обработки информации и управления”

Группа ИУ5-32Б

Отчёт по лабораторной работе №3

«Работа с коллекциями»

Студент: Преподаватель:

Ваганов Даниил Дмитриевич Гапанюк Юрий Евгеньевич

Дата: \_\_\_.\_\_\_.\_\_\_\_\_\_ Дата: \_\_\_.\_\_\_.\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_ Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020

**Условие 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.

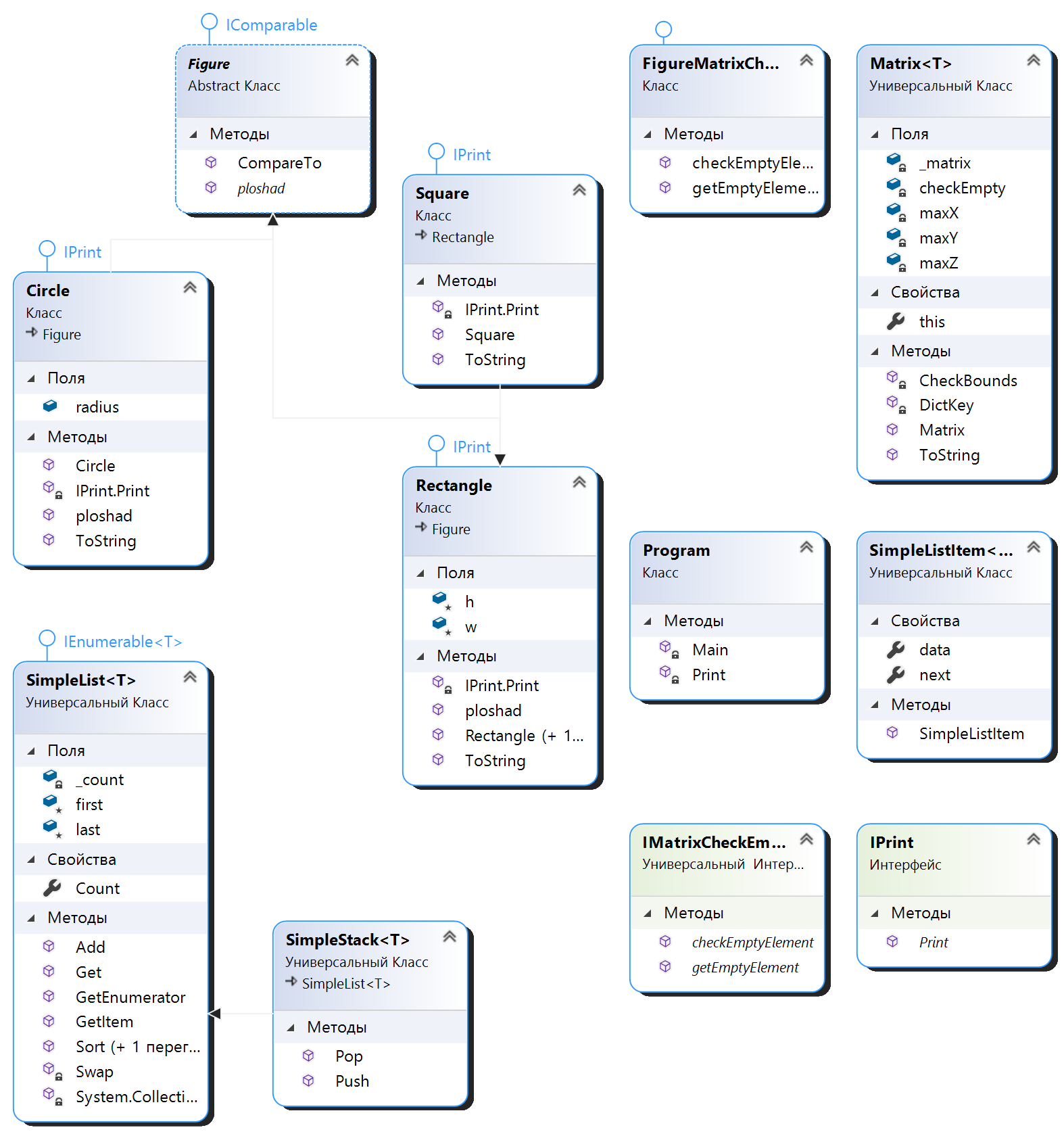
7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:

• 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 System;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

public 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;

}

}

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 this.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;

}

}

public class SimpleListItem<T>

{

public T data { get; set; }

public SimpleListItem<T> next { get; set; }

public SimpleListItem(T param)

{

this.data = param;

}

}

public class Matrix<T>

{

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

int maxX, maxY, maxZ;

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.checkEmpty = checkEmptyParam;

}

IMatrixCheckEmpty<T> checkEmpty;

public T this[int x, int y, int z] //индексатор

{

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("z = " + (k + 1) + "\n");

for (int j = 0; j < this.maxY; j++)

{

b.Append("[");

for (int i = 0; i < this.maxX; i++)

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

if (!this.checkEmpty.checkEmptyElement(this[i, j, k]))

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

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

}

return b.ToString();

}

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

bool result = false;

if (element == null)

{

result = true;

}

return result;

}

}

abstract class Figure : IComparable

{

public abstract double ploshad();

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.ploshad() < p.ploshad()) { return -1; }

else if (this.ploshad() == p.ploshad()) { return 0; }

else return 1;

}

}

class Rectangle : Figure, IPrint

{

protected double w, h;

public Rectangle()

{

w = h = 0;

}

public Rectangle(double a, double b)

{

w = a;

h = b;

}

public override double ploshad()

{

return (w \* h);

}

public override string ToString()

{

return $"Rectangle {ploshad()}";

}

void IPrint.Print()

{

Console.WriteLine(ToString());

}

}

class Square : Rectangle, IPrint

{

public Square(double a) : base(a, a) { }

public override string ToString()

{

return $"Square {ploshad()}";

}

void IPrint.Print()

{

Console.WriteLine(ToString());

}

}

class Circle : Figure, IPrint

{

public double radius;

public Circle(double r)

{

radius = r;

}

public override double ploshad()

{

return (radius \* radius \* Math.PI);

}

public override string ToString()

{

return $"Circle {ploshad()}";

}

void IPrint.Print()

{

Console.WriteLine(ToString());

}

}

interface IPrint

{

void Print();

}

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class Program

{

static void Print(IPrint pr)

{

pr.Print();

}

static void Main(string[] args)

{

Rectangle rect = new Rectangle(5, 4);

Circle circle = new Circle(5);

Square square = new Square(5);

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

fl.Add(rect);

fl.Add(circle);

fl.Add(square);

fl.Sort();

Console.WriteLine("Вывод List<Figure>");

foreach (Figure i in fl)

{

Console.WriteLine(i.ToString());

}

Matrix <Figure> matrix = new Matrix<Figure>(3, 3, 2, new FigureMatrixCheckEmpty());

matrix[0, 0, 1] = rect;

matrix[1, 1, 0] = square;

matrix[2, 2, 0] = circle;

Console.WriteLine("\nВывод Matrix<Figure>");

Console.WriteLine(matrix.ToString());

SimpleStack<Figure> \_stack = new SimpleStack<Figure>();

\_stack.Add(rect);

\_stack.Add(circle);

\_stack.Add(square);

Console.WriteLine("Вывод SimpleStack<Figure> до метода Pop()");

foreach (Figure i in \_stack)

{

Console.WriteLine(i.ToString());

}

\_stack.Pop();

Console.WriteLine("\nВывод SimpleStack<Figure> после метода Pop()");

foreach (Figure i in \_stack)

{

Console.WriteLine(i.ToString());

}

}

}

}

**Анализ результатов:**

|  |  |
| --- | --- |
| *Тест* | *Вывод программы* |
| 1 |  |
| 2 |  |