МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н.Э.БАУМАНА

Факультет: «Информатика и системы управления»

Кафедра: «Системы обработки информации и управления (ИУ5)»

Отчет по лабораторной работе №3

По дисциплине «Базовые компоненты интернет-технологий»

Выполнил: Тураев Глеб Вадимович

Группа: ИУ5Ц-51Б

Преподаватель: Гапанюк Юрий Евгеньевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 25.12.2018

Подпись:

**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. 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 (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.

**Текст программы:**

*Program.cs*

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using FigureCollections;

using LR3;

namespace LR2

{

class Program

{

static void Main(string[] args)

{

//создание экзмепляров классов

var rect = new Rectangle(4, 5);

var square = new Square(6);

var circle = new Circle(10);

//создание обобщенного списка и вывод

var arrayList = new ArrayList { square, circle, rect };

Console.WriteLine("Через необобщенную коллекцию ArrayList ");

Console.WriteLine("До сортировки:");

foreach (var figure in arrayList)

{

Console.WriteLine(((Figure)figure).area());

}

// сортировка и вывод

arrayList.Sort();

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

foreach (var figure in arrayList)

{

Console.WriteLine(((Figure)figure).area());

}

Console.WriteLine("\nТо же самое, но только через обобщенную коллекцию List<T> ");

//создание необобщенного списка, сортировка и вывод

var list = new List<Figure> { square, circle, rect };

list.Sort();

foreach (var figure in list)

{

Console.WriteLine(figure.area());

}

//создание разреженной матрицы

Console.WriteLine("\nМатрица");

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

for (var i = 0; i < 2; i++)

{

for (var j = 0; j < 2; j++)

{

for (var k = 0; k < 2; k++)

{

matrix[i, j, k] = list[new Random().Next(0, 2)];

}

}

}

Console.WriteLine(matrix.ToString());

Console.WriteLine("Работаем со стеком: ");

var stack = new SimpleStack<Figure>();

stack.push(rect);

stack.push(square);

stack.push(circle);

stack.add(circle);

Console.WriteLine("\nВывод стэка простым перебором обычного списка по принципу FIFO: ");

foreach (var element in stack)

{

Console.WriteLine(element.ToString());

}

Console.WriteLine("\nВывод стэка методом pop() по принципу LIFO: ");

while (stack.count > 0)

{

Console.WriteLine(stack.pop().ToString());

}

}

}

}

*Circle.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2

{

class Circle : Figure

{

public double radius { get; set; }

public Circle(double radius)

{

this.radius = radius;

}

public override double area()

{

return 2 \* Math.PI \* radius \* radius;

}

public override string ToString()

{

return "Радиус круга = " + radius + ";Площадь = " + area();

}

}

}

*Figure.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2

{

abstract class Figure : IPrint, IComparable

{

public abstract double area();

public void Print()

{

Console.WriteLine(ToString());

}

public int CompareTo(object obj)

{

var right = (Figure)obj;

if (area() < right.area()) return -1;

if (area() == right.area()) return 0;

return 1;

}

}

}

*FigureMatrixCheckEmpty.cs*

using LR2;

namespace LR3

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement() => null;

public bool checkEmptyElement(Figure element) => element == null;

//Figure IMatrixCheckEmpty<Figure>.getEmptyElement()

//{

// throw new System.NotImplementedException();

//}

}

}

*IMatrixCheckEmpty.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR3

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

*IPrint.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2

{

public interface IPrint

{

void Print();

}

}

*Rectangle.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2

{

class Rectangle : Figure

{

public Rectangle(double length, double width)

{

this.length = length;

this.width = width;

}

public double length { get; set; }

public double width { get; set; }

public override double area()

{

return length \* width;

}

public override string ToString()

{

return "Длина прямоугольника = " + length + "; Ширина прямоугольника = " + width + "; Площадь прямоугольника= " + area();

}

}

}

*SimpleList.cs*

using System;

using System.Collections;

using System.Collections.Generic;

namespace LR3

{

public class SimpleList<T> : IEnumerable<T> where T : IComparable

{

protected SimpleListItem<T> first;

protected SimpleListItem<T> last;

public int count { get; protected set; }

public void add(T element)

{

var newItem = new SimpleListItem<T>(element);

count++;

if (last == null)

{

first = newItem;

last = newItem;

}

else

{

last.next = newItem;

last = newItem;

}

}

public SimpleListItem<T> getItem(int number)

{

if (number < 0 || number >= count)

{

throw new ArgumentOutOfRangeException("Выход за границу индекса");

}

var current = first;

var i = 0;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

public T Get(int number)

{

return getItem(number).data;

}

public IEnumerator<T> GetEnumerator()

{

var current = first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void sort()

{

sort(0, 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;

}

}

}

*SimpleListItem.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR3

{

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T data)

{

this.data = data;

}

}

}

*SimpleStack.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR3

{

class SimpleStack<T> : SimpleList<T> where T : IComparable

{

public void push(T element)

{

add(element);

}

public T pop()

{

var result = default(T);

if (count == 0) return result;

if (count == 1)

{

result = first.data;

first = null;

last = null;

}

else

{

var newLast = getItem(count - 2);

result = newLast.next.data;

last = newLast;

newLast.next = null;

}

count--;

return result;

}

}

}

*SparceMatrix.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using LR3;

namespace FigureCollections

{

public class Matrix<T>

{

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

private int \_maxX;

private int \_maxY;

private int \_maxZ;

IMatrixCheckEmpty<T> \_сheckEmpty;

public Matrix(int maxX, int maxY, int maxZ,

IMatrixCheckEmpty<T> сheckEmptyParam)

{

\_maxX = maxX;

\_maxY = maxY;

\_maxZ = maxZ;

\_сheckEmpty = сheckEmptyParam;

}

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

{

set

{

checkBounds(x, y, z);

string key = getDictKey(x, y, z);

\_matrix.Add(key, value);

}

get

{

checkBounds(x, y, z);

var key = getDictKey(x, y, z);

if (\_matrix.ContainsKey(key))

{

return \_matrix[key];

}

return \_сheckEmpty.getEmptyElement();

}

}

void checkBounds(int x, int y, int z)

{

if (x < 0 || x >= \_maxX)

{

throw new ArgumentOutOfRangeException("x", "x=" + x + " выходит за границы");

}

if (y < 0 || y >= \_maxY)

{

throw new ArgumentOutOfRangeException("y", "y=" + y + " выходит за границы");

}

if (z < 0 || z >= \_maxZ)

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");

}

}

string getDictKey(int x, int y, int z)

{

return x + "\_" + y + "\_" + z;

}

public override string ToString()

{

var b = new StringBuilder();

for (int k = 0; k < \_maxZ; k++)

{

b.Append("K = " + k + "\n");

for (int j = 0; j < \_maxY; j++)

{

for (int i = 0; i < \_maxX; i++)

{

if (i > 0)

{

b.Append("\t");

}

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

{

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

}

else

{

b.Append(" - ");

}

}

b.Append("\n");

}

b.Append("\n");

}

return b.ToString();

}

}

}

*Square.cs*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LR2

{

class Square : Rectangle

{

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

public override string ToString()

{

return "Сторона квадрата = " + length + "; Площадь = " + area();

}

}

}

**Результат работы:**

