Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

****

Факультет ИУ

Кафедра ИУ5

**Отчет по лабораторной работе № 3**

Студент Гаджиев К.К.

Группа ИУ5-32

Название предмета БКИТ

Руководитель

Гапанюк Ю.Е. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата подпись

Студент

Гаджиев К.К.

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

дата подпись

Задание

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 (разобранного в пособии). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * public T Pop() – чтение с удалением из стека.

Листинг программы

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

//task #2-3

Rectangle rect = new Rectangle(5, 10);

Square square = new Square(5);

Circle circle = new Circle(5);

//task #4

Console.WriteLine("ArrayList");

ArrayList Alist = new ArrayList();

Alist.Add(rect);

Alist.Add(square);

Alist.Add(circle);

foreach (var x in Alist) Console.WriteLine(x);

Console.WriteLine("ArrayList sorted");

Alist.Sort();

foreach (var x in Alist) Console.WriteLine(x);

//task #5

Console.WriteLine("\nList");

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

list.Add(rect);

list.Add(square);

list.Add(circle);

foreach (var x in list) Console.WriteLine(x);

Console.WriteLine("List sorted");

list.Sort();

foreach (var x in list) Console.WriteLine(x);

//task #6

Console.WriteLine("\nМатрица");

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

matrix[0, 0, 0] = rect;

matrix[1, 1, 0] = square;

matrix[2, 2, 1] = circle;

Console.WriteLine(matrix.ToString());

try

{

Figure temp = matrix[123, 123, 12];

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

//task #7

Console.WriteLine("\nСтэк");

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

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;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

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

{

public void Push(T elem)

{

Add(elem);

}

public T Pop()

{

T res = default(T); //!!

if (this.Count == 0) return res;

if (this.Count == 1)

{

res = this.first.data;

this.first = null;

this.last = null;

}

else

{

SimpleListItem<T> newLast = this.GetItem(this.Count - 2);

res = newLast.next.data;

this.last = newLast;

newLast.next = null;

}

this.Count--;

return res;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

bool here = false;

if (element == null)

{

here = true;

}

return here;

}

}

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++)

{

string temp = "z = " + k.ToString() + '\n';

b.Append(temp);

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("]\n");

}

}

return b.ToString();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

interface IPrint

{

void Print();

}

abstract class Figure : IComparable

{

string Figure\_Type;

public string Type

{

get

{

return this.Figure\_Type;

}

protected set

{

this.Figure\_Type = value;

}

}

public abstract double calc\_s();

public override string ToString()

{

return "The area of " + this.Type + " is " + this.calc\_s().ToString();

}

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.calc\_s() < p.calc\_s()) return -1;

else if (this.calc\_s() == p.calc\_s()) return 0;

else return 1;

}

}

class Rectangle : Figure, IPrint

{

private double width;

private double height;

public Rectangle(double w, double h)

{

this.width = w;

this.height = h;

this.Type = "rectangle";

}

public override double calc\_s()

{

return this.width \* this.height;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle

{

private double length;

public Square(double l) : base(l, l)

{

this.Type = "square";

}

}

class Circle : Figure, IPrint

{

private double radius;

public Circle(double r)

{

this.radius = r;

this.Type = "circle";

}

public override double calc\_s()

{

return Math.Round(Math.PI \* radius \* radius, 2);

}

public void Print()

{

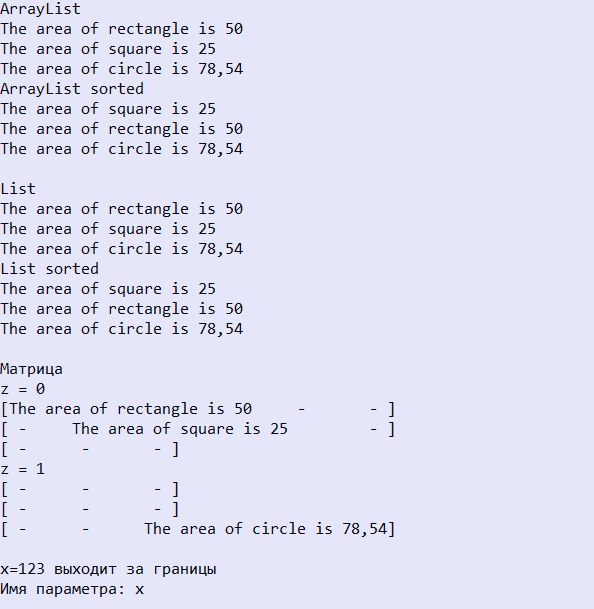
Console.WriteLine(this.ToString());

}

}

}

Результаты работы программы



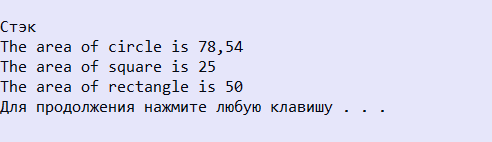


Диаграмма классов

