|  |  |  |
| --- | --- | --- |
| Защищено:  Гапанюк Ю.Е.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |  | Демонстрация:  Гапанюк Ю.Е.  "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

**Отчет по лабораторной работе № 2 по курсу** «**базовые компоненты интернет технологий»**

|  |  |
| --- | --- |
| ИСПОЛНИТЕЛЬ: |  |
| студент группы:  ИУ5Ц-51б | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | (подпись) |
| Авлияров М.В. | "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

Москва, МГТУ - 2017

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

**Лабораторная работа №3**

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.

**Листинг программы**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace lab3

{

interface IComparable<Figure>

{

int CompareTo(Figure obj);

}

interface IPrint

{

void Print();

}

abstract class Figure : IPrint, IComparable<Figure>, IComparable

{

public string \_Type;

public string Type

{

get

{ return this.\_Type; }

protected set

{

this.\_Type = value;

}

}

public abstract double area();

public override string ToString()

{

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

}

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

public int CompareTo(Figure obj)

{

Figure p = (Figure)obj;

if (this.area() < p.area())

return -1;

else if (this.area() == p.area())

return 0;

else

return 1;

}

public int CompareTo(object other)

{

return CompareTo(other as Figure);

}

}

class Rectangle : Figure

{

public double height;

public double width;

public Rectangle(double height, double width)

{

this.height = height;

this.width = width;

this.Type = "Прямоугольник";

}

public override double area() { return height \* width; }

}

class Square : Rectangle

{

public Square(double size) : base(size, size) { this.Type = "Квадрат"; }

}

class Circle : Figure

{

public double radius;

public Circle(double radius) { this.radius = radius; this.Type = "Круг"; }

public override double area() { return Math.PI \* radius \* radius; }

}

public class Matrix3D<T>

{

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

int maxX;

int maxY;

int maxZ;

T nullElement;

public Matrix3D(int px, int py, int pz, T nullElementParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.nullElement = nullElementParam;

}

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

{

get

{

CheckBounds(x, y, z);

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

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

set

{

CheckBounds(x, y, z);

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

this.\_matrix.Add(key, value);

}

}

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

{

if (x < 0 || x >= this.maxX)

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

if (y < 0 || y >= this.maxY)

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

if (z < 0 || z >= this.maxZ)

throw new Exception("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 i = 0; i < this.maxX; i++)

{

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

{

b.Append("[");

for (int k = 0; k < this.maxZ; k++)

{

T element = this[i, j, k];

b.Append(String.Format("{0,25}", element == null ? "0" : element.ToString()));

}

b.Append("]\n");

}

b.Append("\n");

}

return b.ToString();

}

}

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 MainClass

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(5, 4);

Square square = new Square(5);

Circle circle = new Circle(5);

Console.WriteLine("\nArrayList ");

ArrayList al = new ArrayList();

al.Add(circle);

al.Add(rect);

al.Add(square);

foreach (var x in al)

Console.WriteLine(x);

Console.WriteLine("\nArrayList - сортировка ");

al.Sort();

foreach (var x in al)

Console.WriteLine(x);

Console.WriteLine("\nList < Figure > ");

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

fl.Add(circle);

fl.Add(rect);

fl.Add(square);

foreach (var x in fl)

Console.WriteLine(x);

Console.WriteLine("\nList < Figure > - сортировка ");

fl.Sort();

foreach (var x in fl)

Console.WriteLine(x);

Console.WriteLine("\nМатрица ");

Matrix3D<Figure> cube = new Matrix3D<Figure>(3, 3, 3, null);

cube[0, 0, 0] = rect;

cube[1, 1, 1] = square;

cube[2, 2, 2] = circle;

Console.WriteLine(cube.ToString());

Console.WriteLine("\nСписок ");

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

list.Add(square);

list.Add(rect);

list.Add(circle);

foreach (var x in list)

Console.WriteLine(x);

list.Sort();

Console.WriteLine("\nСортировка списка ");

foreach (var x in list)

Console.WriteLine(x);

/\*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);

}

Console.ReadLine(); \*/

}

}

}

**Результаты работы программы**

