МГТУ им. Баумана  
Факультет «Информатика и системы управления»  
Кафедра «Автоматизированные системы обработки информации и управления»  
Дисциплина «Базовые компоненты интернет-технологий»

Отчёт по лабораторной работе №3

Выполнил:  
Студент группы ИУ5-33Б  
Андреев А.А.   
Преподаватель:  
Гапанюк Ю.

1. **Описание задания**
   1. Разработать программу, реализующую работу с коллекциями.
   2. Программа должна быть разработана в виде консольного приложения на языке C#.
   3. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».
   4. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.
   5. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
   6. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.
   7. 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.
   8. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:
   9. public void Push(T element) – добавление в стек;
   10. public T Pop() – чтение с удалением из стека.
   11. Пример работы класса SimpleStack реализовать на основе геометрических фигур.
2. **Текст программы**

**Элемент Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

class Program

{

static void Main\_menu()

{

Console.WriteLine();

Console.WriteLine("выберите действие:");

Console.WriteLine();

Console.WriteLine("1.работать с ArrayList");

Console.WriteLine("2.работать с List");

Console.WriteLine("3.работать с разреженной матрицей");

Console.WriteLine("4.работать с Simple Stack");

Console.WriteLine("5.выход");

Console.WriteLine();

}

static int Main(string[] args)

{

#region

int n = 0;

ArrayList arli = new ArrayList();

List <Geometric\_figures> li = new List<Geometric\_figures>();

double len;

Rectangle rect = new Rectangle(0, 0);

Console.WriteLine("создание прямоугольника");

Console.WriteLine("пожалуйста введите значение");

Console.Write("длина 1 ");

len = Double.Parse(Console.ReadLine());

rect.length1 = len;

Console.Write("длина 2 ");

len = Double.Parse(Console.ReadLine());

rect.length2 = len;

Square scv = new Square(0);

Console.WriteLine("создание квадрата");

Console.WriteLine("пожалуйста введите значение");

Console.Write("длина ");

len = Double.Parse(Console.ReadLine());

scv.length1 = len;

scv.length2 = len;

Circle cir = new Circle(0);

Console.WriteLine("создание круга");

Console.WriteLine("пожалуйста введите значение");

Console.Write("радиус ");

len = Double.Parse(Console.ReadLine());

cir.radius = len;

arli.Add(rect);

li.Add(rect);

arli.Add(scv);

li.Add(scv);

arli.Add(cir);

li.Add(cir);

#endregion

while (n != 5)

{

Main\_menu();

n = int.Parse(Console.ReadLine());

switch (n)

{

case 1:

{

int yeah;

Console.WriteLine("как отсортировать эту коллекцию?");

Console.WriteLine(" 1. по возрастанию");

Console.WriteLine(" 2. по убыванию");

yeah = int.Parse(Console.ReadLine());

if (yeah == 1)

for (int j=0; j< arli.Count - 1; j++)

for (int i=0; i<arli.Count-1-j;i++)

{

if (((Geometric\_figures)arli[i]).CompareTo(arli[i+1])==1)

{

Object spec = arli[i];

arli[i] = arli[i + 1];

arli[i + 1] = spec;

}

}

else

for (int j = 0; j < arli.Count - 1; j++)

for (int i = 0; i < arli.Count - 1 - j; i++)

{

if (((Geometric\_figures)arli[i]).CompareTo(arli[i + 1]) == 0)

{

Object spec = arli[i];

arli[i] = arli[i + 1];

arli[i + 1] = spec;

}

}

Console.WriteLine();

foreach (object i in arli)

{

if (i.GetType().Name == "прямоугольник")

{

Console.WriteLine(i.GetType().Name + ":");

((Rectangle)i).Print();

}

else

if (i.GetType().Name == "квадрат")

{

Console.WriteLine(i.GetType().Name + ":");

((Square)i).Print();

}

else

if (i.GetType().Name == "круг")

{

Console.WriteLine(i.GetType().Name + ":");

((Circle)i).Print();

}

}

break;

}

case 2:

{

int yeah;

Console.WriteLine("как отсортировать эту коллекцию?");

Console.WriteLine(" 1. по возрастанию");

Console.WriteLine(" 2. по убыванию");

yeah = int.Parse(Console.ReadLine());

if (yeah == 1)

for (int j = 0; j < li.Count - 1; j++)

for (int i = 0; i < li.Count - 1 - j; i++)

{

if (((Geometric\_figures)li[i]).CompareTo(li[i + 1]) == 0)

{

Object spec = li[i];

li[i] = li[i + 1];

li[i + 1] = (Geometric\_figures)spec;

}

}

else

for (int j = 0; j < li.Count - 1; j++)

for (int i = 0; i < li.Count - 1 - j; i++)

{

if (((Geometric\_figures)li[i]).CompareTo(li[i + 1]) == 1)

{

Object spec = li[i];

li[i] = li[i + 1];

li[i + 1] = (Geometric\_figures)spec;

}

}

foreach (object i in li)

{

if (i.GetType().Name == "прямоугольник")

{

Console.WriteLine(i.GetType().Name + ":");

((Rectangle)i).Print();

}

else

if (i.GetType().Name == "квадрат")

{

Console.WriteLine(i.GetType().Name + ":");

((Square)i).Print();

}

else

if (i.GetType().Name == "круг")

{

Console.WriteLine(i.GetType().Name + ":");

((Circle)i).Print();

}

}

break;

}

case 3:

{

Console.WriteLine("\nматрица");

Matrix<Geometric\_figures> matrix = new Matrix<Geometric\_figures>(3, 3, 3, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = scv;

matrix[2, 2, 2] = cir;

Console.WriteLine(matrix.ToString());

break;

}

case 4:

{

SimpleStack<Geometric\_figures> stack = new SimpleStack<Geometric\_figures>();

stack.Push(rect);

stack.Push(scv);

stack.Push(cir);

while (stack.Count > 0)

{

Geometric\_figures f = stack.Pop();

Console.WriteLine(f);

}

break;

}

case 5:

{

Console.WriteLine("до свидания");

Console.ReadKey();

break;

}

default:

{

Console.WriteLine("ошибка");

}

break;

}

}

return 0;

}

}

}

**Элемент Rectangle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

class Rectangle : Geometric\_figures, IPrint

{

protected double \_length1=0;

protected double \_length2=0;

public Rectangle()

{ }

public Rectangle(double yourlength1, double yourlength2)

{

\_length1 = yourlength1;

\_length2 = yourlength2;

}

public double length1

{

get { return \_length1; }

set

{

\_length1 = value;

area = value \* length2;

}

}

public double length2

{

get { return \_length2; }

set

{

\_length2 = value;

area = value \* length1;

}

}

public override double Area

{

get

{

return area;

}

}

public override string ToString()

{

return "длина сторон: " + \_length1.ToString() +", "+ \_length2.ToString() + "; площадь: " + area.ToString() + ";";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Элемент Square.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

class Square: Rectangle, IPrint

{

public Square(double yourlength)

{

\_length1 = \_length2 = yourlength;

}

public override string ToString()

{

return "длина стороны: " + \_length1.ToString() + "; площадь: " + area.ToString() + ";";

}

}

}

**Элемент Circle.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

class Circle : Geometric\_figures,IPrint

{

private double \_radius;

public override double Area

{

get

{

return area;

}

}

public Circle(double yourradius)

{

\_radius = yourradius;

}

public double radius

{

get { return \_radius; }

set

{

\_radius = value;

area = Math.Pow(value, 2) \* Math.PI;

}

}

public override string ToString()

{

return "радиус круга " + \_radius.ToString() + "; площадь: " + area.ToString() + ";";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Элемент FigureMatrixCheckEmpty.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Geometric\_figures>

{

public Geometric\_figures getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Geometric\_figures element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

**Элемент Geometric\_figures.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

abstract class Geometric\_figures: IComparable

{

protected double area;

public abstract double Area {get;}

public virtual void finding\_area()

{ }

public int CompareTo(object o)

{

if (Area > ((Geometric\_figures)o).Area)

return 1;

else

return 0;

}

}

}

**Элемент IMatrixCheckEmpty.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

**Элемент IPrint.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Collections;

namespace Lab3

{

interface IPrint

{

void Print();

}

}

**Элемент Matrix.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

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 + " вне диапозона");

}

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

{

throw new ArgumentOutOfRangeException("y", "y=" + y + " вне диапозона");

}

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

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " вне диапозона");

}

}

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("\n");

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();

}

}

}

**Элемент SimpleList.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

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

{

protected SimpleListItem<T> first = null;

protected SimpleListItem<T> last = null;

int \_count;

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

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("вне диапозона");

}

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;

}

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

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 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);

}

}

}

**Элемент SimpleListItem.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

}

**Элемент SimpleStack.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Lab3

{

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;

}

}

}

1. **Экранные формы с примерами выполнения программы**

