**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

«Работа с коллекциями»

|  |  |  |
| --- | --- | --- |
| Выполнила: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Перова Анна Евгеньевна |  | Гапанюк Юрий  Евгеньевич |
| Подпись и дата: |  | Подпись и дата:  09.11.20 |

Москва, 2020 г.

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 (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.

Текст программы

**Figure.cs**

using System;

namespace лаба3

{

interface IPrint

{

void Print();

}

abstract class Figure : IComparable

{

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public abstract double Area();

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.Area() < p.Area()) return -1;

else if (this.Area() == p.Area()) return 0;

else return 1;

}

}

class Rectangle : Figure, IPrint

{

public double width { get; set; }

public double height { get; set; }

public Rectangle(double ph, double pw)

{

this.height = ph;

this.width = pw;

this.Type = "Прямоугольник";

}

public override double Area()

{

double Result = this.width \* this.height;

return Result;

}

public override string ToString()

{

return "Rectangle {width: " + width + ", height: " + height + ", square: " + Area() + "}";

//return string.Format("Width: {0}, Height: {1}, Area: {2}", width, height, Area());

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class Square : Rectangle

{

public double Size { get; set; }

public Square(double size)

: base(size, size)

{

this.Type = "Квадрат";

Size = size;

}

public override string ToString()

{

return "Square {width: " + Size + ", area: " + Area() + "}";

//return string.Format("Size: {0}, Area: {1}", Size, Area());

}

public new void Print()

{

Console.WriteLine(ToString());

}

}

class Circle : Figure, IPrint

{

public double radius { get; set; }

public Circle(double pr)

{

this.radius = pr;

this.Type = "Круг";

}

public override double Area()

{

double Result = Math.PI \* this.radius \* this.radius;

return Result;

}

public override string ToString()

{

return "Circle {radius: " + radius + ", square: " + Area() + "}";

//return string.Format("Radius: {0}, Area: {1:f2}", radius, Area());

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class NullFigure : Figure, IPrint

{

public override double Area()

{

return 0;

}

public override string ToString()

{

return "None";

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

**Matrix.cs**

using System;

using System.Collections.Generic;

using System.Text;

namespace лаба3

{

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

T nullElement;

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

{

maxX = px;

maxY = py;

maxZ = pz;

nullElement = nullElementParam;

}

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

{

get

{

CheckBounds(x, y, z);

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

return (\_matrix.ContainsKey(key)) ? \_matrix[key] : nullElement;

}

set

{

CheckBounds(x, y, z);

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

\_matrix.Add(key, value);

}

}

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

{

if (x < 0 || x >= maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= 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 z = 0; z != maxZ; ++z)

{

for (int j = 0; j != maxY; ++j)

{

b.Append("[ ");

for (int i = 0; i != maxX; ++i)

{

if (i > 0) b.Append(" | ");

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

}

b.Append(" ]\n");

}

b.Append("\n");

}

return b.ToString();

}

}

}

**SimpleList.cs**

using System;

using System.Collections.Generic;

namespace лаба3

{

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;

}

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);

}

}

}

**SimpleListItem.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace лаба3

{

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

}

**SimpleStack.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace лаба3

{

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;

}

}

}

**Program.cs**

using System;

using System.Collections;

using System.Collections.Generic;

namespace лаба3

{

class Program

{

static void Main(string[] args)

{

Rectangle rect= new Rectangle(5, 7);

Square sq = new Square(8);

Circle circ = new Circle(3);

Console.WriteLine("Before sorting:\n\t" + rect + "\n\t" + sq + "\n\t" + circ);

ArrayList figuresArrayList = new ArrayList();

figuresArrayList.Add(rect);

figuresArrayList.Add(sq);

figuresArrayList.Add(circ);

figuresArrayList.Sort();

Console.WriteLine("\nArrayList sorting: ");

foreach (object obj in figuresArrayList)

Console.WriteLine("\t" + obj);

List<Figure> figuresList = new List<Figure> {rect, sq, circ};

figuresList.Sort();

Console.WriteLine("\nList sorting: ");

foreach (Figure figure in figuresList)

Console.WriteLine("\t" + figure);

Console.Write("\nMatrix: ");

Matrix<Figure> matrix = new Matrix<Figure>(4, 3, 2, new NullFigure());

matrix[0, 1, 1] = rect;

matrix[2, 2, 0] = sq;

matrix[1, 2, 1] = circ;

Console.Write("\n" + matrix);

Console.WriteLine("\nElements in stack: ");

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

figuresStack.Push(rect);

figuresStack.Push(circ);

figuresStack.Push(sq);

while (figuresStack.Count != 0)

Console.WriteLine("\t" + figuresStack.Pop());

Console.ReadKey();

}

}

}

Скриншот с примером работы программы

