Московский государственный технический университет и Н.Э. Баумана

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-5 «Системы обработки информации и управления»

ОТЧЕТ

по лабораторной работе №3

Выполнил:

Студент ИУ5-32Б

Нырков Илья Алексеевич

Проверил:

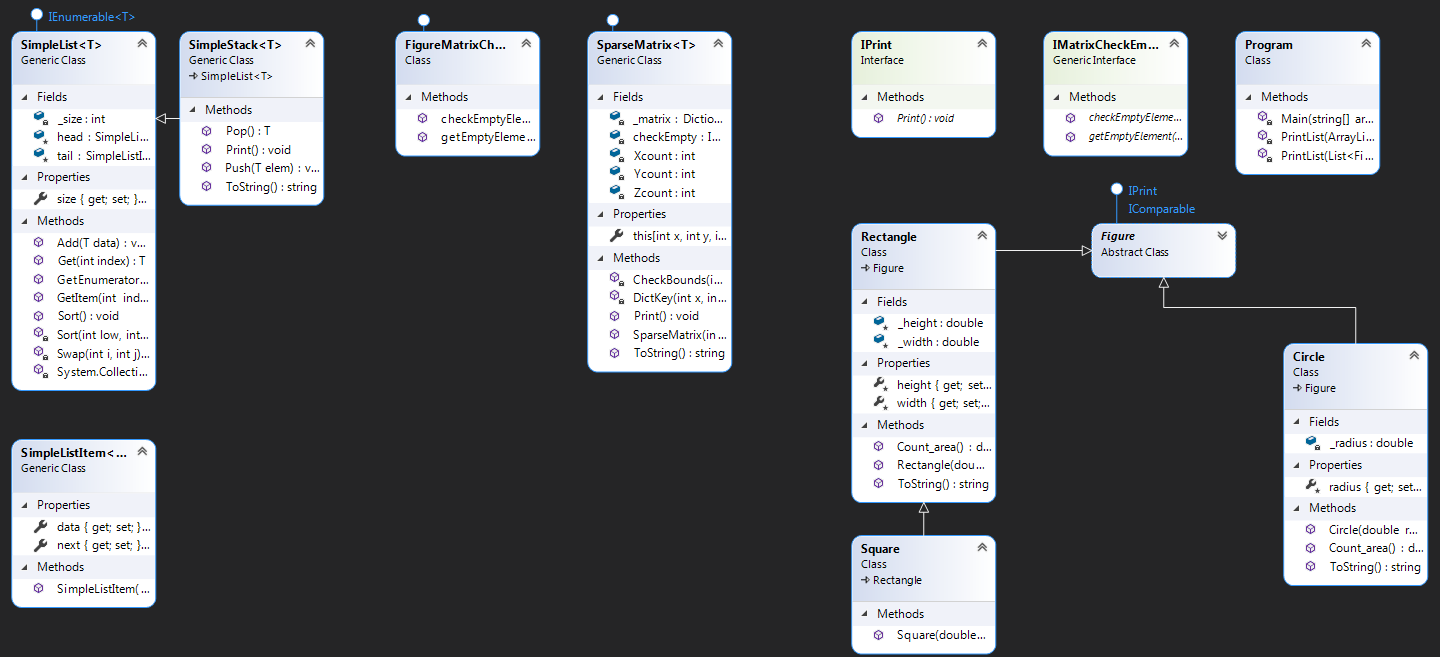
Доцент Гапанюк Ю. Е.

МОСКВА 2020

**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<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.

**Диаграмма классов:**

****

**Листинг кода программы:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Geometric\_figures

{

interface IPrint

{

void Print();

}

abstract class Figure : IPrint, IComparable

{

public abstract double Count\_area();

protected string Type;

public void Print()

{

Console.WriteLine(this.ToString());

}

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.Count\_area() < p.Count\_area()) return -1;

else if (this.Count\_area() == p.Count\_area()) return 0;

else return 1;

}

}

class Rectangle : Figure

{

protected double \_height;

protected double height

{

get { return \_height; }

set

{

if (value < 0)

throw new ArgumentOutOfRangeException(

"height must not be negative");

\_height = value;

}

}

protected double \_width;

protected double width

{

get { return \_width; }

set

{

if (value < 0)

throw new ArgumentOutOfRangeException(

"width must not be negative");

\_width = value;

}

}

public Rectangle(double height, double width)

{

this.height = height;

this.width = width;

this.Type = "Rectangle";

}

public override double Count\_area()

{

return height \* width;

}

public override string ToString()

{

return Type + " with Area: " + Count\_area();

}

}

class Square : Rectangle

{

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

{

Type = "Square";

}

}

class Circle : Figure

{

private double \_radius;

protected double radius

{

get { return \_radius; }

set

{

if (value < 0)

throw new ArgumentOutOfRangeException(

"radius must not be negative");

\_radius = value;

}

}

public override double Count\_area()

{

return 3.1415 \* \_radius \* \_radius;

}

public Circle(double radius)

{

this.radius = radius;

Type = "Circle";

}

public override string ToString()

{

return Type + " with Area: " + Count\_area();

}

}

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

public Figure getEmptyElement()

{

return null;

}

}

//разряженная матрица

class SparseMatrix<T> : IPrint

{

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

int Xcount;

int Ycount;

int Zcount;

IMatrixCheckEmpty<T> checkEmpty;

public SparseMatrix(int px, int py, int pz,

IMatrixCheckEmpty<T> checkEmptyParam)

{

this.Xcount = px;

this.Ycount = py;

this.Zcount = pz;

this.checkEmpty = checkEmptyParam;

}

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.checkEmpty.getEmptyElement();

}

}

}

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

{

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

{

throw new ArgumentOutOfRangeException

("x= " + x + " is out of range\n");

}

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

{

throw new ArgumentOutOfRangeException

("y= " + y + " is out of range\n");

}

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

{

throw new ArgumentOutOfRangeException

("z= " + z + " is out of range\n");

}

}

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.Zcount; i++)

{

b.Append("z = " + i.ToString() + "\n");

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

{

b.Append("|");

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

{

if (k > 0)

b.Append("\t");

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

{

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

}

else

{

b.Append(" - ");

}

}

b.Append("|\n");

}

b.Append("\n");

}

return b.ToString();

}

public void Print()

{

Console.WriteLine(ToString());

}

}

//элемент списка

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> head = null;

protected SimpleListItem<T> tail = null;

int \_size;

public int size

{

get { return \_size; }

protected set { \_size = value; }

}

public void Add(T data)

{

SimpleListItem<T> newItem =

new SimpleListItem<T>(data);

this.\_size++;

if (tail == null)

{

this.head = newItem;

this.tail = newItem;

}

else

{

this.tail.next = newItem;

this.tail = newItem;

}

}

public SimpleListItem<T> GetItem(int index)

{

if ((index < 0) || (index >= this.\_size))

{

throw new ArgumentException("Index out of range");

}

SimpleListItem<T> current = this.head;

int i = 0;

while ( i < index)

{

current = current.next;

i++;

}

return current;

}

public T Get(int index)

{

return GetItem(index).data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.head;

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.size - 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 SimpleStack<T> : SimpleList<T> where T : IComparable, IPrint

{

public void Push(T elem)

{

Add(elem);

}

public T Pop()

{

T Result = default(T);

if (this.size == 0) return Result;

if (this.size == 1)

{

Result = this.head.data;

this.head = null;

this.tail = null;

}

else

{

SimpleListItem<T> newTail = this.GetItem(this.size - 2);

Result = newTail.next.data;

this.tail = newTail;

newTail.next = null;

}

this.size--;

return Result;

}

public override string ToString()

{

StringBuilder str = new StringBuilder();

for (int i = this.size - 1; i >= 0; i--)

{

str.Append(this.Get(i).ToString() + "\n");

}

return str.ToString();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Program

{

static void PrintList(List<Figure> list)

{

foreach (Figure i in list){

i.Print();

}

}

static void PrintList(ArrayList list)

{

foreach (object i in list)

{

Figure p = (Figure)i;

p.Print();

}

}

static void Main(string[] args)

{

Circle circle = new Circle(4);

Circle circle2 = new Circle(100);

Square square = new Square(10);

Square square2 = new Square(1);

Rectangle rectangle = new Rectangle(4, 5);

List<Figure> list\_of\_figs = new List<Figure>

{

circle2, circle, rectangle, square, square2

};

Console.WriteLine("List before sort:");

PrintList(list\_of\_figs);

list\_of\_figs.Sort();

Console.WriteLine("\nList after sort:");

PrintList(list\_of\_figs);

ArrayList list\_of\_figs2 = new ArrayList

{

circle2, circle, rectangle, square, square2

};

Console.WriteLine("\n\n\n\nArrayList before sort:");

PrintList(list\_of\_figs2);

list\_of\_figs2.Sort();

Console.WriteLine("\nArrayList after sort:");

PrintList(list\_of\_figs2);

Console.WriteLine("\n\n\n\nSparseMatrix:");

SparseMatrix<Figure> sparsematrix =

new SparseMatrix<Figure>(3, 3, 3, new FigureMatrixCheckEmpty());

sparsematrix[0, 1, 0] = circle;

sparsematrix[1, 2, 0] = circle2;

sparsematrix[0, 0, 0] = square;

sparsematrix[2, 2, 2] = square2;

sparsematrix[1, 2, 2] = rectangle;

sparsematrix.Print();

Console.WriteLine("\n\n\n\nSimpleStack:\n\n\n");

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

stack.Push(circle2);

stack.Push(circle);

stack.Push(square);

stack.Push(square2);

stack.Push(rectangle);

Console.WriteLine("Print method:");

stack.Print();

Console.WriteLine("\nPop method:");

while(stack.size > 0)

{

Figure f = stack.Pop();

f.Print();

}

}

}

}

**Скриншоты работы программы**

