**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35б |  | преподаватель каф. ИУ5 |
| Хижняков Вадим |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

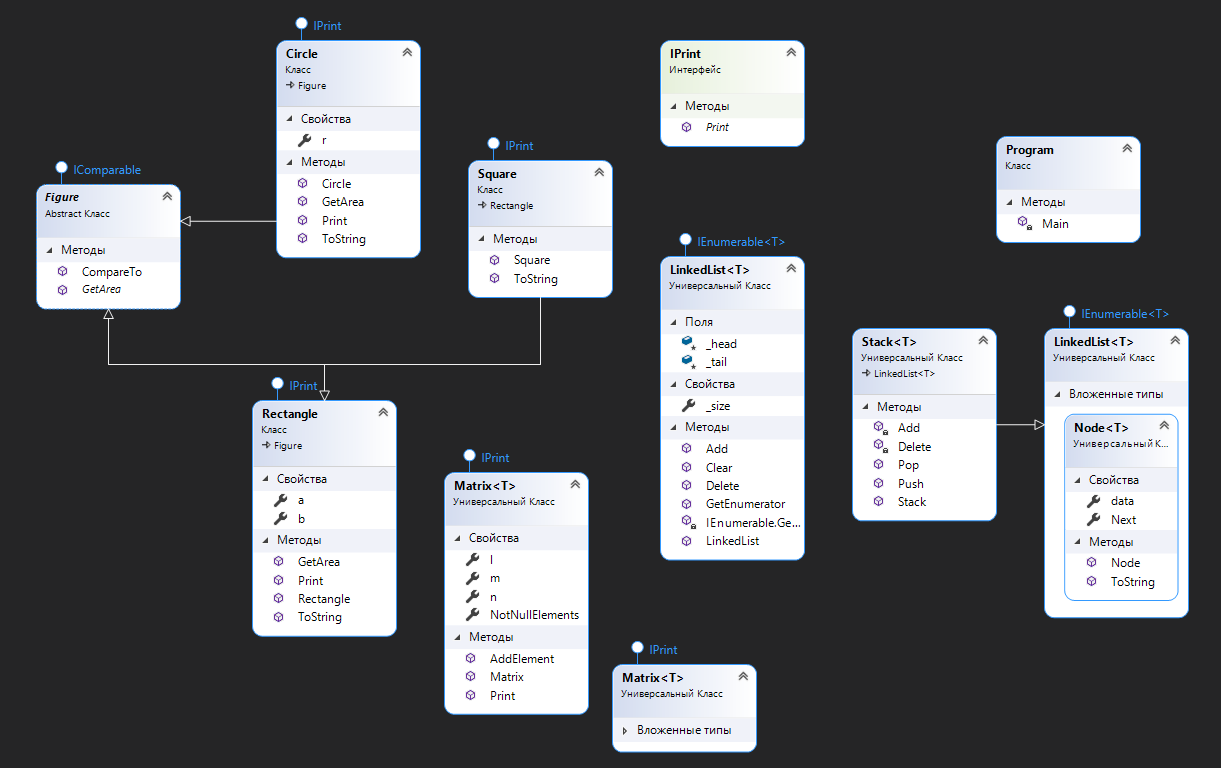
# Задание.

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.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

# Диаграмма классов



# Текст программы

Figures.cs

using Microsoft.VisualBasic.CompilerServices;

using System;

using System.Collections.Generic;

using System.Reflection.Metadata.Ecma335;

using System.Text;

namespace Sem3Lab3\_Collections

{

abstract class Figure : IComparable

{

public int CompareTo(object o)

{

if (o == null)

return 1;

Figure f = o as Figure;

if (f != null)

{

return this.GetArea().CompareTo(f.GetArea());

}

else

{

throw new Exception("Невозможно сравнить");

}

}

public abstract double GetArea();

}

interface IPrint

{

public void Print();

}

class Rectangle : Figure, IPrint

{

public double a { get; set; } = 0;

public double b { get; set; } = 0;

public Rectangle(double a, double b)

{

this.a = a;

this.b = b;

}

public override double GetArea()

{

return a \* b;

}

public override string ToString()

{

return "Lenght: " + a.ToString() + " Width: " + b.ToString() + " Area: " + this.GetArea().ToString();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

public Square(double a) : base(a, a) { }

public override string ToString()

{

return "Side: " + a.ToString() + " Area: " + this.GetArea().ToString();

}

}

class Circle : Figure, IPrint

{

public double r { get; set; } = 0;

public Circle(double r)

{

this.r = r;

}

public override double GetArea()

{

return Math.PI \* Math.Pow(r, 2);

}

public override string ToString()

{

return "Radius: " + r.ToString() + " Area: " + this.GetArea().ToString();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

LinkedList.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Security.Cryptography.X509Certificates;

using System.Text;

namespace Sem3Lab3\_Collections

{

partial class LinkedList<T> : IEnumerable<T>

{

protected Node<T> \_head;

protected Node<T> \_tail;

public int \_size { get; protected set; }

public LinkedList()

{

this.\_head = null;

this.\_tail = null;

this.\_size = 0;

}

public void Add(T data)

{

if (data == null)

{

throw new ArgumentNullException(nameof(data));

}

var node = new Node<T>(data);

if (\_head == null)

{

this.\_head = node;

}

else

{

this.\_tail.Next = node;

}

\_tail = node;

\_size++;

}

public void Delete(T data)

{

if (data == null)

{

throw new ArgumentNullException(nameof(data));

}

var current = this.\_head;

Node<T> previous = null;

while (current != null)

{

if (current.data.Equals(data))

{

if (previous != null)

{

previous.Next = current.Next;

if (current.Next == null)

{

this.\_tail = previous;

}

}

else

{

this.\_head = this.\_head.Next;

if (this.\_head == null)

{

this.\_tail = null;

}

}

this.\_size--;

break;

}

previous = current;

current = current.Next;

}

}

public void Clear()

{

this.\_head = null;

this.\_tail = null;

this.\_size = 0;

}

public IEnumerator<T> GetEnumerator()

{

var current = this.\_head;

while (current != null)

{

yield return current.data;

current = current.Next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return ((IEnumerable)this).GetEnumerator();

}

}

}

Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection.Metadata.Ecma335;

using System.Text;

namespace Sem3Lab3\_Collections

{

partial class Matrix<T> : IPrint

{

public List<MatrixElement<T>> NotNullElements { get; private set; }

public int n { get; private set; }

public int m { get; private set; }

public int l { get; private set; }

public Matrix(int n, int m, int l)

{

this.n = n;

this.m = m;

this.l = l;

this.NotNullElements = new List<MatrixElement<T>>();

}

public void AddElement(T data, int x, int y, int z)

{

if (x > this.n || x <= 0 || y > this.m || y <= 0 || z > this.l || z <= 0)

{

throw new InvalidOperationException("Invalid indexes");

}

var newElem = new MatrixElement<T>(data, x, y, z);

this.NotNullElements.Add(newElem);

}

public void Print()

{

for (int i = 0; i < this.n; i++)

{

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

{

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

{

var notNullElem = this.NotNullElements.Where(item => item.x - 1 == i && item.y - 1 == j && item.z - 1 == k).ToList();

if (notNullElem.Any())

{

Console.Write(notNullElem.First().ToString() + "\t");

}

else

{

Console.Write("0 \t");

}

}

Console.WriteLine();

}

Console.WriteLine();

}

}

}

}

MatrixElement.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Sem3Lab3\_Collections

{

partial class Matrix<T>

{

public class MatrixElement<T>

{

public int x { get; set; }

public int y { get; set; }

public int z { get; set; }

public T data { get; set; }

public MatrixElement(T data, int x, int y, int z)

{

this.x = x;

this.y = y;

this.z = z;

this.data = data;

}

public override string ToString()

{

return "(" + this.x.ToString() + "," + this.y.ToString() + "," + this.z.ToString() + ":" + this.data.ToString() + ")";

}

}

}

}

Node.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Sem3Lab3\_Collections

{

partial class LinkedList<T> : IEnumerable<T>

{

public class Node<T>

{

public T data { get; set; } = default(T);

public Node<T> Next { get; set; } = null;

public Node(T data)

{

if (data != null)

{

this.data = data;

}

else

{

throw new ArgumentNullException(nameof(data));

}

}

public override string ToString()

{

return this.data.ToString();

}

}

}

}

Stack.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Sem3Lab3\_Collections

{

class Stack<T> : LinkedList<T>

{

private new void Add(T data) { }

private new void Delete(T data) { }

public Stack() : base() { }

public void Push(T data)

{

var node = new Node<T>(data);

node.Next = this.\_head;

this.\_head = node;

this.\_size++;

}

public T Pop()

{

var tmp = this.\_head;

this.\_head = this.\_head.Next;

this.\_size--;

return tmp.data;

}

}

}

Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

namespace Sem3Lab3\_Collections

{

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(4, 5);

Square sqr = new Square(6);

Circle crcl = new Circle(1);

ArrayList figures = new ArrayList();

figures.Add(rect);

figures.Add(sqr);

figures.Add(crcl);

try

{

figures.Sort();

}

catch (Exception e)

{

throw e;

}

foreach (var figure in figures)

{

Console.WriteLine(figure.ToString());

}

Console.WriteLine("-------------------------------------------");

List<Figure> figures\_list = new List<Figure>();

figures\_list.Add(rect);

figures\_list.Add(sqr);

figures\_list.Add(crcl);

try

{

figures\_list.Sort();

}

catch (Exception e)

{

throw e;

}

foreach (var figure in figures\_list)

{

Console.WriteLine(figure);

}

LinkedList<int> list = new LinkedList<int>();

list.Add(3);

list.Add(5);

list.Add(8);

list.Add(5);

Console.WriteLine();

foreach (var item in list)

{

Console.WriteLine(item);

}

Console.WriteLine();

list.Delete(5);

foreach (var item in list)

{

Console.WriteLine(item);

}

Console.WriteLine("------------------------------------------");

Stack<int> stack = new Stack<int>();

stack.Push(3);

stack.Push(7);

foreach (var item in stack)

{

Console.WriteLine(stack.Pop());

}

////////////////////////////////////////////////////////////

Console.WriteLine();

Matrix<int> m = new Matrix<int>(2, 3, 4);

m.AddElement(99, 1, 2, 3);

m.Print();

}

}

}

# Результаты

