**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра «Системы обработки информации и управления»

Дисциплина «Базовые компоненты интернет-технологий»

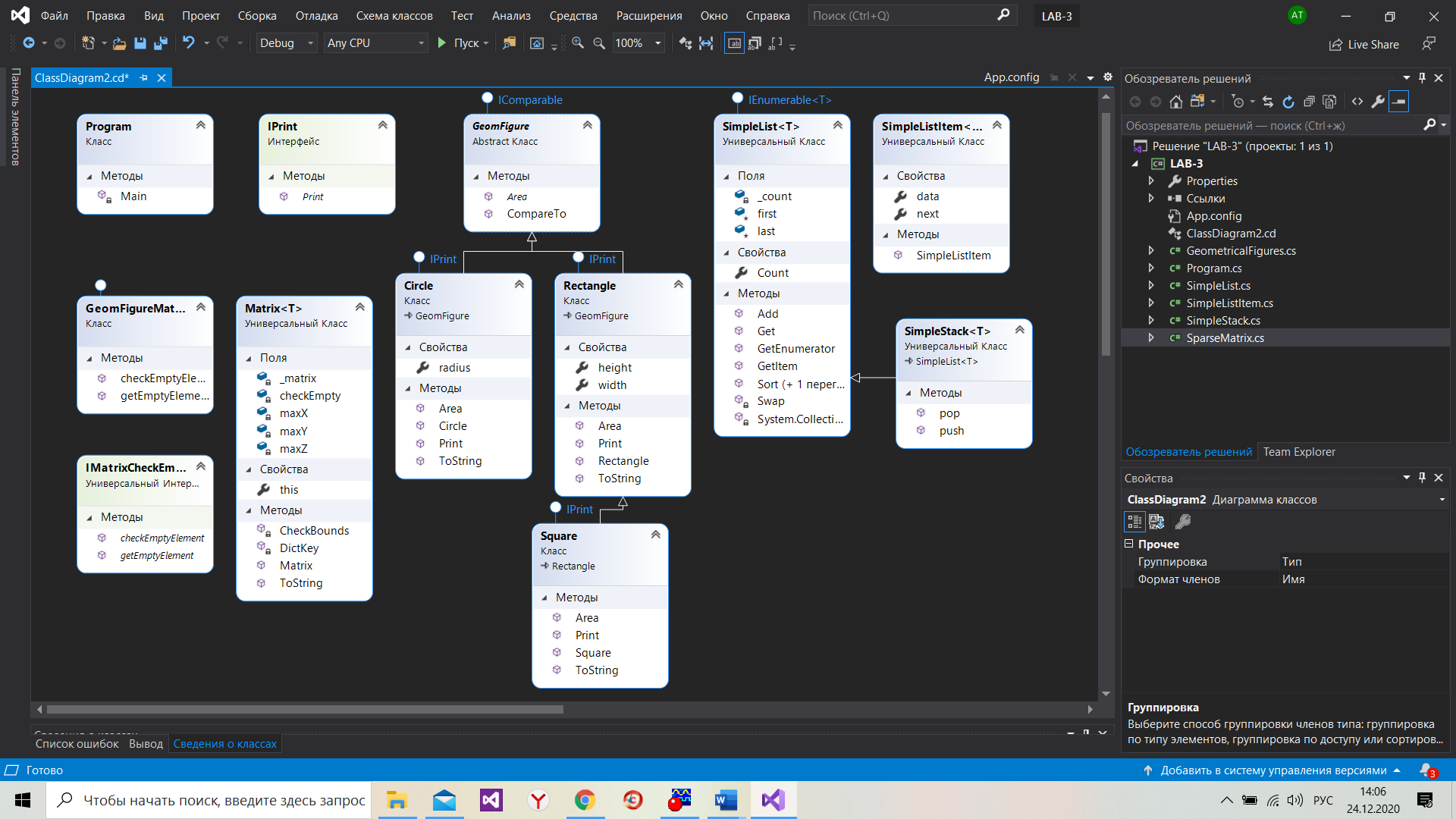
Отчет по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы  ИУ5-35Б |  | преподаватель каф. ИУ5 |
| Озеров Павел |  | Гапанюк Ю.Е. |

**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:**

**Текст программы:**

Файл GeometricalFigures.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LAB\_3

{ // ВСЕ КЛАССЫ ФИГУР (с наследованием) + ИНТЕРФЕЙС

interface IPrint

{

void Print();

}

public abstract class GeomFigure : IComparable

{

public abstract double Area();

public int CompareTo(object obj)

{

GeomFigure p = obj as GeomFigure;

if (this.Area() > p.Area())

return 1;

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

return 0;

else

return -1;

}

}

class Rectangle : GeomFigure, IPrint

{

public double width { get; set; }

public double height { get; set; }

public Rectangle(double width, double height)

{

this.width = width;

this.height = height;

}

public override double Area()

{

return width \* height;

}

public override string ToString()

{

return $"Ширина:{width} Высота:{height} Площадь прямоугольника:{Area()}";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

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

{

}

public override double Area()

{

return width \* width;

}

public override string ToString()

{

return $"Длина стороны:{width} Площадь квадрата:{Area()}";

}

new public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Circle : GeomFigure, IPrint

{

public double radius { get; set; }

public Circle(double radius)

{

this.radius = radius;

}

public override double Area()

{

return Math.PI \* radius \* radius;

}

public override string ToString()

{

return $"Радиус:{radius} Площадь круга:{Area()}";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

Файл Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LAB\_3

{ // ВСЕ КЛАССЫ ФИГУР (с наследованием) + ИНТЕРФЕЙС

interface IPrint

{

void Print();

}

public abstract class GeomFigure : IComparable

{

public abstract double Area();

public int CompareTo(object obj)

{

GeomFigure p = obj as GeomFigure;

if (this.Area() > p.Area())

return 1;

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

return 0;

else

return -1;

}

}

class Rectangle : GeomFigure, IPrint

{

public double width { get; set; }

public double height { get; set; }

public Rectangle(double width, double height)

{

this.width = width;

this.height = height;

}

public override double Area()

{

return width \* height;

}

public override string ToString()

{

return $"Ширина:{width} Высота:{height} Площадь прямоугольника:{Area()}";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

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

{

}

public override double Area()

{

return width \* width;

}

public override string ToString()

{

return $"Длина стороны:{width} Площадь квадрата:{Area()}";

}

new public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Circle : GeomFigure, IPrint

{

public double radius { get; set; }

public Circle(double radius)

{

this.radius = radius;

}

public override double Area()

{

return Math.PI \* radius \* radius;

}

public override string ToString()

{

return $"Радиус:{radius} Площадь круга:{Area()}";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

Файл SimpleList.cs

using System;

using System.Collections;

using System.Collections.Generic;

namespace LAB\_3

{

public class SimpleList<T> : IEnumerable<T> where T : IComparable //Класс SimpleList реализует список

{

protected SimpleListItem<T> first = null; // Первый элемент списка

protected SimpleListItem<T> last = null;// Последний элемент списка

public int Count // Количество элементов

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

public void Add(T element) // Добавление элемента

{

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) //Чтение контейнера с заданным номером

{

if ((number < 0) || number > Count)

{

throw new Exception("Выход за границу массива");

}

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;

}// Чтение элемента с заданным номером

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;

}

}

public void Sort()//сортировка

{

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);

} //быстрая сортировка

private void Swap(int i, int j) //вспомогательный метод, который помогает при сортировке

{

SimpleListItem<T> ci = GetItem(i);

SimpleListItem<T> cj = GetItem(j);

T temp = ci.data;

ci.data = cj.data;

cj.data = temp;

}

}

}

Файл SimpleListItem.cs

using System;

using System.Collections;

using System.Collections.Generic;

namespace LAB\_3

{

public class SimpleListItem<T>//контейнер элемента списка

{

public T data { get; set; } //Тип-обобщение T – тип данных списка

public SimpleListItem<T> next { get; set; } //"указатель" на след.элемент

public SimpleListItem(T param)//конструктор public SimpleListItem

{

this.data = param;

}

}

}

Файл SimpleStack.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LAB\_3

{

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

{

public void push(T element)//Вставляет объект как верхний элемент стека

{

this.Add(element);

}

public T pop() //Удаляет и возвращает объект, находящийся в начале SimpleStack

{

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;

}

}

}

Файл SparseMatrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LAB\_3

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

public class Matrix<T>//Класс «Разреженная матрица» реализован в виде обобщенной коллекции, класс-обобщение T

{

Dictionary<string, T> \_matrix = new Dictionary<String, T>();//словарь для хранения значений

int maxX;//кол-во элементов по горизонтали

int maxY;

int maxZ;

IMatrixCheckEmpty<T> checkEmpty;// Реализация интерфейса для проверки пустого элемента

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> checkEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = 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)//проверка границ

{

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

{

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

}

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

{

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

}

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

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " выходит за границы");

}

}

string DictKey(int x, int y, int z)

{

return x.ToString() + "\_" + y.ToString() + "\_" + z.ToString();

}

public override string ToString() //ПЕРЕОПРЕДЕЛЕНИЕ ВЫВОДА ДЛЯ УДОБСТВА ПРОСМОТРА

{

StringBuilder b = new StringBuilder();//Чтобы изменять строку без создания нового объекта, можно использовать класс System.Text.StringBuilder

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

{

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

{

b.Append("[");//Append позволяет добавить текст или строковое представление объекта к концу строки

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

{

if (i > 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();

}

}

class GeomFigureMatrixCheckEmpty : IMatrixCheckEmpty<GeomFigure> //наследование интерфейся для проверки пустого элемента

{

public GeomFigure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(GeomFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

**Экранные формы:**

