**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»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Корчевский Александр |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

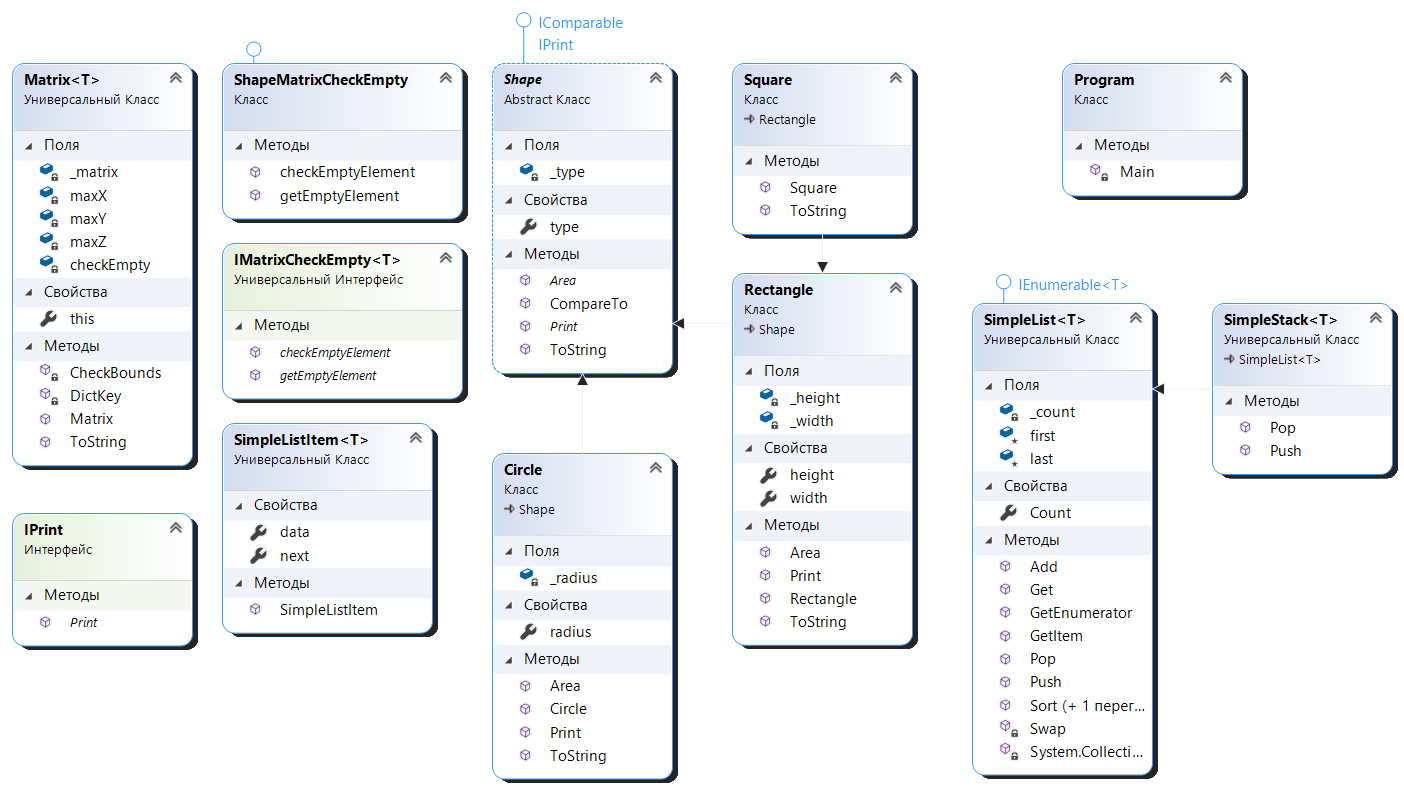
Москва, 2020 г.

**Условие 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:**

using System;

using System.Collections;

using System.Collections.Generic;

using FigureCollections;

using MyShapes;

using Simple;

namespace lab03

{

class Program

{

static void Main(string[] args)

{

Rectangle r = new Rectangle(2,3);

Circle c = new Circle(3);

Square s = new Square(4);

ArrayList ShapesArr = new ArrayList();

ShapesArr.Add(r);

ShapesArr.Add(c);

ShapesArr.Add(s);

ShapesArr.Sort();

Console.WriteLine("Вывод элементов из ArrayList");

foreach(object o in ShapesArr)

{

Console.WriteLine(o.ToString());

}

Console.WriteLine();

List<Shape> ShapesList = new List<Shape>();

ShapesList.Add(r);

ShapesList.Add(c);

ShapesList.Add(s);

Console.WriteLine("Вывод элементов из List<>");

ShapesList.Sort();

foreach(Shape i in ShapesList)

{

i.Print();

}

Console.WriteLine();

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

stack.Push(r);

stack.Push(c);

stack.Push(s);

stack.Pop().Print();

stack.Pop().Print();

stack.Pop().Print();

Console.WriteLine();

Console.WriteLine("\nМатрица");

Matrix<Shape> matrix = new Matrix<Shape>(3, 3, 3, new ShapeMatrixCheckEmpty());

matrix[0, 0, 0] = r;

matrix[1, 1, 1] = c;

matrix[2, 2, 2] = s;

Console.WriteLine(matrix.ToString());

Console.WriteLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using MyShapes;

namespace FigureCollections

{

/// <summary>

/// Проверка пустого элемента матрицы

/// </summary>

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

class ShapeMatrixCheckEmpty : IMatrixCheckEmpty<Shape>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public Shape getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(Shape element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

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

/// <summary>

/// Количество элементов по горизонтали (максимальное количество столбцов)

/// </summary>

int maxX;

/// <summary>

/// Количество элементов по вертикали (максимальное количество строк)

/// </summary>

int maxY;

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

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();

}

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

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.maxY)

{

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

}

}

/// <summary>

/// Формирование ключа

/// </summary>

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

{

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

}

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

StringBuilder b = new StringBuilder();

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

{

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

{

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t\t\t");

}

//Если текущий элемент не пустой

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

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

}

return b.ToString();

}

}

}

using System;

using System.Collections;

using System.Collections.Generic;

namespace MyShapes

{

interface IPrint

{

void Print();

}

abstract class Shape : IComparable, IPrint

{

private string \_type;

public string type

{

get => \_type;

protected set

{

\_type = value;

}

}

public abstract double Area();

public override string ToString()

{

return type + " площадью " + Area().ToString();

}

public abstract void Print();

public int CompareTo(object other)

{

Shape s = other as Shape;

if (this.Area() > s.Area())

{

return 1;

}

if (s.Area() < this.Area())

{

return -1;

}

return 0;

}

}

class Rectangle : Shape

{

private double \_width;

private double \_height;

public double width

{

get => \_width;

set

{

if (value < 0)

{

Console.WriteLine("Width can not be < 0, it is set to 0.");

\_width = 0;

}

else

\_width = value;

}

}

public double height

{

get => \_height;

set

{

if (value < 0)

{

Console.WriteLine("Height can not be < 0, it is set to 0.");

\_height = 0;

}

else

\_height = value;

}

}

public override double Area() => width \* height;

public Rectangle(double Width, double Height)

{

width = Width;

height = Height;

type = "Прямоугольник";

}

public override string ToString()

{

return base.ToString() + ", высота - " + height + ", ширина - " + width;

}

public override void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle

{

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

{

type = "Квадрат";

}

public override string ToString()

{

return type + " площадью " + Area().ToString() + ", длина - " + width;

}

}

class Circle : Shape

{

private double \_radius;

public double radius { get; set; }

public override double Area() => Math.PI \* radius \* radius;

public override string ToString()

{

return base.ToString() + ", радиус - " + radius;

}

public override void Print()

{

Console.WriteLine(ToString());

}

public Circle(double Radius) { radius = Radius; type = "Круг"; }

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using FigureCollections;

namespace Simple

{

class SimpleStack<T> : SimpleList<T>

where T : IComparable

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

T result = default(T);

if (Count == 0)

return result;

if (Count == 1)

{

result = first.data;

first.next = null;

last.next = null;

}

else

{

SimpleListItem<T> newlast = GetItem(Count - 2);

result = newlast.next.data;

last = newlast;

newlast.next = null;

}

Count--;

return result;

}

}

}

using System;

using System.Collections.Generic;

namespace FigureCollections

{

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

T pop = last.data;

return last.data;

}

/// <summary>

/// Первый элемент списка

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент списка

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

/// <summary>

/// Добавление элемента

/// </summary>

public void Add(T element)

{

SimpleListItem<T> newItem = new SimpleListItem<T>(element);

this.Count++;

//Добавление первого элемента

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

//Добавление следующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Просоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

if ((number < 0) || (number >= this.Count))

{

//Можно создать собственный класс исключения

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

}

SimpleListItem<T> current = this.first;

int i = 0;

//Пропускаем нужное количество элементов

while (i < number)

{

//Переход к следующему элементу

current = current.next;

//Увеличение счетчика

i++;

}

return current;

}

/// <summary>

/// Чтение элемента с заданным номером

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Для перебора коллекции

/// </summary>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

//Перебор элементов

while (current != null)

{

//Возврат текущего значения

yield return current.data;

//Переход к следующему элементу

current = current.next;

}

}

//Реализация обобщенного IEnumerator<T> требует реализации необобщенного интерфейса

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Алгоритм быстрой сортировки

/// </summary>

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);

}

/// <summary>

/// Вспомогательный метод для обмена элементов при сортировке

/// </summary>

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;

}

}

}

using System;

namespace FigureCollections

{

/// <summary>

/// Элемент списка

/// </summary>

public class SimpleListItem<T>

{

/// <summary>

/// Данные

/// </summary>

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

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

///конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

}

**Результаты:**

