**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №3

«Работа с коллекциями»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31 |  | преподаватель каф. ИУ5 |
| Агличеев Михаил |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

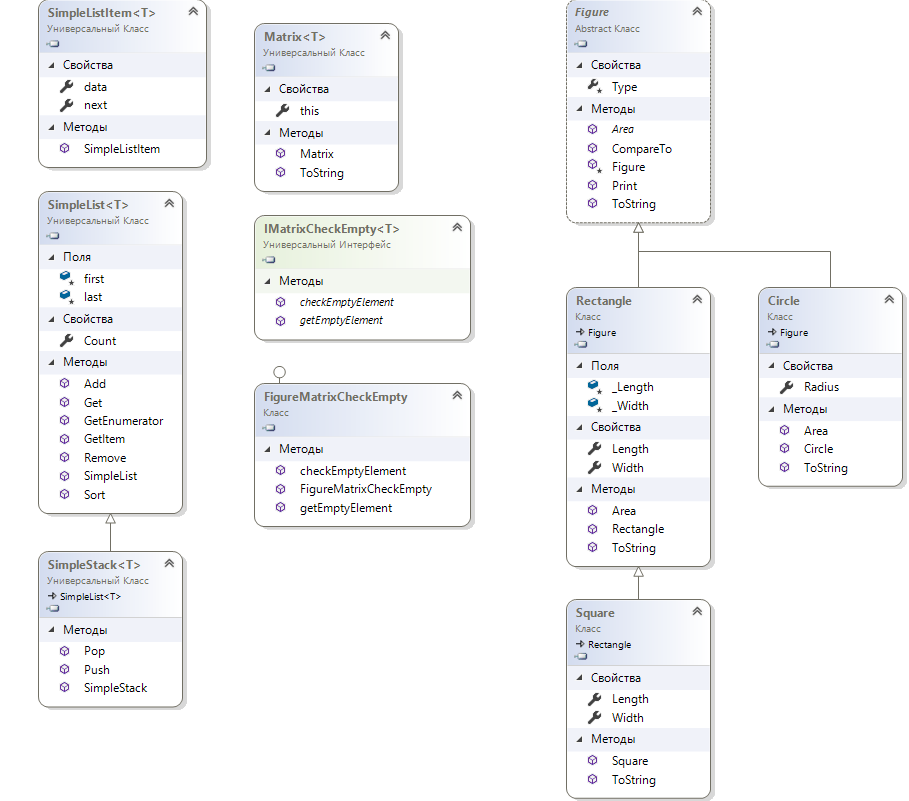
Москва, 2020г.

1. 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.

2. UML-диаграмма классов



3. Листинг программы

Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using FigureProject;

using SimpleListProject;

using MatrixProject;

namespace Лабораторная\_\_\_3

{

class Program

{

public enum Options

{

SortArrayList = 1,

SortListFigure = 2,

DemSparseMatrix = 3,

DemStack = 4,

Exit = 5

}

static Options Menu()

{

Console.WriteLine("Введите номер нужной опции:");

Console.WriteLine("1: Сортировка на основе ArrayList");

Console.WriteLine("2: Сортировка на основе List<Figure>");

Console.WriteLine("3: Демонстрация разряженной матрицы");

Console.WriteLine("4: Демонстрация стека на основе односвязного списка");

Console.WriteLine("5: Выход");

int option = 0;

while (true)

{

if (!int.TryParse(Console.ReadLine(), out option))

{

Console.WriteLine("Неправильный ввод. Повторите ввод");

}

else if (!Enum.IsDefined(typeof(Options), option))

{

Console.WriteLine("Неизвестная опция. Повторите ввод");

}

else

{

return (Options)option;

}

}

}

static void Main(string[] args)

{

Rectangle rectangle = new Rectangle(2.4, 67);

Square square = new Square(1);

Circle circle = new Circle(1);

Options option = Menu();

while (option != Options.Exit)

{

switch (option)

{

case Options.SortArrayList:

{

ArrayList nonGenericList = new ArrayList {

square,

rectangle,

circle

};

Console.WriteLine("До сортировки:");

foreach (Figure figure in nonGenericList)

{

figure.Print();

}

Console.Write("\n");

nonGenericList.Sort();

Console.WriteLine("После сортировки (ArrayList):");

foreach (Figure figure in nonGenericList)

{

figure.Print();

}

Console.Write("\n");

break;

}

case Options.SortListFigure:

List<Figure> GenericList = new List<Figure> {

square,

rectangle,

circle

};

Console.WriteLine("До сортировки:");

foreach (Figure figure in GenericList)

{

figure.Print();

}

Console.Write("\n");

GenericList.Sort();

Console.WriteLine("После сортировки (List<Figure>):");

foreach (Figure figure in GenericList)

{

figure.Print();

}

Console.Write("\n");

break;

case Options.DemSparseMatrix:

Matrix<Figure> matrix = new Matrix<Figure>(2, 2, 2, new FigureMatrixCheckEmpty());

matrix[0, 1, 1] = square;

matrix[1, 0, 1] = rectangle;

matrix[1, 1, 0] = circle;

Console.WriteLine(matrix.ToString());

break;

case Options.DemStack:

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

Console.WriteLine($"Добавляем {square.ToString()} в стэк");

stack.Push(square);

Console.WriteLine($"Добавляем {rectangle.ToString()} в стэк");

stack.Push(rectangle);

Console.WriteLine($"Добавляем {circle.ToString()} в стэк");

stack.Push(circle);

for (int i = 0; i < 3; ++i)

{

Console.WriteLine($"Производим операцию Pop() и удаляем {stack.Pop().ToString()} из стэка");

}

break;

}

Console.Write('\n');

option = Menu();

}

}

}

}

Figure.cs

using System;

using static System.Math;

namespace FigureProject

{

public abstract class Figure : IPrint, IComparable

{

protected string Type { get; set; }

public abstract double Area();

public override string ToString()

{

return this.Type + " площадью " + this.Area();

}

public void Print()

{

Console.WriteLine(this.ToString());

}

public int CompareTo(object other)

{

if (other == null)

{

return 1;

}

return this.Area().CompareTo(((Figure)other).Area());

}

}

interface IPrint

{

void Print();

}

public class Rectangle : Figure

{

protected double \_Length;

public virtual double Length

{

get { return \_Length; }

set

{

if (value >= 0)

{

\_Length = value;

}

else

{

throw new Exception("Длина не может быть отрицательной");

}

}

}

protected double \_Width;

public virtual double Width

{

get { return \_Width; }

set

{

if (value >= 0)

{

\_Width = value;

}

else

{

throw new Exception("Ширина не может быть отрицательной");

}

}

}

public Rectangle(double Length = 0, double Width = 0)

{

this.Length = Length;

this.Width = Width;

}

public override double Area()

{

return this.Length \* this.Width;

}

public override string ToString()

{

return $"Прямоугольник со сторонами {this.Length} и {this.Width} и площадью {this.Area():G3}";

}

}

public class Square : Rectangle

{

public Square(double side = 0) : base(side, side) {; }

public override double Length

{

get { return \_Length; }

set

{

if (value >= 0)

{

\_Width = value;

\_Length = value;

}

else

{

throw new Exception("Сторона не может быть отрицательной");

}

}

}

public override double Width

{

get { return \_Length; }

set

{

if (value >= 0)

{

\_Width = value;

\_Length = value;

}

else

{

throw new Exception("Сторона не может быть отрицательной");

}

}

}

public override string ToString()

{

return $"Квадрат со стороной {this.Length} и площадью {this.Area():G3}";

}

}

public class Circle : Figure

{

private double \_Radius;

public double Radius

{

get { return \_Radius; }

set

{

if (value >= 0)

{

\_Radius = value;

}

else

{

throw new Exception("Радиус не может быть отрицательным");

}

}

}

public Circle(double Radius = 0)

{

this.Radius = Radius;

}

public override double Area()

{

return PI \* Pow(Radius, 2);

}

public override string ToString()

{

return $"Круг с радиусом {this.Radius} и площадью {this.Area():G3}";

}

}

}

Matrix.cs

using System;

using System.Text;

using System.Collections.Generic;

using FigureProject;

namespace MatrixProject

{

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;

/// <summary>

/// Количество элементов по оси Z

/// </summary>

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.maxZ)

{

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();

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

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

{

result.Append(k);

result.Append("-й слой матрицы:\n");

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

{

result.Append('[');

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

{

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

{

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

}

else

{

result.Append(" - ");

}

if (j < maxY - 1)

{

result.Append("; ");

}

}

result.Append("]\n");

}

result.Append('\n');

}

return result.ToString();

}

}

/// <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);

}

public class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public Figure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(Figure element)

{

return element == null;

//bool Result = false;

//if (element == null)

//{

// Result = true;

//}

//return Result;

}

}

}

SimpleStack.cs

using System;

using System.Collections.Generic;

namespace SimpleListProject

{

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;

}

}

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

/// <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;

//Присоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Удаление элемента

/// </summary>

public void Remove(int index)

{

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

{

//Можно создать собственный класс исключения

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

}

//Удаление элемента из начала списка

if (index == 0)

{

this.first = this.first.next;

}

else

{

SimpleListItem<T> current = this.first;

for (int i = 0; i < index - 1; ++i)

{

current = current.next;

}

current.next = current.next.next;

//Меняем last если удаляемый элемент был последним

if (index == this.Count - 1)

{

last = current;

}

}

--this.Count;

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int index)

{

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

{

//Можно создать собственный класс исключения

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

}

SimpleListItem<T> current = this.first;

for (int i = 0; i < index; ++i)

{

current = current.next;

}

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;

}

}

/// <summary>

/// Стэк

/// </summary>

public class SimpleStack<T>: SimpleList<T>

where T:IComparable

{

public void Push (T element)

{

base.Add(element);

}

public T Pop()

{

T result = base.GetItem(Count - 1).data;

base.Remove(Count - 1);

return result;

}

}

}

4. Результаты

