**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 ИУ5-32 |  | преподаватель каф. ИУ5 |
| Барышников Михаил |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Задача**

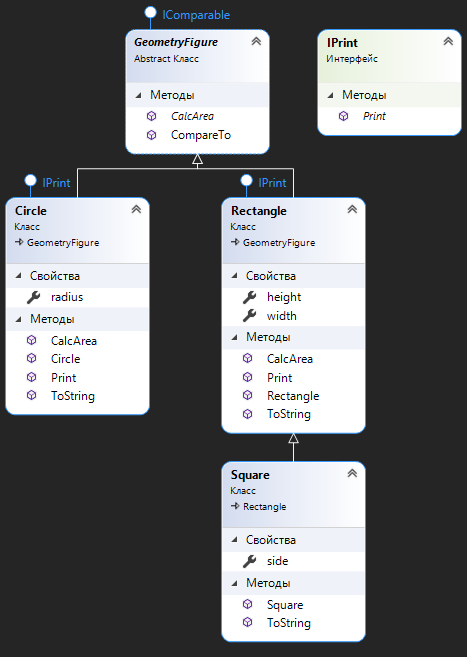
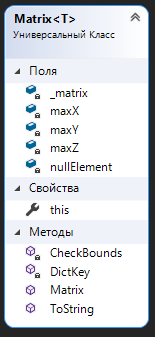
Разработать консольную программу реализующую работу с классами. Взять за основу предыдущую лабораторную работу с классом «Геометрическая фигура». Добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции. 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. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:

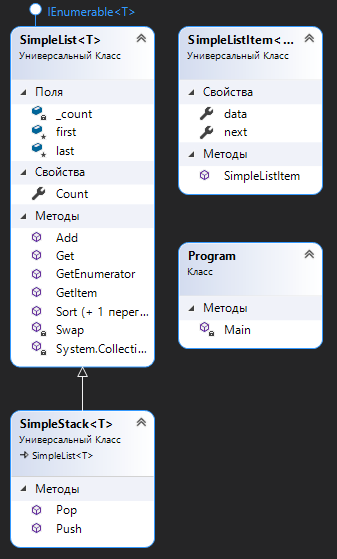
• public void Push(T element) – добавление в стек;

• public T Pop() – чтение с удалением из стека.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Диаграмма классов**



**Текст программы**

* GeometryFigures.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Collections

{

interface IPrint

{

void Print();

}

abstract class GeometryFigure : IComparable

{

public abstract double CalcArea();

public int CompareTo(object obj)

{

GeometryFigure next = obj as GeometryFigure;

if (CalcArea() < next.CalcArea())

return -1;

if (CalcArea() > next.CalcArea())

return 1;

return 0;

}

}

class Rectangle : GeometryFigure, IPrint

{

public double height { set; get; }

public double width { set; get; }

public Rectangle(double \_height, double \_width)

{

height = \_height;

width = \_width;

}

public override double CalcArea()

{

return height \* width;

}

public override string ToString()

{

return "Высота: " + height.ToString() + "; ширина: " + width.ToString() + "; площадь: " + CalcArea().ToString();

}

public void Print()

{

Console.WriteLine(ToString());

}

}

class Square : Rectangle

{

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

public double side

{

get

{

return height;

}

set

{

height = value;

width = value;

}

}

public override string ToString()

{

return "Сторона: " + height.ToString() + "; площадь: " + CalcArea().ToString();

}

}

class Circle : GeometryFigure, IPrint

{

public double radius { set; get; }

public Circle(double \_radius)

{

radius = \_radius;

}

public override double CalcArea()

{

return Math.PI \* radius \* radius;

}

public override string ToString()

{

return "Радиус: " + radius.ToString() + "; площадь: " + CalcArea().ToString();

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

* Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace SparseMatrix

{

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 третьей оси (максимальное количество строк)

/// </summary>

int maxZ;

/// <summary>

/// Пустой элемент, который возвращается если элемент с нужными координатами не был задан

/// </summary>

T nullElement;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int px, int py, int pz, T nullElementParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.nullElement = nullElementParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y, int z]

{

get

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

set

{

CheckBounds(x, y, z);

string key = DictKey(x, y, z);

this.\_matrix.Add(key, value);

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= this.maxZ) throw new Exception("y=" + 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 j = 0; j < this.maxY; j++)

{

b.Append("[");

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

{

b.Append("[");

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

{

if (k != 0) b.Append("\t");

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

}

b.Append("]");

}

b.Append("]\n");

}

return b.ToString();

}

}

}

* SimpleList.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace SimpleListProject

{

/// <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;

}

}

/// <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>

/// <param name="element"></param>

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 обощенного 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>

/// <param name="low"></param>

/// <param name="high"></param>

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;

}

}

}

* SimpleStack.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Collections

{

class SimpleStack<T> : SimpleListProject.SimpleList<T>

where T : IComparable

{

public void Push(T element)

{

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

if (last == null)

Add(element);

else

{

Count++;

newItem.next = first;

first = newItem;

}

}

public T Pop()

{

if (first == null) throw new Exception("Out of range!");

T tmp = first.data;

first = first.next;

return tmp;

}

}

}

* Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Security.Cryptography;

namespace Collections

{

class Program

{

static void Main(string[] args)

{

Random a = new Random();

Console.WriteLine("ArrayList заполненый Square с стороной выбранной случайно");

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

ArrayList figureArray = new ArrayList();

for (int i = 0; i != 20; ++i)

figureArray.Add(new Square(a.Next(10, 500)));

figureArray.Sort();

for (int i = 0; i != 20; ++i)

Console.WriteLine(figureArray[i].ToString());

Console.WriteLine("\nList заполненый Square с стороной выбранной случайно");

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

List<GeometryFigure> figureList = new List<GeometryFigure>();

for (int i = 0; i != 20; ++i)

figureList.Add(new Square(a.Next(100, 200)));

figureList.Sort();

for (int i = 0; i != 20; ++i)

Console.WriteLine(figureList[i].ToString());

Console.WriteLine("\nMatrix заполненая");

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

SparseMatrix.Matrix<int> x = new SparseMatrix.Matrix<int>(5, 5, 5, 9);

Console.WriteLine(x.ToString());

Console.WriteLine("\nSimpleStack заполненый Square с стороной выбранной случайно");

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

SimpleStack<GeometryFigure> figureSimpleStack = new SimpleStack<GeometryFigure>();

for (int i = 0; i != 20; ++i)

figureSimpleStack.Push(new Square(a.Next(700, 10000)));

figureSimpleStack.Sort();

for (int i = 0; i != 20; ++i)

Console.WriteLine(figureSimpleStack.Pop().ToString());

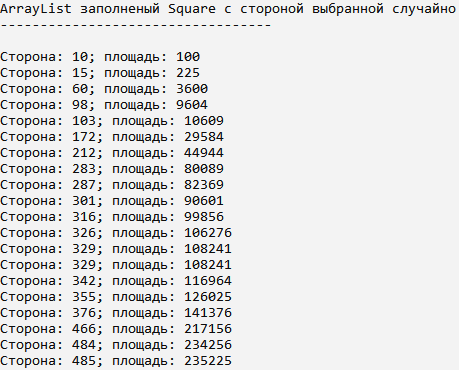
}

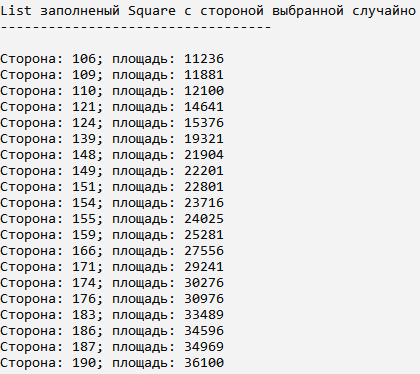
}

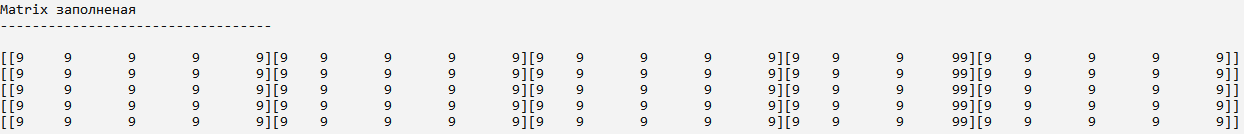
}

**Результаты выполнения программы**

Для теста были созданы следующие объекты, как можно заметить все значения отсортированы по возрастанию:







Вывод осуществлен с помощью созданного метода Pop():

