**Министерство науки и высшего образования Российской** **Федерации** **Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Московский государственный технический университет** **имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Радиотехнический»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Лабораторная работа №3

по дисциплине «Базовые компоненты интернет-технологий» на тему:

«Коллекции в C#»

Выполнил:

студент группы РТ5-31Б

В. И. Плешаков

Проверил:

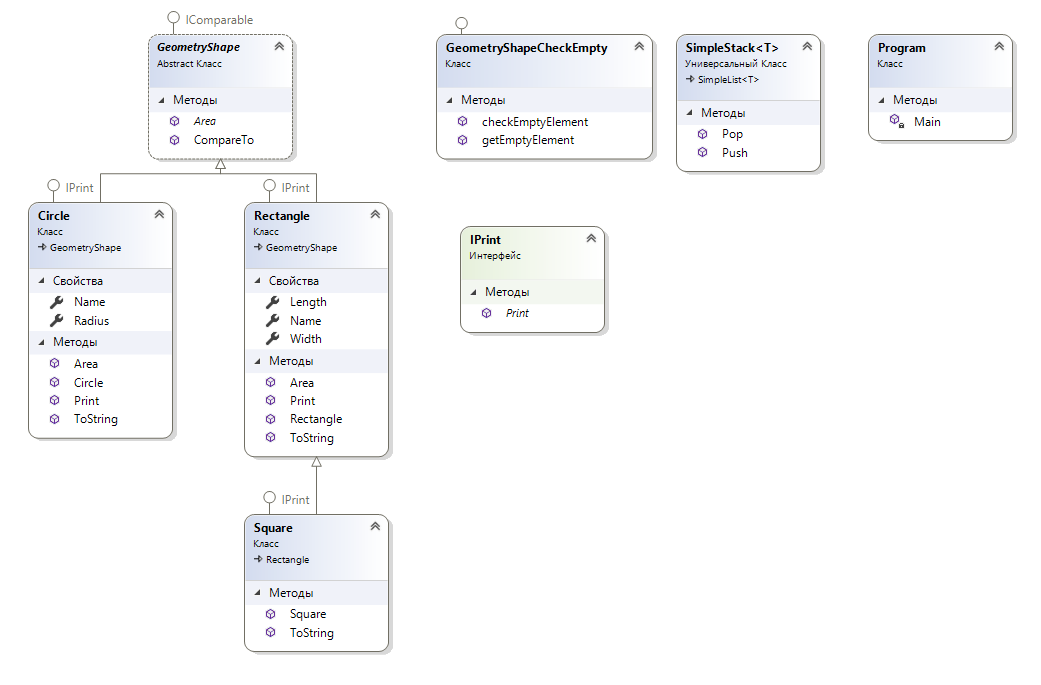
Ю. Е. Гапанюк

2020 г.

Задание:

Разработать программу, реализующую работу с коллекциями

Диаграмма классов:



Текст программы:

Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using SparseMatrix;

using SimpleListProject;

namespace GeometryShape

{

public abstract class GeometryShape : IComparable

{

public abstract double Area();

public int CompareTo(object obj)

{

GeometryShape g = (GeometryShape)obj;

if (this.Area() < g.Area()) return -1;

else if (this.Area() == g.Area()) return 0;

else return 1;

}

}

interface IPrint

{

void Print();

}

class Rectangle : GeometryShape, IPrint

{

public Rectangle(double a, double b)

{

Length = a;

Width = b;

Name = "Прямоугольник";

}

public string Name { get; set; }

public double Length { get; set; }

public double Width { get; set; }

public override double Area()

{

return Length \* Width;

}

public override string ToString()

{

return "Фигура: " + Name + ", Длина = " + Length.ToString("F2") + ", Ширина = " + Width.ToString("F2") + ", Площадь = " + Area().ToString("F2");

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

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

{

Name = "Квадрат";

}

public override string ToString()

{

return "Фигура: " + Name + ", Длина = " + Length.ToString("F2") + ", Площадь = " + Area().ToString("F2");

}

}

class Circle : GeometryShape, IPrint

{

public Circle(double r)

{

Name = "Окружность";

Radius = r;

}

public string Name { get; set; }

public double Radius { get; set; }

public override double Area()

{

return Math.PI \* Radius \* Radius;

}

public override string ToString()

{

return "Фигура: " + Name + ", Радиус = " + Radius.ToString("F2") + ", Площадь = " + Area().ToString("F2");

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

/// <summary>

/// Обработчик пустого элемента матрицы для GeometryShape

/// </summary>

public class GeometryShapeCheckEmpty : IMatrixCheckEmpty<GeometryShape>

{

public bool checkEmptyElement(GeometryShape element)

{

bool result = false;

if (element == null) result = true;

return result;

}

public GeometryShape getEmptyElement()

{

return null;

}

}

/// <summary>

/// Класс стека

/// </summary>

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

{

public void Push(T element)

{

Add(element);

}

/// <summary>

/// Чтение с удалением из стека

/// </summary>

public T Pop()

{

//default(T) - значение для типа T по умолчанию

T Result = default(T);

//Если стек пуст, возвращается значение по умолчанию для типа

if (this.Count == 0) return Result;

//Если элемент единственный

if (this.Count == 1)

{

Result = this.first.data; //обнуляются указатели начала и конца списка

this.first = null;

this.last = null;

}

else

{

//Поиск предпоследнего элемента

SimpleListItem<T> newLast = this.GetItem(this.Count - 2);

//Чтение значения из последнего элемента

Result = newLast.next.data;

//предпоследний элемент считается последним

this.last = newLast;

//последний элемент удаляется из списка

newLast.next = null;

}

//Уменьшение количества элементов в списке

this.Count--;

//Возврат результата

return Result;

}

}

class Program

{

static void Main(string[] args)

{

Rectangle rec = new Rectangle(3, 4);

Square sq = new Square(3);

Circle cr = new Circle(3);

//Необобщенный список

Console.WriteLine("Необобщенный список");

ArrayList al = new ArrayList

{

rec, sq, cr

};

Console.WriteLine("До сортировки");

foreach (object g in al)

Console.WriteLine(g.ToString());

al.Sort();

Console.WriteLine("После сортировки");

foreach (object g in al)

Console.WriteLine(g.ToString());

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

//Обобщенный список

Console.WriteLine("Обобщенный список");

List<GeometryShape> l = new List<GeometryShape>

{

cr, sq, rec

};

Console.WriteLine("До сортировки");

foreach (GeometryShape g in l)

Console.WriteLine(g.ToString());

l.Sort();

Console.WriteLine("После сортировки");

foreach (GeometryShape g in l)

Console.WriteLine(g.ToString());

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

//Разреженная матрица

Console.WriteLine("Разреженная матрица");

SparseMatrix<GeometryShape> matrix = new SparseMatrix<GeometryShape>(3, 3, 3, new GeometryShapeCheckEmpty());

matrix[1, 0, 0] = cr;

matrix[0, 2, 0] = rec;

matrix[0, 0, 3] = sq;

Console.WriteLine(matrix.ToString());

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

//Стек

Console.WriteLine("Стек");

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

//добавление данных в стек

stack.Push(rec);

stack.Push(sq);

stack.Push(cr);

stack.Sort();

//чтение данных из стека

while (stack.Count > 0)

{

GeometryShape f = stack.Pop();

Console.WriteLine(f);

}

}

}

}

SparseMatrix.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace SparseMatrix

{

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент пустой

/// </summary>

bool checkEmptyElement(T element);

}

public class SparseMatrix<T>

{

/// <summary>

/// Словарь для хранения данных матрицы

/// </summary>

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

//Максимальное количество элементов

int maxX;

int maxY;

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> checkEmpty;

/// <summary>

/// Конструктор

/// </summary>

public SparseMatrix(int mx, int my, int mz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = mx;

this.maxY = my;

this.maxZ = mz;

this.checkEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данным

/// </summary>

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

{

set

{

CheckBounds(x, y, z);

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

\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y, z);

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

if (this.\_matrix.ContainsKey(key)) return \_matrix[key];

else return this.checkEmpty.getEmptyElement();

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

if (x < 0 || x > maxX)

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

if (y < 0 || y > maxY)

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

if (z < 0 || z > maxZ)

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

}

/// <summary>

/// Формирование ключа

/// </summary>

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

{

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

}

/// <summary>

/// Приведение к строке

/// </summary>

public override string ToString()

{

StringBuilder str = new StringBuilder();

str.Append("x\ty\tz\tvalue\n");

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

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

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

{

string key = CreateKey(i, j, k);

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

str.Append(i.ToString() + "\t" + j.ToString() + "\t" + k.ToString() + "\t" + this[i, j, k].ToString() + "\n");

}

return str.ToString();

}

}

class SparseMatrix

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

SimpleList.cs

using System;

using System.Collections;

using System.Collections.Generic;

namespace SimpleListProject

{

/// <summary>

/// Класс для элемента списка

/// </summary>

public class SimpleListItem<T>

{

/// <summary>

/// Данные

/// </summary>

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

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

/// <summary>

/// Конструктор

/// </summary>

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

public SimpleListItem(T param)

{

this.data = param;

}

}

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

/// <summary>

/// Первый элемент

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count { get; protected set; }

/// <summary>

/// Добавление нового 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 последующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Присоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Чтение контейнера по номеру

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

int i = 0;

SimpleListItem<T> current = this.first;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

/// <summary>

/// Чтение данных из контейнера по номеру

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Перебор коллекции

/// </summary>

/// <returns></returns>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Сортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Быстрая сортировка

/// </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>

/// Метод для перестановки значений

/// </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;

}

}

class SimpleList

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

Пример выполнения программы:

