**Отчет по лабораторной работе № 3 по курсу С#**

11

(количество листов)

Студент группы ИУ5-32:

Платонов Артемий

Подпись Дата 06.12.2017

Руководитель: Гапанюк Ю.Е

Подпись

Дата

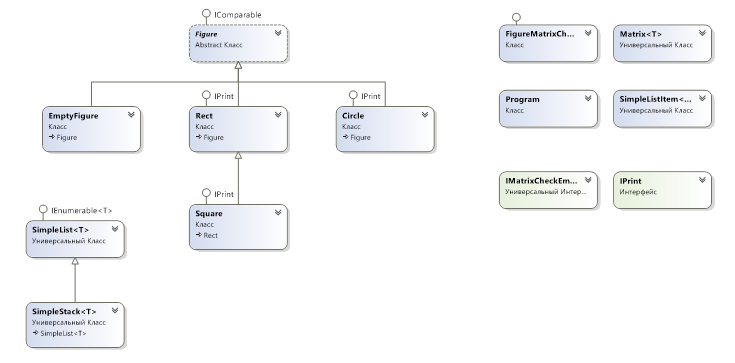
Москва, 2017

***Задание:***

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 (разобранного в пособии). 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.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections;

using System.Collections.Generic;

namespace Lab2

{

abstract class Figure: IComparable

{

public virtual double CalcArea() { return 0; }

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public override string ToString()

{

return this.Type + " площадью " + this.CalcArea().ToString();

}

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

if (this.CalcArea() < p.CalcArea()) return -1;

else if (this.CalcArea() == p.CalcArea()) return 0;

else return 1;

}

}

class Rect : Figure, IPrint

{

private double \_Width;

public double Width

{

get

{

return \_Width;

}

set

{

\_Width = value;

}

}

private double \_Height;

public double Height

{

get

{

return \_Height;

}

set

{

\_Height = value;

}

}

public override double CalcArea()

{

return Width \* Height;

}

public Rect(double w, double h)

{

Width = w;

Height = h;

this.Type = "Прямоугольник";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rect, IPrint

{

public Square(double a) : base(a, a) { this.Type = "Квадрат"; }

}

class Circle : Figure, IPrint

{

private double \_Radius;

public double Radius

{

get

{

return \_Radius;

}

set

{

\_Radius = value;

}

}

public Circle(double r)

{

this.Radius = r;

this.Type = "Круг";

}

public override double CalcArea()

{

return Radius \* Radius \* Math.PI;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

interface IPrint

{

void Print();

}

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> сheckEmpty;

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

{

maxX = px;

maxY = py;

maxZ = pz;

сheckEmpty = сheckEmptyParam;

}

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

{

set

{

CheckBounds(x,y,z);

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

\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y,z);

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

if (\_matrix.ContainsKey(key))

{

return \_matrix[key];

}

else

{

return сheckEmpty.getEmptyElement();

}

}

}

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");

}

}

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

{

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

}

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

b.Append("z=" + k+"\n");

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

{

b.Append("[");

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

{

if (i > 0)

{

b.Append("\t");

}

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

{

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

}

else

{

b.Append(" - ");

}

}

b.Append("]\n");

}

}

return b.ToString();

}

}

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

class EmptyFigure : Figure

{

public override double CalcArea()

{

return 0;

}

}

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

protected SimpleListItem<T> first = null;

protected SimpleListItem<T> last = null;

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

public void Add(T element)

{

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)

{

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

{

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

}

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;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

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

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, 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);

}

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;

}

}

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

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

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

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;

}

}

class Program

{

static void Main()

{

Rect rect = new Rect(5, 4);

Square square = new Square(5);

Circle circle = new Circle(5);

Console.WriteLine("\nArrayList");

ArrayList al = new ArrayList();

al.Add(circle);

al.Add(rect);

al.Add(square);

foreach (var x in al) Console.WriteLine(x);

Console.WriteLine("\nArrayList - сортировка");

al.Sort();

foreach (var x in al) Console.WriteLine(x);

Console.WriteLine("\nList<Figure>");

List<Figure> fl = new List<Figure>();

fl.Add(circle);

fl.Add(rect);

fl.Add(square);

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in fl) Console.WriteLine(x);

fl.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in fl) Console.WriteLine(x);

Console.WriteLine("\nМатрица");

Matrix<Figure> matrix = new Matrix<Figure>(3, 3, 3, new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = square;

matrix[2, 2, 2] = circle;

Console.WriteLine(matrix.ToString());

try

{

Figure temp = matrix[123, 123, 123];

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

Console.WriteLine("\nСписок");

SimpleList<Figure> list = new SimpleList<Figure>();

list.Add(circle);

list.Add(rect);

list.Add(square);

Console.WriteLine("\nПеред сортировкой:");

foreach (var x in list) Console.WriteLine(x);

list.Sort();

Console.WriteLine("\nПосле сортировки:");

foreach (var x in list) Console.WriteLine(x);

Console.WriteLine("\nСтек");

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

