**Лабораторная работа № 3 по курсу**

**“Базовые компоненты интернет-технологий”**

*Наврузов Эмир Русланович*

*РТ5-31*

*МГТУ им. Баумана*

**Описание задания лабораторной работы.**

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.

**Код программы:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_laba

{

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;

}

}

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> сheckEmpty;

public Matrix(int x, int y, int z, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = x;

this.maxY = y;

this.maxZ = z;

this.сheckEmpty = сheckEmptyParam;

}

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();

}

}

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

{

if (x < 0 || x >= this.maxX)

{

throw new ArgumentOutOfRangeException("x", "x=" + x + " out of bounds");

}

if (y < 0 || y >= this.maxY)

{

throw new ArgumentOutOfRangeException("y", "y=" + y + " out of bounds");

}

if (z < 0 || z >= this.maxZ)

{

throw new ArgumentOutOfRangeException("z", "z=" + z + " out of bounds");

}

}

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 i = 0; i < this.maxX; i++)

{

b.Append("[");

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

{

b.Append("(");

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

{

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

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

else

b.Append("");

}

if (j < this.maxY - 1)

b.Append("), ");

else

b.Append(")");

}

b.Append("]\n");

}

return b.ToString();

}

}

public class SimpleListItem<T>

{

public T data

{

get;

set;

}

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

public SimpleListItem(T param)

{

this.data = param;

}

}

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("out of index's bound");

}

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, this.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 (high > j) 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;

}

}

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 = null;

}

this.Count--;

return Result;

}

}

#region materials from lab2

interface IPrint

{

void IPrint();

}

/// <summary>

/// Класс Фигура

/// </summary>

abstract class Figure : IComparable

{

protected string FigureType { get; set; }

public abstract double Area();

public override string ToString()

{

return (this.FigureType + " has area " + this.Area().ToString());

}

public int CompareTo(object obj)

{

Figure ofigure = (Figure)obj;

if (this.Area() < ofigure.Area())

return -1;

else if (this.Area() == ofigure.Area())

return 0;

else return 1;

}

}

#region Figures

/// <summary>

/// Класс Прямоугольник

/// </summary>

class Rectangle : Figure, IPrint

{

protected double Width { get; set; }

protected double Length { get; set; }

public Rectangle(double width, double length)

{

this.FigureType = "Rectangle";

this.Width = width;

this.Length = length;

}

public override double Area()

{

return Math.Round((this.Width \* this.Length), 2);

}

public void IPrint()

{

Console.WriteLine(this.ToString());

}

}

/// <summary>

/// Класс Квадрат

/// </summary>

class Square : Rectangle, IPrint

{

public Square(double length)

: base(length, length)

{

this.FigureType = "Square";

this.Length = length;

}

public override double Area()

{

return Math.Round((this.Length \* this.Length), 2);

}

}

/// <summary>

/// Класс Круг

/// </summary>

class Cirle : Figure, IPrint

{

private double Radius { get; set; }

public Cirle(double radius)

{

this.FigureType = "Circle";

this.Radius = radius;

}

public override double Area()

{

return Math.Round(Math.PI \* this.Radius \* this.Radius, 2);

}

public void IPrint()

{

Console.WriteLine(this.ToString());

}

}

#endregion

#endregion

class Program

{

static void Main(string[] args)

{

Rectangle obj\_rect = new Rectangle(4, 6);

Square obj\_square = new Square(4);

Cirle obj\_circle = new Cirle(3.14);

Console.WriteLine("\tIPRINT:");

obj\_rect.IPrint();

obj\_square.IPrint();

obj\_circle.IPrint();

List<Figure> list\_f = new List<Figure>();

list\_f.Add(obj\_rect);

list\_f.Add(obj\_square);

list\_f.Add(obj\_circle);

Console.WriteLine("\tNOT SORTED LIST: ");

foreach (var obj in list\_f)

Console.WriteLine(obj);

list\_f.Sort();

Console.WriteLine("\tSORTED LIST:");

foreach (var obj in list\_f)

Console.WriteLine(obj);

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

stack\_f.Add(obj\_rect);

stack\_f.Add(obj\_square);

stack\_f.Add(obj\_circle);

Console.WriteLine("\tSTACK");

while (stack\_f.Count > 0)

{

var a = stack\_f.Pop();

Console.WriteLine(a);

}

Console.WriteLine("\tMATRIX:");

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

matrix\_f[0, 0, 0] = obj\_rect;

matrix\_f[1, 1, 1] = obj\_square;

matrix\_f[2, 2, 2] = obj\_circle;

Console.WriteLine(matrix\_f.ToString());

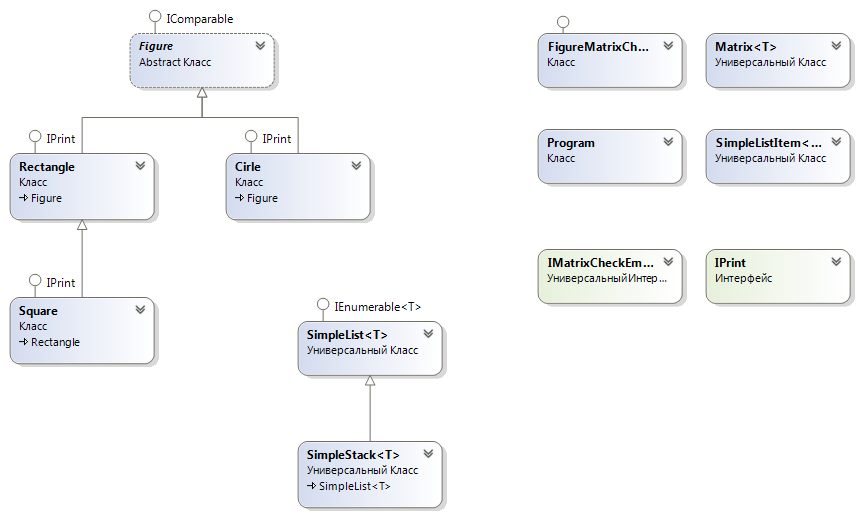
Console.Write("\n\nEnter any key for continue...");

Console.ReadKey();

}

}

}

**Диаграмма классов:** ****

**Пример консольного вывода:**

