**Отчет по Лабораторной работе №3**

Студент группы ИУ5-34

Тимаков Анатолий

Дата: 27.12.17

**Формулировка задачи:**

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 Matrix (представлен в разделе «Вспомогательные материалы для выполнения лабораторных работ») 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 (представлен в разделе 9 «Вспомогательные материалы для выполнения лабораторных работ»). 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.Collections;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3

{

public class Matrix<T>

{

Dictionary<string, T> \_matrix = new Dictionary<string, T>(); // Словарь для хранения значений

int maxX; //max столбцов

int maxY; //max строк

int maxZ;

string name;

T nullElement; //Пустой элемент

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.name = name;

this.nullElement = nullElementParam;

}

void CheckBounds(int x, int y, int z) //Проверка границ

{

if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " out of bounds");

if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " out of bounds");

if (z < 0 || z >= this.maxY) throw new Exception("z=" + z + " out of bounds");

}

string DictKey(int x, int y, int z) //Создание ключа

{

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

}

public T this[int x, int y, int z] //Индикатор для доступа к данным

{

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);

}

}

public override string ToString() //Приведение к строке

{

int num = 0;

StringBuilder b = new StringBuilder();

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

{

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

{

num = 0;

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

{

if (num > 1) b.Append("\t");

b.Append(this.name + "[" + i + "," + j + "," + z + "]: ");

if (this[i, j, z].Equals(nullElement)) b.Append("-" + "\t\t");

else

{

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

num++;

}

}

b.Append("\n");

}

}

return b.ToString();

}

}

abstract class GeomFigure : IComparable

{

public abstract double area();

public int CompareTo(object ob) //with IComparable

{

GeomFigure f = (GeomFigure)ob;

if (this.area() == f.area()) return 0;

else if (this.area() > f.area()) return 1;

else return -1;

}

}

interface IPrint

{

void Print();

}

class Rectangle : GeomFigure, IPrint

{

private double \_width;

public double width

{

get

{

return \_width;

}

set

{

\_width = value;

}

}

public double height { get; set; }

public Rectangle(double a, double b) { \_width = a; height = b; }

public override double area()

{

return \_width \* height;

}

public override string ToString()

{

return "Rectangle \nWidth = " + this.\_width.ToString() + "\nHeight = " + this.height.ToString() + "\nArea = " + area().ToString() + "\n";

}

public void Print()

{

System.Console.WriteLine(ToString());

}

}

class Square : Rectangle, IPrint

{

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

public override string ToString()

{

return "Square \nSide = " + this.width.ToString() + "\nArea = " + area().ToString() + "\n";

}

public void Print()

{

System.Console.WriteLine(ToString());

}

}

class Circle : GeomFigure, IPrint

{

public double rad { get; set; }

public Circle(double r) { this.rad = r; }

public override double area()

{

return Math.PI \* rad \* rad;

}

public override string ToString()

{

return "Circle \nRadius = " + this.rad.ToString() + "\nArea = " + area().ToString() + "\n";

}

public void Print()

{

System.Console.WriteLine(ToString());

}

}

public class SimpleListItem<T> // Элемент списка

{

public T data { get; set; } // Данные

public SimpleListItem<T> next { get; set; } // Следующий элемент

public SimpleListItem(T param) //конструктор

{

this.data = param;

}

}

public class SimpleList<T> : IEnumerable<T> // Список

where T : IComparable

{

protected SimpleListItem<T> first = null; // Первый элемент списка

protected SimpleListItem<T> last = null; // Последний элемент списка

int \_count;

public int Count // Количество элементов

{

get { return \_count; }

protected set { \_count = value; }

}

public void Add(T element) // Добавление элемента

{

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

this.Count++;

if (last == null) //Добавление первого элемента

{

this.first = newItem;

this.last = newItem;

}

else //Добавление следующих элементов

{

this.last.next = newItem; //Присоединение элемента к цепочке

this.last = newItem; //Просоединенный элемент считается последним

}

}

public SimpleListItem<T> GetItem(int number) // Чтение контейнера с заданным номером

{

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

{

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

}

SimpleListItem<T> current = this.first;

int i = 0;

while (i < number) //Пропускаем нужное количество элементов

{

current = current.next;

i++;

}

return current;

}

public T Get(int number) // Чтение элемента с заданным номером

{

return GetItem(number).data;

}

public IEnumerator<T> GetEnumerator() // Для перебора коллекции

{

SimpleListItem<T> current = this.first;

while (current != null) //Перебор элементов

{

yield return current.data; //Возврат текущего значения

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();

}

public void Sort() // Cортировка

{

Sort(0, this.Count - 1);

}

private void Sort(int low, int high) // Реализация алгоритма быстрой сортировки

{

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) // Вспомогательный метод для обмена элементов при сортировке

{

SimpleListItem<T> ci = GetItem(i);

SimpleListItem<T> cj = GetItem(j);

T temp = ci.data;

ci.data = cj.data;

cj.data = temp;

}

}

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

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

T res = default(T);

if (this.Count == 0)

return res;

else if (this.Count == 1)

{

res = this.first.data;

this.first = null;

this.last = null;

}

else

{

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

res = newLast.next.data;

this.last = newLast;

newLast.next = null;

}

this.Count--;

return res;

}

}

class Program

{

static void Main(string[] args)

{

double a = 1, b = 2, r = 3;

Rectangle rec = new Rectangle(a, b);

Square sq = new Square(a);

Circle circ = new Circle(r);

ArrayList al = new ArrayList();

al.Add(rec);

al.Add(sq);

al.Add(circ);

al.Sort();

Console.WriteLine("Список");

foreach (object obj in al) Console.WriteLine(obj.ToString());

Rectangle rec2 = new Rectangle(4, 5);

Square sq2 = new Square(4);

Circle circ2 = new Circle(1);

List<GeomFigure> list = new List<GeomFigure>();

list.Add(rec2);

list.Add(sq2);

list.Add(circ2);

list.Sort();

Console.WriteLine("Список2");

Console.WriteLine("\nСписок с " + list.Count() + " элементами");

foreach (GeomFigure figure in list)

{

Console.WriteLine(figure.ToString());

}

Console.WriteLine("\nМатрица");

Square nullFigure = new Square(0);

Matrix<GeomFigure> matr = new Matrix<GeomFigure>(3, 3, 3, "matr", nullFigure);

matr[0, 0, 0] = rec;

matr[1, 1, 1] = sq;

matr[2, 2, 2] = circ;

Console.WriteLine(matr.ToString());

Console.WriteLine("\nСписок");

SimpleList<GeomFigure> Ex1 = new SimpleList<GeomFigure>();

Ex1.Add(rec);

Ex1.Add(circ);

Ex1.Add(sq);

foreach (var x in Ex1) Console.WriteLine(x);

Console.WriteLine("\nСтек");

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

stack.Push(rec);

stack.Push(sq);

stack.Push(sq);

stack.Push(circ);

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

Console.WriteLine("\nРезультат чтения стека: " + stack.Pop().ToString());

Console.WriteLine("\nСтек после чтения");

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

}

}

}

**Результаты работы**









