#### Московский государственный технический университет имени Н.Э. Баумана

## Лабораторная работа по дисциплине БКИТ №3

Выполнил: Байбарин Р.Г., ИУ5-34, 18.10.2017

### I. Описание задания

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 (проект SimpleListProject). 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.

### II. Код программы

﻿using System;  
using System.Collections;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
//Program.cs  
namespace Lab\_3  
{  
 class Program  
 {  
   
 static void Main()  
 {  
 Menu m =new Menu();  
 m.run();  
 }  
 }  
}

//Lab\_2\_Source.cs  
﻿using System;  
  
namespace Lab\_3  
{  
 interface IPrint  
 {  
 void print();  
 }  
 abstract class Figure : IComparable, IPrint  
 {  
 virtual public double square() { return 0; }  
 public int CompareTo(object obj)  
 {  
 Figure f1 = obj as Figure;  
 if (f1 == null)  
 throw new ArgumentException("obj is not Figure!!!");  
 //if (this.square() > f1.square()) return 1;  
 //else if (this.square() == f1.square()) return 0;  
 //else return -1;  
 return this.square().CompareTo(f1.square());  
 }  
 abstract public void print();  
  
 }  
 class Circle : Figure  
 {  
 private double r;  
 public double radius  
 {  
 get  
 {  
 return r;  
 }  
 set  
 {  
 if (value < 0)  
 throw new ArgumentOutOfRangeException($"{nameof(value)} должно быть положительным!");  
 r = value;  
 }  
 }  
 public Circle(double r1)  
 {  
 radius = r1;  
 }  
 public override string ToString()  
 {  
 string res;  
 res = "Class: Circle," +  
 $" Area: {Math.Round(square(), 3)}";  
 return res;  
 }  
 public override double square()  
 {  
 return Math.PI \* r \* r;  
 }  
 public override void print()  
 {  
 Console.WriteLine(this.ToString());  
 }  
  
 }  
 class Rect : Figure  
 {  
 private double a;  
 private double b;  
 virtual public double A  
 {  
 get  
 {  
 return a;  
 }  
 set  
 {  
 if (value < 0)  
 throw new ArgumentOutOfRangeException($"{nameof(value)} должно быть положительным!");  
 a = value;  
 }  
 }  
 virtual public double B  
 {  
 get  
 {  
 return b;  
 }  
 set  
 {  
 if (value < 0)  
 throw new ArgumentOutOfRangeException($"{nameof(value)} должно быть положительным!");  
 b = value;  
 }  
 }  
 public Rect(double a1, double b1)  
 {  
 A = a1;  
 B = b1;  
 }  
 public override string ToString()  
 {  
 string res;  
 res = "Class: Rectangle," +  
 $" Area: {Math.Round(square(),3)}";  
 return res;  
 }  
 public override double square()  
 {  
 return A \* B;  
 }  
 public override void print()  
 {  
 Console.WriteLine(this.ToString());  
 }  
 }  
 class Square : Rect  
 {  
 private double a;  
 //public override double A  
 //{  
 // get  
 // {  
 // return a;  
 // }  
 // set  
 // {  
 // if (value < 0)  
 // throw new ArgumentOutOfRangeException($"{nameof(value)} должно быть положительным!");  
 // a = value;  
 // }  
 //}  
 public override string ToString()  
 {  
 string res;  
 res = "Class: Square," +  
 $" Area: {Math.Round(square(), 3)}";  
 return res;  
 }  
 //public override double square()  
 //{  
 // return A \* A;  
 //}  
 public Square(double a1) : base(a1,a1)  
 {  
 }  
 }  
}

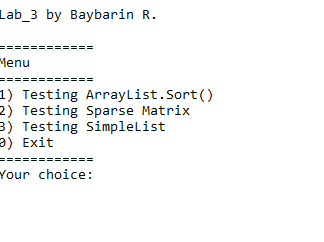
//Lab\_3\_Source.cs  
﻿using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Lab\_3  
{  
  
 class TestArray : IPrint  
 {  
 ArrayList arr1;  
 Random rand;  
 public TestArray()  
 {  
 arr1 = new ArrayList();  
 rand = new Random();  
 }  
 public void Gener(int rmax, int times)  
 {  
 for (int i = 0; i < times; i++)  
 {  
 Figure res = null;  
 int a = rand.Next(3);  
 switch (a)  
 {  
 case 0:  
 res = new Circle(rand.Next(rmax));  
 break;  
 case 1:  
 res = new Rect(rand.Next(rmax), rand.Next(rmax));  
 break;  
 case 2:  
 res = new Square(rand.Next(rmax));  
 break;  
 }  
 arr1.Add(res);  
 }  
 }  
 public void print()  
 {  
 foreach (Figure f in arr1)  
 {  
 f.print();  
 }  
 }  
 public void Sort()  
 {  
 arr1.Sort();  
 }  
 }  
 //==================================  
 public interface IMatrAdapter<T>  
 {  
 T getEmptyElement();  
 bool checkEmptyElement(T element);  
 }  
 public class FigureMatrixCheckEmpty : IMatrAdapter<Figure>  
 {  
 Figure IMatrAdapter<Figure>.getEmptyElement()  
 {  
 return null;  
 }  
 bool IMatrAdapter<Figure>.checkEmptyElement(Figure element)  
 {  
 bool res = false;  
 if (element == null)  
 res = true;  
 return res;  
 }  
 }  
 class Matrix<T>:IPrint  
 {  
 int xmax, ymax, zmax;  
 public IMatrAdapter<T> Adap;  
 Dictionary<string, T> \_matr = new Dictionary<string, T>();  
 public Matrix(int \_xmax, int \_ymax, int \_zmax, IMatrAdapter<T> IMCE)  
 {  
 xmax = \_xmax;  
 ymax = \_ymax;  
 zmax = \_zmax;  
 Adap = IMCE;  
 }  
 public T this[int x, int y, int z]  
 {  
 set  
 {  
 CheckBounds(x, y, z);  
 string key = DictKey(x, y, z);  
 if (this.\_matr.ContainsKey(key))  
 {  
 this.\_matr[key] = value;  
 }  
 else  
 {  
 this.\_matr.Add(key, value);  
 }  
 }  
 get  
 {  
 CheckBounds(x, y, z);  
 string key = DictKey(x, y, z);  
 if (this.\_matr.ContainsKey(key))  
 {  
 return this.\_matr[key];  
 }  
 else  
 {  
 return this.Adap.getEmptyElement();  
 }  
 }  
 }  
 void CheckBounds(int x, int y, int z)  
 {  
 if (x < 0 || x >= xmax)  
 throw new ArgumentOutOfRangeException("WRONG x");  
 if (y < 0 || y >= ymax)  
 throw new ArgumentOutOfRangeException("WRONG y");  
 if (z < 0 || z >= zmax)  
 throw new ArgumentOutOfRangeException("WRONG z");  
  
 }  
 string DictKey(int x, int y, int z)  
 {  
 return x.ToString() + "\_" + y.ToString()+"\_"+ z.ToString();  
 }  
 public override string ToString()  
 {  
 StringBuilder res = new StringBuilder();  
 res.Append("x | y | z | value\n");  
 for (int i = 0; i < xmax; i++)  
 {  
 for (int j = 0; j < ymax; j++)  
 {  
 for (int k = 0; k < zmax; k++)  
 {  
 if (!Adap.checkEmptyElement(this[i, j, k]))  
 {  
 res.Append(i.ToString()+" | "+j.ToString() + " | "+k.ToString() + " | "+this[i, j, k].ToString()+"\n");  
 }  
 }  
 }  
 }  
 res.Append("All other elements are empty\n");  
 return res.ToString();  
 }  
 public void print()  
 {  
 Console.WriteLine(this.ToString());  
 }  
 }  
 //==================================  
 class SimpleList<T> : IPrint  
 where T : IComparable  
 {  
 protected SimpleListItem<T> first;  
 protected SimpleListItem<T> last;  
 int \_count;  
 public SimpleList()  
 {  
 first = null;  
 last = null;  
 }  
 public int count  
 {  
 get  
 {  
 return \_count;  
 }  
 protected set  
 {  
 \_count = value;  
 }  
 }  
 public void Add(T element)  
 {  
 SimpleListItem<T> a = new SimpleListItem<T>(element);  
 if (this.last == null)  
 {  
 first = a;  
 last = a;  
 }  
 else  
 {  
 last.next = a;  
 last = a;  
 }  
 count++;  
 }  
 public SimpleListItem<T> getItem(int number)  
 {  
 if (number < 0 || number >= count)  
 {  
 throw new ArgumentException("WRONG INDEX");  
 }  
 else  
 {  
 int i = 0;  
 SimpleListItem<T> buf = this.first;  
 while (i<number)  
 {  
 i++;  
 buf = buf.next;  
 }  
 return buf;  
 }  
 }  
 public T Get(int number)  
 {  
 return getItem(number).data;  
 }  
 public override string ToString()  
 {  
 StringBuilder res = new StringBuilder();  
 for (int i = 0; i < count; i++)  
 {  
 res.Append(i.ToString()+") "+Get(i).ToString()+"\n");  
 }  
 return res.ToString();  
 }  
 public void print()  
 {  
 Console.WriteLine(this.ToString());  
 }  
 }  
 class SimpleListItem<T>  
 where T : IComparable  
 {  
 public SimpleListItem(T param)  
 {  
 data = param;  
 }  
 public T data  
 {  
 get  
 {  
 return \_data;  
 }  
 set  
 {  
 \_data = value;  
 }  
 }  
 T \_data;  
 public SimpleListItem<T> next  
 {  
 get  
 {  
 return \_next;  
 }  
 set  
 {  
 \_next = value;  
 }  
 }  
 SimpleListItem<T> \_next = null;  
 }  
 class SimpleStack<T> : SimpleList<T>  
 where T:IComparable  
 {  
 public SimpleListItem<T> Pop(int i)  
 {  
 SimpleListItem<T> buf = getItem(i);  
 if (first == buf)  
 {  
 first = buf.next;//ошибка, если в списке 1 элемент!  
 if (count == 1)  
 last = null;  
 }  
 else  
 {  
 getItem(i - 1).next = buf.next;  
 }  
 count--;  
 return buf;  
 }  
 public void Push(T element)  
 {  
 Add(element);  
 }  
 }  
 //==================================  
 class Menu  
 {  
 MySystem mySys = new MySystem();  
 int intReadKey()  
 {  
 int res;  
 bool f = int.TryParse(Console.ReadLine(), out res);  
 if (!f)  
 throw new ArgumentException("Wrong input");  
 return res;  
 }  
 void case1()  
 {  
 Console.Clear();  
 Console.Write("Enter number of elements. Array will be filled by randomly generated figures: ");  
 TestArray a = new TestArray();  
 a.Gener(10, intReadKey());  
 mySys.printPalka(20);  
 a.print();  
 mySys.printPalka(20);  
 Console.WriteLine("Sorting...");  
 mySys.printPalka(20);  
 a.Sort();  
 a.print();  
 }  
 void case2()  
 {  
 Console.Clear();  
 Console.Write("Enter maximum size: ");  
 int rmax= intReadKey();  
 Console.Write("Enter number of elements generated: ");  
 int times = intReadKey();  
 mySys.printPalka(20);  
 Random rand = new Random();  
 Matrix<Figure> matr = new Matrix<Figure>(rmax,rmax,rmax,new FigureMatrixCheckEmpty());  
 for (int i = 0; i < times; i++)  
 {  
 int i0 = rand.Next(rmax), i1 = rand.Next(rmax), i2 = rand.Next(rmax);  
 matr[i0, i1, i2] = mySys.GenerFigure(10, rand);  
 }  
 Console.WriteLine("Printing existing elements...");  
 mySys.printPalka(20);  
 matr.print();  
 }  
 void case3()  
 {  
 Console.Clear();  
 Console.Write("Enter number of elements generated: ");  
 int times = intReadKey();  
 SimpleStack<Figure> a = new SimpleStack<Figure>();  
 Random rand = new Random();  
 for (int i = 0; i < times; i++)  
 {  
 a.Push(mySys.GenerFigure(10, rand));  
 }  
 Console.WriteLine("Printing generated data...");  
 mySys.printPalka(20);  
 a.print();  
 mySys.printPalka(20);  
 Console.Write("Enter number of eliminating element: ");  
 bool t = true;  
 SimpleListItem<Figure> f = null;  
 while (t)  
 {  
 times = intReadKey();  
 try  
 {  
 f = a.Pop(times);  
 t = false;  
 }  
 catch (ArgumentException)  
 {  
 Console.Write("Wrong input! Please, enter again: ");  
 continue;  
 }  
 }  
 mySys.printPalka(20);  
 Console.WriteLine("Eliminated element : " + f.data.ToString());  
 mySys.printPalka(20);  
 Console.WriteLine("Printing stack: ");  
 a.print();  
 }  
 public void run()  
 {  
 bool t = true;  
 bool p = true;  
 while (p)  
 {  
 Console.Clear();  
 if (t) Console.WriteLine("Lab\_3 by Baybarin R.\n");  
 t = false;  
 Console.Write("============\n" +  
 "Menu\n" +  
 "============\n" +  
 "1) Testing ArrayList.Sort()\n" +  
 "2) Testing Sparse Matrix\n" +  
 "3) Testing SimpleList\n" +  
 "0) Exit\n" +  
 "============\n" +  
 "Your choice: ");  
 int caseSwitch;  
 bool f;  
 f = int.TryParse(Console.ReadLine(), out caseSwitch);  
 if (!f || caseSwitch < 0 || caseSwitch > 3)  
 {  
 Console.WriteLine("ERROR!!!!! WRONG INPUT");  
 mySys.pause();  
 continue;  
 }  
 switch (caseSwitch)  
 {  
 case 0:  
 p = false;  
 Console.WriteLine("Exiting...Press Enter");  
 break;  
 case 1: case1();  
 break;  
 case 2: case2();  
 break;  
 case 3: case3();  
 break;  
 }  
 mySys.pause();  
 }  
 }  
 }  
}

//MySystem.cs  
﻿using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Lab\_3  
{  
 class MySystem  
 {  
 public void pause()  
 {  
 Console.WriteLine("Input enter...");  
 Console.ReadLine();  
 }  
 public void printPalka(int l)  
 {  
 for (int i = 0; i < l; i++)  
 {  
 Console.Write("=");  
 }  
 Console.Write("\n");  
 }  
 public Figure GenerFigure(int param, Random rand)  
 {  
 Figure res = null;  
 int a = rand.Next(3);  
 switch (a)  
 {  
 case 0:  
 res = new Circle(rand.Next(param));  
 break;  
 case 1:  
 res = new Rect(rand.Next(param), rand.Next(param));  
 break;  
 case 2:  
 res = new Square(rand.Next(param));  
 break;  
 }  
 return res;  
 }  
 }  
}

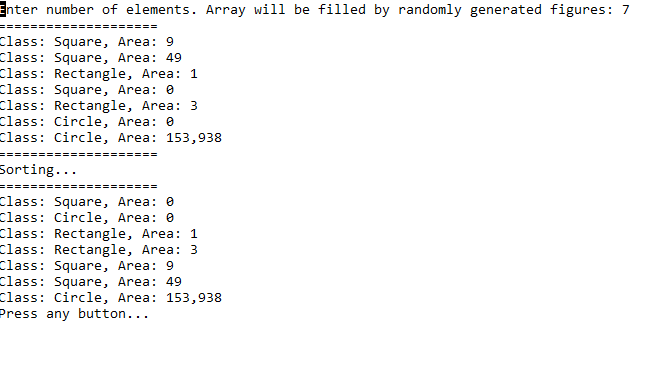
### 

### III. Примеры работы

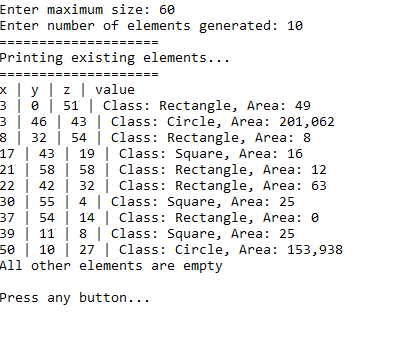
//Начало работы



//Ввели 1



//В главном меню ввели 2



//В главном меню ввели 3

