МГТУ им. Баумана  
Факультет «Информатика и системы управления»  
Кафедра «Автоматизированные системы обработки информации и управления»  
Дисциплина «Базовые компоненты интернет-технологий»

Отчёт по лабораторной работе №3

Выполнил:  
Студент группы ИУ5-33Б  
Волков А.С.  
Преподаватель:  
Гапанюк Ю.

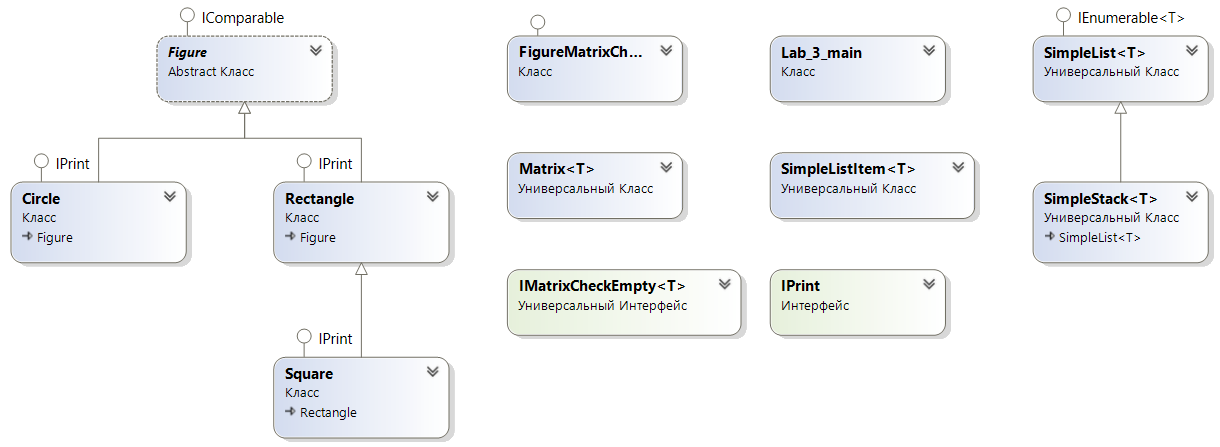
Описание задания:

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 (разобранного в пособии). Необходимо добавить в класс методы:
   * public void Push(T element) – добавление в стек;
   * public T Pop() – чтение с удалением из стека.

Пример работы класса SimpleStack реализовать на основе геометрических фигур.

Диаграмма классов:



Текст программы:

using System;

using System.Text;

using System.Collections;

using System.Collections.Generic;

namespace Lab\_3

{

class Lab\_3\_main

{

static void Main(string[] args)

{

while (true)

{

switch (Menu())

{

case 1:

double ra = ReadDouble("Введите длину прямоугольника: ");

double rb = ReadDouble("Введите ширину прямоугольника: ");

Rectangle rect = new Rectangle(ra, rb);

double sa = ReadDouble("Введите сторону квадрата: ");

Square square = new Square(sa);

double cr = ReadDouble("Введите радиус круга: ");

Circle circle = new Circle(cr);

Console.WriteLine("\n \*\*\* Создание и сортировка ArrayList \*\*\*");

ArrayList arr = new ArrayList { circle, rect, square };

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

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

arr.Sort();

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

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

Console.WriteLine("\n");

Console.WriteLine("\n \*\*\* Создание и сортировка List<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("\n");

Console.WriteLine("\n \*\*\* Создание Matrix \*\*\*");

double ra1 = ReadDouble("Введите длину прямоугольника: ");

double rb1 = ReadDouble("Введите ширину прямоугольника: ");

Rectangle rect1 = new Rectangle(ra1, rb1);

double sa1 = ReadDouble("Введите сторону квадрата: ");

Square square1 = new Square(sa1);

double cr1 = ReadDouble("Введите радиус круга: ");

Circle circle1 = new Circle(cr1);

Matrix<Figure> matr = new Matrix<Figure>(3, 3, 2, new FigureMatrixCheckEmpty());

matr[0, 0, 0] = square;

matr[1, 1, 0] = circle;

matr[2, 2, 0] = rect;

matr[0, 0, 1] = square1;

matr[1, 1, 1] = circle1;

matr[2, 2, 1] = rect1;

Console.WriteLine(matr.ToString());

Console.WriteLine("\n \*\*\* Создание SimpleStack и удаление из стека\*\*\*");

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

stack.Push(square);

stack.Push(circle);

stack.Push(rect);

stack.Push(square1);

stack.Push(circle1);

stack.Push(rect1);

foreach (var s in stack)

{

Console.WriteLine(s.ToString());

}

Figure popped = stack.Pop();

Console.WriteLine("\nИз SimpleStack удалён " + popped.ToString() + "\n");

foreach (var s in stack)

{

Console.WriteLine(s.ToString());

}

Console.WriteLine("\n");

break;

case 0:

return;

}

}

}

static double ReadDouble(string message)

{

string resultString;

double resultDouble;

bool flag;

do

{

Console.Write(message);

resultString = Console.ReadLine();

flag = double.TryParse(resultString, out resultDouble);

if (!flag)

{

Console.WriteLine("Необходимо ввести вещественное число");

}

}

while (!flag);

return resultDouble;

}

static int Menu()

{

int choice;

string resultString;

bool flag;

Console.WriteLine("\*\*\* Меню \*\*\* \n" +

"1. Выполнить \n" +

"0. Выход");

do

{

resultString = Console.ReadLine();

flag = int.TryParse(resultString, out choice);

if (!flag || (choice != 0 & choice != 1))

{

Console.WriteLine("Необходимо ввести целое число, соответствующее пункту меню");

}

}

while (!flag || (choice != 0 & choice != 1));

return choice;

}

}

abstract class Figure : IComparable

{

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public abstract double Area();

public override string ToString()

{

return this.Type + " площадью " +

this.Area().ToString();

}

public int CompareTo(object obj)

{

Figure p = (Figure)obj;

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

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

else return 1; //(this.Area() > p.Area())

}

}

interface IPrint

{

void Print();

}

class Rectangle : Figure, IPrint

{

double height;

double width;

public Rectangle(double ph, double pw)

{

this.height = ph;

this.width = pw;

this.Type = "Прямоугольник";

}

public override double Area()

{

double Result = this.width \* this.height;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

public Square(double size)

: base(size, size)

{

this.Type = "Квадрат";

}

}

class Circle : Figure, IPrint

{

double radius;

public Circle(double pr)

{

this.radius = pr;

this.Type = "Круг";

}

public override double Area()

{

double Result = Math.PI \* this.radius \* this.radius;

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

interface IMatrixCheckEmpty<T>

{

T GetEmptyElement();

bool CheckEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure GetEmptyElement()

{

return null;

}

public bool CheckEmptyElement(Figure element)

{

if (element == null)

{

return true;

}

else

{

return false;

}

}

}

class Matrix<T>

{

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

int maxX, maxY, maxZ;

IMatrixCheckEmpty<T> CheckEmpty;

public Matrix(int xx, int yy, int zz, IMatrixCheckEmpty<T> CheckEmptyParam)

{

maxX = xx;

maxY = yy;

maxZ = zz;

CheckEmpty = CheckEmptyParam;

}

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

{

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

}

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

{

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

{

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

}

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

{

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

}

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

{

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

}

}

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 CheckEmpty.GetEmptyElement();

}

}

}

public override string ToString()

{

StringBuilder tab = new StringBuilder();

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

{

tab.Append("Z = " + k.ToString() + "\n\n");

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

{

tab.Append("[");

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

{

if (i > 0)

{

tab.Append('\t');

}

if (!CheckEmpty.CheckEmptyElement(this[i, j, k]))

{

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

}

else

{

tab.Append(" . ");

}

}

tab.Append("]\n");

}

if (k < maxZ - 1)

{

tab.Append("\n==========================================================================\n\n");

}

}

return tab.ToString();

}

}

public class SimpleListItem<T>

{

public T Data { get; set; }

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

public SimpleListItem(T param)

{

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; protected set; }

public void Add(T elem)

{

SimpleListItem<T> NewItem = new SimpleListItem<T>(elem);

Count++;

if (Last == null)

{

First = NewItem;

Last = NewItem;

}

else

{

Last.Next = NewItem;

Last = NewItem;

}

}

public SimpleListItem<T> GetItem(int index)

{

if ((index < 0) || (index >= Count))

{

throw new Exception("Выход за границы списка.");

}

SimpleListItem<T> Current = First;

int i = 0;

while (i < index)

{

Current = Current.Next;

i++;

}

return Current;

}

public T Get(int index)

{

return GetItem(index).Data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> Current = First;

while (Current != null)

{

yield return Current.Data;

Current = Current.Next;

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, Count - 1);

}

private void Sort(int low, int high)

{

int i = low, j = high;

T x = Get((low + high) / 2);

while (i <= j)

{

while (Get(i).CompareTo(x) < 0) ++i;

while (Get(j).CompareTo(x) > 0) --j;

if (i <= j)

{

Swap(i, j);

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;

}

}

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

{

public void Push(T elem)

{

Add(elem);

}

public T Pop()

{

T Result = default(T);

if (Count == 0)

{

return Result;

}

if (Count == 1)

{

Result = First.Data;

First = null;

Last = null;

}

else

{

SimpleListItem<T> NewLast = GetItem(Count - 2);

Result = NewLast.Next.Data;

Last = NewLast;

NewLast.Next = null;

}

Count--;

return Result;

}

}

}

Экранные формы:

