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

9

(количество листов)

Студент группы ИУ5-34:

Тюлькина Нина

Подпись Дата

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

Руководитель: Гапанюк Ю.Е.

Подпись Дата

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

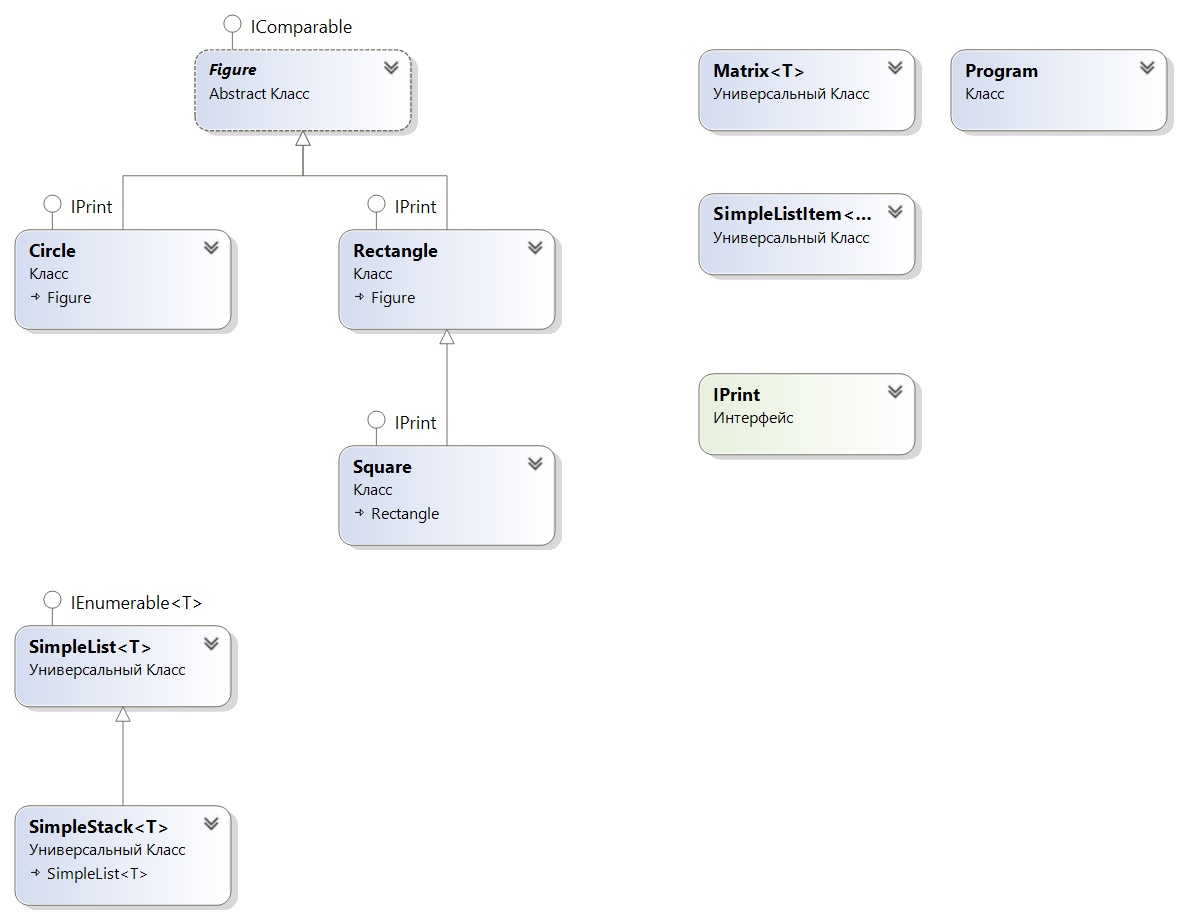
Москва, 2017

*Условие 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 (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.

*Диаграмма классов*



*Текст программы*

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace lab3

{

class Program

{

static void Main(string[] args)

{

///-------------------------------п.4------------------------------------

ArrayList arr = new ArrayList();

FigureCollections.Rectangle rect1 = new FigureCollections.Rectangle(3, 4);

FigureCollections.Circle cir1 = new FigureCollections.Circle(5);

FigureCollections.Square sq1 = new FigureCollections.Square(10);

arr.Add(rect1);

arr.Add(cir1);

arr.Add(sq1);

arr.Sort();

Console.WriteLine("ArrayList was sorted by ArrayList.Sort()");

Console.WriteLine("Results:");

for (int i = 0; i < arr.Capacity - 1; i++) {

Console.WriteLine(arr[i]);

}

Console.WriteLine();

///-------------------------------п.5------------------------------------

FigureCollections.Rectangle rect2 = new FigureCollections.Rectangle(5, 6);

FigureCollections.Circle cir2 = new FigureCollections.Circle(7);

FigureCollections.Square sq2 = new FigureCollections.Square(4);

List<FigureCollections.Figure> list = new List<FigureCollections.Figure>();

list.Add(rect2);

list.Add(sq2);

list.Add(cir2);

list.Sort();

Console.WriteLine("List<Figure> was sorted by List<Figure>.Sort()");

Console.WriteLine("Results:");

for (int i = 0; i < list.Capacity - 1; i++) {

Console.WriteLine(list[i]);

}

///-------------------------------п.6------------------------------------

Console.WriteLine();

Console.WriteLine("Matrix:");

FigureCollections.Circle nullElem = new FigureCollections.Circle(0);

SparseMatrix.Matrix<FigureCollections.Figure> matr = new SparseMatrix.Matrix<FigureCollections.Figure>(3, 3, 3, nullElem);

matr[0, 0, 0] = rect1;

matr[1, 1, 1] = sq1;

matr[2, 2, 2] = cir1;

Console.WriteLine(matr.ToString());

///-------------------------------п.8------------------------------------

Console.WriteLine();

Console.WriteLine("Using SimpleStack:");

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

stack.Add(rect2);

stack.Add(sq2);

stack.Add(cir2);

stack.Add(sq2);

Console.WriteLine();

Console.WriteLine("SimpleStack before Stack.Pop():");

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

stack.Pop();

Console.WriteLine();

Console.WriteLine("SimpleStack after Stack.Pop():");

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

Console.ReadLine();

}

}

}

namespace FigureCollections

{

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

public int Count /// Количество элементов

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

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"); //Можно создать собственный класс исключения

}

SimpleListItem<T> current = this.first;

int i = 0;

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

{

current = current.next; //Переход к следующему элементу

i++; //Увеличение счетчика

}

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; //Переход к следующему элементу

}

}

//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;

}

/// <summary>

/// Класс стек

/// </summary>

}

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

{

public void Push(T element) /// Добавление в стек

{

Add(element); //Добавление в конец списка уже реализовано

}

public T Pop() /// Удаление и чтение из стека

{

T Result = default(T); //default(T) - значение для типа T по умолчанию

if (this.Count == 0) return Result; //Если стек пуст, возвращается значение по умолчанию для типа

if (this.Count == 1) //Если элемент единственный, то из него читаются данные

{

Result = this.first.data;

this.first = null; //обнуляются указатели начала и конца списка

this.last = null;

}

else //В списке более одного элемента

{

SimpleListItem<T> newLast = this.GetItem(this.Count - 2); //Поиск предпоследнего элемента

Result = newLast.next.data; //Чтение значения из последнего элемента

this.last = newLast; //предпоследний элемент считается последним

newLast.next = null; //последний элемент удаляется из списка

}

this.Count--; //Уменьшение количества элементов в списке

return Result; //Возврат результата

}

}

abstract class Figure : IComparable //класс фигура

{

public string Type //тип фигуры

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

public abstract double Area(); //вычисление площади

public override string ToString() // Приведение к строке, переопределение метода Object

{

return this.Type + "'s area = " + this.Area().ToString();

}

public int CompareTo(object obj) // Сравнение элементов (для сортировки списка)

{

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; // Высота

double width; // Ширина

// Основной конструктор

public Rectangle(double ph, double pw) //ph - высота, pw - ширина

{

this.height = ph;

this.width = pw;

this.Type = "Rectangle";

}

public override double Area() // Вычисление площади

{

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

return Result;

}

public void Print() //вывод

{

Console.WriteLine(this.ToString());

}

}

class Square : Rectangle, IPrint

{

public Square(double size)

: base(size, size)

{

this.Type = "Square";

}

}

class Circle : Figure, IPrint

{

double radius; // Ширина

public Circle(double pr) //pr - радиус

{

this.radius = pr;

this.Type = "Circle";

}

public override double Area()

{

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

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

namespace SparseMatrix

{

public class Matrix<T>

{

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

int maxX; /// Количество элементов по горизонтали (максимальное количество столбцов)

int maxY; /// Количество элементов по вертикали (максимальное количество строк)

int maxZ;

T nullElement; /// Пустой элемент, который возвращается если элемент с нужными координатами не был задан

public Matrix(int px, int py, int pz, T nullElementParam) /// Конструктор

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.nullElement = nullElementParam;

}

public T this[int x, int y, int z] /// Индексатор для доступа к данных

{

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);

}

}

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

{

if (x < 0 || x >= this.maxX) throw new Exception("x=" + x + " выходит за границы");

if (y < 0 || y >= this.maxY) throw new Exception("y=" + y + " выходит за границы");

if (z < 0 || z >= this.maxY) throw new Exception("z=" + z + " выходит за границы");

}

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

{

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

}

public override string ToString() /// Приведение к строке

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

StringBuilder b = new StringBuilder();

int a = 0;

int l = 0;

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

{

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

{

b.Append("[");

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

{

if (i > 0) b.Append("\t");

l = this[i, i, i].ToString().Length;

if (this[i, j, k].Equals(nullElement)) b.Append(Convert.ToChar(45), l);

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

}

a = 0;

b.Append("]\n");

}

}

return b.ToString();

}

}

}

*Вывод*

