Отчет по лабораторной работе 3 по курсу

«Базовые компоненты интернет технологий (БКИТ)»

Выполнил

Студент группы ИУ5-33

Зинченко Ярослав Игоревич

Москва 2017

**Требования:**

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 «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 на языке C#:**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp6

{

class Program

{

static void Main(string[] args)

{

Rectangle rect = new Rectangle(5, 4);

Square square = new Square(5);

Circle circle = new Circle(5);

Console.WriteLine("\nArrayList");

ArrayList al = new ArrayList();

al.Add(circle);

al.Add(rect);

al.Add(square);

foreach (object x in al) Console.WriteLine(x);

Console.WriteLine("\nArrayList - сортировка");

al.Sort();

foreach (object x in al) Console.WriteLine(x);

Console.WriteLine("\nList<GeometricFigure>");

List<GeometricFigure> fl = new List<GeometricFigure>();

fl.Add(circle);

fl.Add(rect);

fl.Add(square);

foreach (GeometricFigure x in fl) Console.WriteLine(x);

Console.WriteLine("\nList<GeometricFigure> - сортировка");

fl.Sort();

foreach (GeometricFigure x in fl) Console.WriteLine(x);

Console.WriteLine("\nМатрица");

Matrix<GeometricFigure> cube = new Matrix<GeometricFigure>(3, 3, 3, null);

cube[0, 0, 0] = rect;

cube[1, 1, 1] = square;

cube[2, 2, 2] = circle;

Console.WriteLine(cube.ToString());

Console.WriteLine("\nСписок");

SimpleList<GeometricFigure> list = new SimpleList<GeometricFigure>();

list.Add(square);

list.Add(rect);

list.Add(circle);

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

list.Sort();

Console.WriteLine("\nСортировка списка");

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

Console.WriteLine("\nСтек");

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

stack.Push(rect);

stack.Push(square);

stack.Push(circle);

while (stack.Count > 0)

{

GeometricFigure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

interface IPrint

{

void Print();

}

abstract class GeometricFigure : IComparable, IPrint

{

public GeometricFigure() { }

public virtual double Area()

{

return 0;

}

public abstract override string ToString();

public void Print()

{

Console.WriteLine(this);

}

public int CompareTo(object obj)

{

GeometricFigure p = (GeometricFigure)obj;

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

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

else return 1;

}

}

class Rectangle : GeometricFigure

{

public Rectangle(double height1, double width1)

{

\_height = height1;

\_width = width1;

}

private double \_height = 0;

public double height

{

get { return \_height; }

set { \_height = value; }

}

private double \_width = 0;

public double width

{

get { return \_width; }

set { \_width = value; }

}

public override double Area()

{

return \_width \* \_height;

}

public override string ToString()

{

return "Rectangle: " + width.ToString() + "x" + height.ToString() + ", S = " + Area().ToString();

}

}

class Square : Rectangle

{

public Square(double height1) : base(height1, height1) { }

public override double Area()

{

return height \* height;

}

public override string ToString()

{

return "Square: " + height.ToString() + "x" + height.ToString() + ", S = " + Area().ToString();

}

}

class Circle : GeometricFigure

{

public Circle(double radius)

{

\_radius = radius;

}

private double \_radius = 0;

public double radius

{

get { return \_radius; }

set { \_radius = value; }

}

public override double Area()

{

return Math.PI \* \_radius \* \_radius;

}

public override string ToString()

{

return "Circle: " + radius.ToString() + ", S = " + Area().ToString();

}

}

public class Matrix<T>

{

/// <summary>

/// Словарь для хранения значений

/// </summary>

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

/// <summary>

/// Количество элементов по горизонтали (максимальное количество столбцов)

/// </summary>

int maxX;

/// <summary>

/// Количество элементов по вертикали (максимальное количество строк)

/// </summary>

int maxY;

/// <summary>

/// Количество элементов по высоте (максимальное количество строк)

/// </summary>

int maxZ;

/// <summary>

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

/// </summary>

T nullElement;

/// <summary>

/// Конструктор

/// </summary>

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

{

maxX = px;

maxY = py;

maxZ = pz;

this.nullElement = nullElementParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </summary>

public T this[int x, int y, int z]

{

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);

}

}

/// <summary>

/// Проверка границ

/// </summary>

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

{

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");

}

/// <summary>

/// Формирование ключа

/// </summary>

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

{

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

}

/// <summary>

/// Приведение к строке

/// </summary>

/// <returns></returns>

public override string ToString()

{

//Класс StringBuilder используется для построения длинных строк

//Это увеличивает производительность по сравнению с созданием и склеиванием

//большого количества обычных строк

StringBuilder b = new StringBuilder();

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

{

b.Append("[");

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

{

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

b.Append("[");

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

{

if (this[i, j, k] != null)

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

else

b.Append("Null");

if (i != (maxX - 1)) b.Append(", ");

}

b.Append("]");

}

b.Append("]\n");

}

return b.ToString();

}

}

public class SimpleListItem<T>

{

/// <summary>

/// Данные

/// </summary>

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

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

///конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

/// <summary>

/// Список

/// </summary>

public class SimpleList<T> : IEnumerable<T>

where T : IComparable

{

/// <summary>

/// Первый элемент списка

/// </summary>

protected SimpleListItem<T> first = null;

/// <summary>

/// Последний элемент списка

/// </summary>

protected SimpleListItem<T> last = null;

/// <summary>

/// Количество элементов

/// </summary>

public int Count

{

get { return \_count; }

protected set { \_count = value; }

}

int \_count;

/// <summary>

/// Добавление элемента

/// </summary>

/// <param name="element"></param>

public void Add(T element)

{

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

this.Count++;

//Добавление первого элемента

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

//Добавление следующих элементов

else

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Просоединенный элемент считается последним

this.last = newItem;

}

}

/// <summary>

/// Чтение контейнера с заданным номером

/// </summary>

public SimpleListItem<T> GetItem(int number)

{

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

{

//Можно создать собственный класс исключения

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

}

SimpleListItem<T> current = this.first;

int i = 0;

//Пропускаем нужное количество элементов

while (i < number)

{

//Переход к следующему элементу

current = current.next;

//Увеличение счетчика

i++;

}

return current;

}

/// <summary>

/// Чтение элемента с заданным номером

/// </summary>

public T Get(int number)

{

return GetItem(number).data;

}

/// <summary>

/// Для перебора коллекции

/// </summary>

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

//Перебор элементов

while (current != null)

{

//Возврат текущего значения

yield return current.data;

//Переход к следующему элементу

current = current.next;

}

}

//Реализация обощенного IEnumerator<T> требует реализации необобщенног интерфейса

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// <summary>

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary>

/// Реализация алгоритма быстрой сортировки

/// </summary>

/// <param name="low"></param>

/// <param name="high"></param>

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 (i < high) Sort(i, high);

}

/// <summary>

/// Вспомогательный метод для обмена элементов при сортировке

/// </summary>

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

{

/// <summary>

/// Добавление в стек

/// </summary>

public void Push(T element)

{

Add(element);

}

/// <summary>

/// Чтение с удалением из стека

/// </summary>

public T Pop()

{

T element = Get(Count - 1);

SimpleListItem<T> listItem = GetItem(Count - 1);

listItem = null;

Count--;

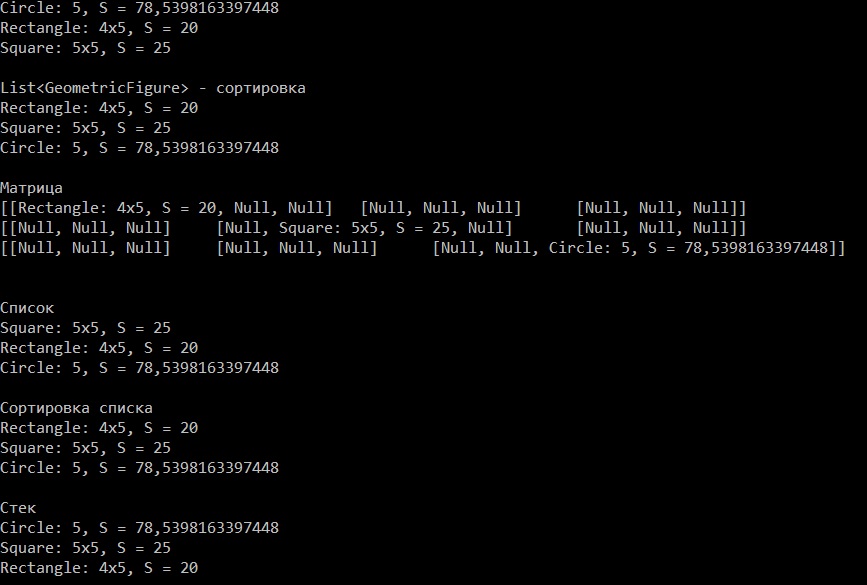
return element;

}

}

}

**Пример:**



**Диаграмма классов:**

****