|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

**Факультет «Информатика и системы управления»**

**Кафедра «Системы обработки информации и управления»**

Отчет по Лабораторной работе № 3

по дисциплине «Базовые компоненты интернет-технологий»

Выполнил:

студент(ка) группы ИУ5-34Б Бузин Е.М

подпись, дата

Проверил:

к.т.н., Ю.Е. Гапанюк

подпись, дата

2020 г.

**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.

**Текст программы**

using System;

using System.Collections.Generic;

using System.Collections;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Math;

namespace LAB3

{

class Figures

{

abstract class GeometricFigure : 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)

{

//Приведение параметра к типу "фигура"

GeometricFigure p = (GeometricFigure)obj;

//Сравнение

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 : GeometricFigure, IPrint

{

double height;

double width;

public double Height

{

set

{

if (this.height >= 0)

{

this.height = value;

}

else

{

Console.WriteLine("Высота не может быть отрицательной");

}

}

get

{

return this.height;

}

}

public double Width

{

set

{

if (this.width >= 0)

{

this.width = value;

}

else

{

Console.WriteLine("Ширина не может быть отрицательной");

}

}

get

{

return this.width;

}

}

public Rectangle(double h, double w)

{

this.height = h;

this.Height = h;

this.width = w;

this.Width = w;

this.Type = "Прямоугольник";

}

public override double Area()

{

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

if (this.width < 0 || this.height < 0)

{

Console.WriteLine("Площадь не может быть отрицательной");

Result = 0;

}

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 : GeometricFigure, IPrint

{

double radius;

public double Rad

{

set

{

if (this.radius >= 0)

{

this.radius = value;

}

else

{

Console.WriteLine("Радиус не может быть отрицательным");

}

}

get

{

return this.radius;

}

}

public Circle(double pr)

{

this.radius = pr;

this.Rad = pr;

this.Type = "Круг";

}

public override double Area()

{

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

if (this.radius < 0)

{

Console.WriteLine("Площадь не может быть отрицательной");

Result = 0;

}

return Result;

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

static void Main()

{

Rectangle rect = new Rectangle(5, 4);

Square square = new Square(5);

Circle circle = new Circle(5);

//ArrayList fl = new ArrayList();

//fl.Add(circle);

//fl.Add(rect);

//fl.Add(square);

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

//foreach (var x in fl) Console.WriteLine(x);

////сортировка

//fl.Sort();

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

//foreach (var x in fl) Console.WriteLine(x);

//List<GeometricFigure> al = new List<GeometricFigure>();

//al.Add(circle);

//al.Add(rect);

//al.Add(square);

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

//foreach (var x in al) Console.WriteLine(x);

////сортировка

//al.Sort();

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

//foreach (var x in al) Console.WriteLine(x);

//Console.WriteLine("\nМатрица");

//Matrix<GeometricFigure> matrix = new Matrix<GeometricFigure>(3, 3, 3,

//new FigureMatrixCheckEmpty());

//matrix[0, 0, 1] = rect;

//matrix[1, 1, 0] = square;

//matrix[2, 2, 2] = circle;

//Console.WriteLine(matrix.ToString());

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

//stack.Push(rect);

//stack.Push(square);

//stack.Push(circle);

////чтение данных из стека

//while (stack.Count > 0)

//{

// GeometricFigure f = stack.Pop();

// Console.WriteLine(f);

//}

bool f = true;

do

{

Console.WriteLine("1 - ArrayList:");

Console.WriteLine("2 - List<Figure>:");

Console.WriteLine("3 - SparseMatrix:");

Console.WriteLine("4 - SimpleStack:");

Console.WriteLine("5 - Exit");

int num = Convert.ToInt32(Console.ReadLine());

if (num <= 0 || num > 5)

{

Console.WriteLine("Повторите ввод пункта от 1 до 5:");

}

switch (num)

{

case 1:

ArrayList fl = new ArrayList();

fl.Add(circle);

fl.Add(rect);

fl.Add(square);

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

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

//сортировка

fl.Sort();

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

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

break;

case 2:

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

al.Add(square);

al.Add(circle);

al.Add(rect);

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

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

//сортировка

al.Sort();

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

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

break;

case 3:

Console.WriteLine("\nМатрица");

Matrix<GeometricFigure> matrix = new Matrix<GeometricFigure>(3, 3, 3,

new FigureMatrixCheckEmpty());

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = square;

matrix[2, 2, 2] = circle;

Console.WriteLine(matrix.ToString());

break;

case 4:

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

stack.Push(rect);

stack.Push(circle);

stack.Push(square);

//чтение данных из стека

while (stack.Count > 0)

{

GeometricFigure s = stack.Pop();

Console.WriteLine(s);

}

break;

case 5:

f = false;

break;

}

} while (f);

}

public interface IMatrixCheckEmpty<T>

{

/// <summary> /// Возвращает пустой элемент /// </summary> T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<GeometricFigure>

{

/// <summary> /// В качестве пустого элемента возвращается null /// </summary> public GeometricFigure getEmptyElement()

{

return null;

}

/// <summary> /// Проверка что переданный параметр равен null /// </summary> public bool checkEmptyElement(GeometricFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result; }

}

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 глубине (максимальное количество строк)

/// </summary>

int maxZ;

/// <summary>

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

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

//T nullElement;

/// <summary>

/// Конструктор

/// </summary>

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

public Matrix(int px, int py, int pz, IMatrixCheckEmpty<T> сheckEmptyParam)

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

//nullElement = nullElementParam;

this.сheckEmpty = сheckEmptyParam;

}

/// <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.сheckEmpty.getEmptyElement();

//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.maxZ) 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 i = 0; i < maxZ; i++)

{

b.Append("Слой ");

b.Append(i + "\n");

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

{

b.Append("[");

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

{

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

b.Append("\t");

if (!this.сheckEmpty.checkEmptyElement(this[k, j, i]))

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

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

}

b.Append("]\n");

}

b.Append("\n");

}

return b.ToString();

}

}

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

{

public void Push(T element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

/// <summary> /// Удаление и чтение из стека /// </summary>

public T Pop()

{

//default(T) - значение для типа T по умолчанию

T Result = default(T);

//Если стек пуст, возвращается значение по умолчанию для типа

if (this.Count == 0) return Result;

//Если элемент единственный

if (this.Count == 1)

{

Result = this.first.data;

//обнуляются указатели начала и конца списка

this.first = null;

this.last = null;

}

//В списке более одного элемента

else

{

//Поиск предпоследнего элемента

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

//Чтение значения из последнего элемента

Result = newLast.next.data;

//предпоследний элемент считается последним

this.last = newLast;

//последний элемент удаляется из списка

newLast.next = null;

}

//Уменьшение количества элементов в списке

this.Count--;

//Возврат результата

return Result;

}

}

public class SimpleListItem<T>

{

/// <summary>

/// Данные

public T data { get; set; }

/// <summary>

/// Следующий элемент

/// </summary>

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

///конструктор

public SimpleListItem(T param)

{

this.data = param;

}

}

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> 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;

//Присоединенный 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;

}

}

//Данный метод добавляется автоматически при реализации интерфейса

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

/// Cортировка

/// </summary>

public void Sort()

{

Sort(0, this.Count - 1);

}

/// <summary> /// Алгоритм быстрой сортировки /// </summary> 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);

}

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;

}

}

}

}

**Экранные формы с примерами выполнения программы**

 

 