МГТУ им. Н.Э. Баумана

Факультет «Информатика и системы управления»

ДИСЦИПЛИНА:

«Электротехника»

Отчет по лабораторной работе №3

Выполнила:

Студент 2 курса

Факультет ИУ

Группа ИУ5-31Б

Кемерова А.М.

Преподаватель:

Белодедов М.В.

**Цель 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 (разобранного в пособии). Необходимо добавить в класс методы:
   1. public void Push(T element) – добавление в стек;
   2. public T Pop() – чтение с удалением из стека.
8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

**Код программы**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace \_3

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<GeometryFigure>

{

public GeometryFigure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(GeometryFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

class Program

{

static void Main(string[] args)

{

Rectangle A = new Rectangle(12, 3);

Square B = new Square(12);

Circle C = new Circle(4);

Console.WriteLine("-------------------------------------");

Console.WriteLine("[Создание коллекции класса ArrayList]");

Console.WriteLine("-------------------------------------");

ArrayList al = new ArrayList();

al.Add(A);

al.Add(B);

al.Add(C);

Console.WriteLine("Перед 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("----------------------------------------");

Console.WriteLine("[Создание коллекции класса List<Figure>]");

Console.WriteLine("----------------------------------------");

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

fl.Add(A);

fl.Add(B);

fl.Add(C);

Console.WriteLine("Перед сортировкой:");

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

//сортировка

fl.Sort();

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

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

Console.WriteLine("------------------------");

Console.WriteLine("[Создание класса Matrix]");

Console.WriteLine("------------------------");

Matrix<GeometryFigure> matrix = new Matrix<GeometryFigure>(3, 3, new FigureMatrixCheckEmpty());

matrix[0, 0] = A;

matrix[1, 1] = B;

matrix[2, 2] = C;

Console.WriteLine(matrix.ToString());

Console.WriteLine("-------------------------------");

Console.WriteLine("[Создание класса «SimpleStack»]");

Console.WriteLine("-------------------------------");

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

stack.Push(A);

stack.Push(B);

stack.Push(C);

while (stack.Count > 0)

{

GeometryFigure f = stack.Pop();

Console.WriteLine(f);

}

}

}

interface IPrint

{

void Print();

}

abstract class GeometryFigure : IComparable

{

public string Type

{

get

{

return this.\_Type;

}

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)

{

GeometryFigure p = (GeometryFigure)obj;

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

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

else return 1;

}

}

class Rectangle : GeometryFigure, 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 : GeometryFigure, 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());

}

}

public class Matrix<T>

{

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

int maxX;

int maxY;

IMatrixCheckEmpty<T> сheckEmpty;

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

{

this.maxX = px;

this.maxY = py;

this.сheckEmpty = сheckEmptyParam;

}

public T this[int x, int y]

{

set

{

CheckBounds(x, y);

string key = DictKey(x, y);

this.\_matrix.Add(key, value);

}

get

{

CheckBounds(x, y);

string key = DictKey(x, y);

if (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.сheckEmpty.getEmptyElement();

}

}

}

void CheckBounds(int x, int y)

{

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

{

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

}

if

(y < 0 || y >= this.maxY)

{

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

}

}

string DictKey(int x, int y)

{

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

}

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

b.Append("[");

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

{

if (i > 0)

{

b.Append("\t");

}

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

{

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

}

else

{

b.Append(" - ");

}

}

b.Append("]\n");

}

return b.ToString();

}

}

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.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 { return \_count; }

protected set { \_count = value; }

}

int \_count;

public void Add(T element)

{

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

this.Count++;

if (last == null)

{

this.first = newItem;

this.last = newItem;

}

else

{

this.last.next = newItem;

this.last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

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

{

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

}

SimpleListItem<T> current = this.first;

int i = 0;

while (i < number)

{

current = current.next;

i++;

}

return current;

}

public T Get(int number)

{

return GetItem(number).data;

}

public IEnumerator<T> GetEnumerator()

{

SimpleListItem<T> current = this.first;

while (current != null)

{

yield return current.data;

current = current.next;

}

}

System.Collections.IEnumerator

System.Collections.IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, this.Count - 1);

}

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;

}

}

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

{

public void Push(T element)

{

Add(element);

}

public T Pop()

{

T Result = default(T);

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

if (this.Count == 1)

{

Result = this.first.data;

this.first = null;

this.last = null;

}

else

{

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

Result = newLast.next.data;

this.last = newLast;

newLast.next = null;

}

this.Count--;

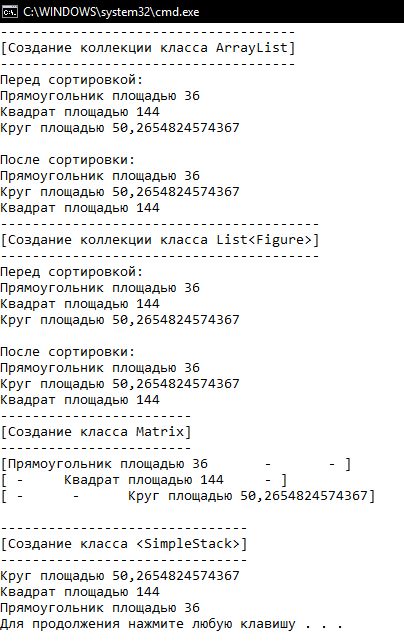
return Result;

}

}

}

**Результаты работы программы**

****