Московский государственный технический университет

им. Н.Э. Баумана

Отчет по лабораторной работе № 3

|  |  |
| --- | --- |
|  | Выполнила:  Персиянова В.  ИУ5-33 |

г. Москва

2017 г.

1. **Описание задания 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 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

9

«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.

1. **Текст программы.**

Program.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace FigureCollections

{

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("ArrayList");

ArrayList al = new ArrayList();

al.Add(circle);

al.Add(rect);

al.Add(square);

Console.WriteLine("\nПеред сортировкой");

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

Console.WriteLine("\nПосле сортировки");

al.Sort();

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

//------------------------------------------------

Console.WriteLine("\nList<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);

//сортировка

fl.Sort();

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

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

//------------------------------------------------

Console.WriteLine("\nМатрица");

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

matrix[0, 0, 0] = rect;

matrix[1, 1, 1] = square;

matrix[2, 2, 2] = circle;

Console.WriteLine(matrix.ToString());

//------------------------------------------------

Console.WriteLine("\nСтек");

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

//добавление данных в стек

Console.WriteLine("\nДобавление данных в стек");

stack.Push(rect);

Console.WriteLine(rect);

stack.Push(square);

Console.WriteLine(square);

stack.Push(circle);

Console.WriteLine(circle);

//чтение данных из стека

Console.WriteLine("\nЧтение данных из стека");

while (stack.Count > 0)

{

Figure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

IPrint.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

interface IPrint

{

void Print();

}

}

Square.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

class Square : Rectangle, IPrint

{

public Square(double size)

: base(size, size)

{

this.Type = "Квадрат";

}

}

}

Rectangle.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

class Rectangle : Figure, IPrint

{

double height;

double width;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="ph">Высота</param>

/// <param name="pw">Ширина</param>

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());

}

}

}

Figure.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

/// <summary>

/// Класс фигура

/// </summary>

abstract class Figure : IComparable

{

/// <summary>

/// Тип фигуры

/// </summary>

public string Type

{

get

{

return this.\_Type;

}

protected set

{

this.\_Type = value;

}

}

string \_Type;

/// <summary>

/// Вычисление площади

/// </summary>

/// <returns></returns>

public abstract double Area();

/// <summary>

/// Приведение к строке, переопределение метода Object

/// </summary>

/// <returns></returns>

public override string ToString()

{

return this.Type + " площадью " + this.Area().ToString();

}

/// <summary>

/// Сравнение элементов (для сортировки)

/// this - левый параметр сравнения

/// </summary>

/// <param name="obj">правый параметр сравнения</param>

/// <returns>

/// -1 - если левый параметр меньше правого

/// 0 - параметры равны

/// 1 - правый параметр меньше левого

/// </returns>

public int CompareTo(object obj)

{

//Приведение параметра к типу "фигура"

Figure p = (Figure)obj;

//Сравнение

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

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

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

}

}

}

Circle.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

class Circle : Figure, IPrint

{

double radius;

/// <summary>

/// Основной конструктор

/// </summary>

/// <param name="ph">Высота</param>

/// <param name="pw">Ширина</param>

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());

}

}

}

EmptyFigure.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

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();

}

/// <summary>

/// Сравнение элементов (для сортировки)

/// this - левый параметр сравнения

/// </summary>

/// <param name="obj">правый параметр сравнения</param>

/// <returns>

/// -1 - если левый параметр меньше правого

/// 0 - параметры равны

/// 1 - правый параметр меньше левого

/// </returns>

public int CompareTo(object obj)

{

//Приведение параметра к типу "фигура"

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())

}

}

}

FigureMatrixCheckEmpty.cs

using System;

namespace FigureCollections

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Figure>

{

public Figure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Figure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

IMatrixCheckEmpty.cs

using System;

namespace FigureCollections

{

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

}

Matrix.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace FigureCollections

{

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> сheckEmpty;

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

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

{

set

{

CheckBounds(x, y, x);

string key = DictKey(x, y, z);

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

}

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();

}

}

}

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

{

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");

}

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

{

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

}

}

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

{

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

}

public override string ToString()

{

StringBuilder b = new StringBuilder();

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

{

b.Append("["+k+"]:\n");

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, k]))

{

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

}

else

{

b.Append(" - ");

}

}

b.Append("]\n");

}

}

return b.ToString();

}

}

}

SimpleList.cs

using System;

using System.Collections.Generic;

namespace FigureCollections

{

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;

}

}

}

SimpleListItem.cs

using System;

namespace FigureCollections

{

public class SimpleListItem<T>

{

public T data { get; set; }

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

public SimpleListItem(T param)

{

this.data = param;

}

}

}

SimpleStack.cs

using System;

namespace FigureCollections

{

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;

}

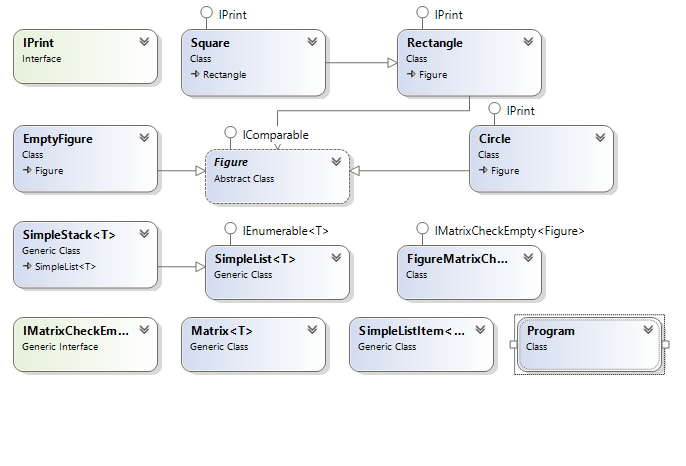
this.Count--;

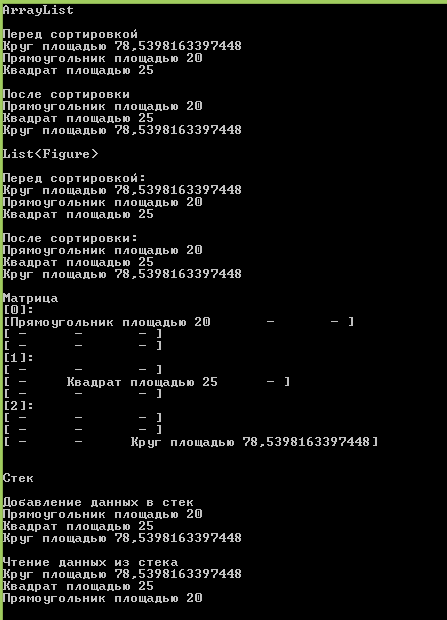
return Result;

}

}

}

1. **Диаграмма классов.**
2. **Результат выполнения программы.**

****