2018

Отчет по лабораторной работе № 3

Базовые компоненты интернет технологий

Зубков андрей, иу5-34б

Описание задания

Добавить реализацию возможности сортировки геометрических фигур для класса Figure добавить реализацию интерфейса IComparable (сортировка производится по площади фигуры). Показать работу на примере ArrayList и List.

Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями. Осуществить вывод элементов в методе ToString() и разработать пример ее использования.

Реализовать класс SimpleStack на основе односвязного списка (наследуется от класса SimpleList). Добавить методы: Push, Pop. Реализовать пример использования.

Текст программы

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab3 {

// Figures classes and interface IPrint

#region

/// <summary>

/// Geometry figure class

/// </summary>

abstract class Figure : IComparable {

/// <summary>

/// Calculating figure area

/// </summary>

public abstract double Area();

/// <summary>

/// Figure class

/// </summary>

string \_type;

public string type {

get { return this.\_type; }

protected set { this.\_type = value; }

}

/// <summary>

/// Override method ToString

/// </summary>

public override string ToString() {

return this.type + " area = " + this.Area().ToString();

}

public int CompareTo(object obj) {

Figure f = (Figure)obj;

if (this.Area() < f.Area())

return -1;

else if (this.Area() == f.Area())

return 0;

else

return 1;

}

}

/// <summary>

/// Legacy class Rectangle, basic class - Figure

/// </summary>

class Rectangle : Figure, IPrint

{

public Rectangle(double w, double h) { this.width = w; this.height = h; this.type = "Rectangle"; }

protected double \_width = 0;

public double width

{

get { return \_width; }

set { \_width = value; }

}

private double \_height = 0;

public double height

{

get { return \_height; }

set { \_height = value; }

}

/// <summary>

/// Calculating figure area

/// </summary>

/// <returns></returns>

public override double Area()

{

return this.width \* this.height;

}

/// <summary>

/// Show general figure parametres

/// </summary>

public void Print()

{

Console.WriteLine(ToString());

}

}

/// <summary>

/// Legacy class Square, basic class - Rectangle

/// </summary>

class Square : Rectangle, IPrint

{

public Square(double a) : base(a, a) { this.type = "Square"; }

/// <summary>

/// Show general figure parametres

/// </summary>

public void Print()

{

Console.WriteLine(ToString());

}

}

/// <summary>

/// Legacy class Circle, basic class - Figure

/// </summary>

class Circle : Figure, IPrint

{

public Circle(double r) { this.radius = r; this.type = "Circle"; }

private double \_radius = 0;

public double radius

{

get { return \_radius; }

set { \_radius = value; }

}

/// <summary>

/// Calculating figure area

/// </summary>

/// <returns></returns>

public override double Area()

{

return this.radius \* this.radius \* Math.PI;

}

/// <summary>

/// Show general figure parametres

/// </summary>

public void Print()

{

Console.WriteLine(ToString());

}

}

interface IPrint

{

/// <summary>

/// Show general figure parametres

/// </summary>

void Print();

}

#endregion // //

// SparseMatrix

#region

public interface ISparseMatrixCheckEmpty<T> {

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureSparseMatrixCheckEmpty : ISparseMatrixCheckEmpty<Figure> {

public Figure getEmptyElement() {

return null;

}

public bool checkEmptyElement(Figure element) {

bool result = false;

if (element == null) {

result = true;

}

return result;

}

}

public class SparseMatrix<T> {

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

/// <summary>

/// Maximum number of horizontal elements

/// </summary>

int maxX;

/// <summary>

/// maximum number of vertical elements

/// </summary>

int maxY;

/// <summary>

/// maximum number of aplicate elements

/// </summary>

int maxZ;

/// <summary>

/// realization interface

/// </summary>

ISparseMatrixCheckEmpty<T> checkEmpty;

public SparseMatrix(int px, int py, int pz,

ISparseMatrixCheckEmpty<T> checkEmptyParam) {

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.checkEmpty = checkEmptyParam;

}

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

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

throw new ArgumentOutOfRangeException("x", "x=" + x + "go beyond");

}

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

{

throw new ArgumentOutOfRangeException("y", "y=" + y + "go beyond");

}

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

{

throw new ArgumentOutOfRangeException("z", "z=" + z + "go beyond");

}

}

/// <summary>

/// formation key

/// </summary>

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

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

}

/// <summary>

/// indexer to access the data

/// </summary>

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

set {

CheckBounds(x, y, z);

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.checkEmpty.getEmptyElement();

}

}

}

public override string ToString() {

StringBuilder b = new StringBuilder();

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

{

b.Append("\nz = " + (k + 1).ToString() + ":\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.checkEmpty.checkEmptyElement(this[i, j , k]))

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

else

b.Append(" \* ");

}

b.Append("]\n");

}

}

return b.ToString();

}

}

#endregion

// SimpleList, SimpleStack

#region

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;

int \_count;

public int Count {

get { return \_count; }

protected set { \_count = value; }

}

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("Out of bounds");

}

SimpleListItem<T> current = this.first;

for (int i = 0; i < number; i++) {

current = current.next;

}

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;

}

}

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;

}

IEnumerator IEnumerable.GetEnumerator() {

return GetEnumerator();

}

}

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

public void Push(T item) { Add(item); }

public T Pop() {

T res = default(T); // default value for the following type

if (this.Count == 0) {

return res;

}

if (this.Count == 1) {

res = this.first.data;

this.first = null;

this.last = null;

}

else {

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

res = NewLast.next.data;

this.last = NewLast;

NewLast.next = null;

}

this.Count--;

return res;

}

}

#endregion

class Program

{

static void Main(string[] args) {

Rectangle r = new Rectangle(5, 9);

Square s = new Square(10);

Circle c = new Circle(3);

ArrayList aList = new ArrayList();

aList.Add(r);

aList.Add(s);

aList.Add(c);

Console.WriteLine("Array list before sorting");

foreach (var x in aList)

Console.Write("{0} ", x);

aList.Sort();

Console.WriteLine("\nArray list after sorting");

foreach (var x in aList)

Console.Write("{0} ", x);

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

list.Add(r);

list.Add(s);

list.Add(c);

Console.WriteLine("\nList before sorting");

foreach (var x in list)

Console.Write("{0} ", x);

list.Sort();

Console.WriteLine("\nList after sorting");

foreach (var x in list)

Console.Write("{0} ", x);

SparseMatrix<Figure> matrix = new SparseMatrix<Figure>

(3, 3, 3, new FigureSparseMatrixCheckEmpty());

matrix[0, 0, 0] = r;

matrix[1, 1, 1] = s;

matrix[2, 2, 2] = c;

Console.WriteLine(matrix.ToString());

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

stack.Push(r);

stack.Push(s);

stack.Push(c);

foreach (Figure f in stack) {

Console.WriteLine(f);

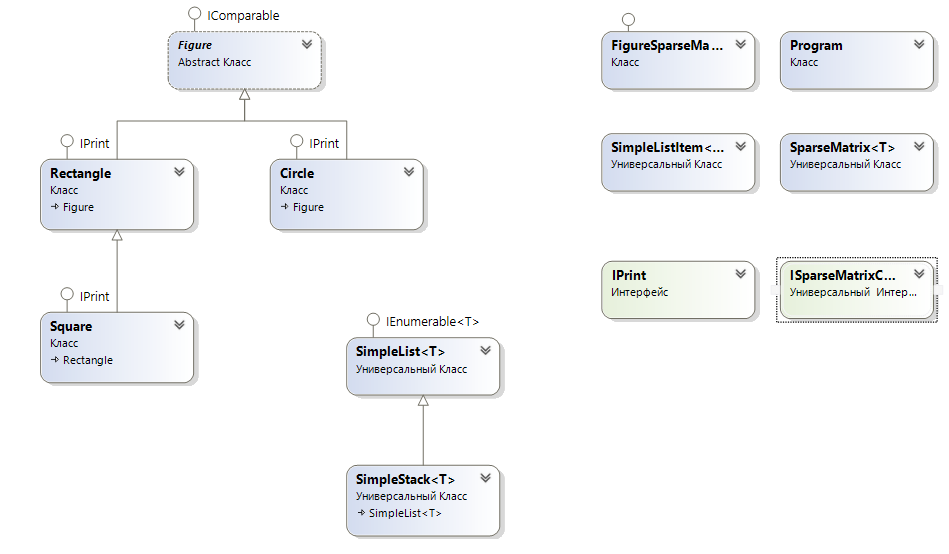
}

Console.Read();

}

}

}

Диаграмма классов

Пример работы программы

