Отчёт по лабораторной работе №3 по курсу БКИТ

«Работа с коллекциями»

Подготовил

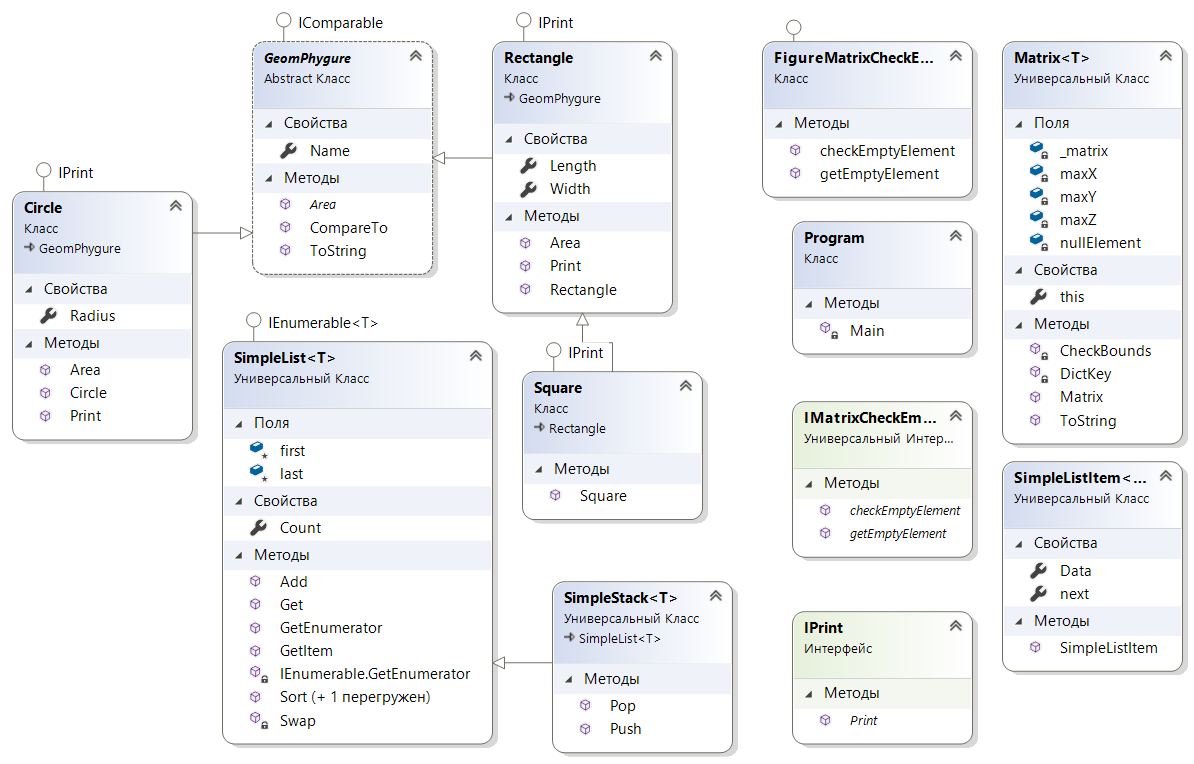
Ионов С.А.

ИУ5-31Б

1. 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 (разобранного в пособии). 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.
9. Диаграмма классов



1. Текст программы

using System;

using System.Collections;

using System.Collections.Generic;

using System.Text;

namespace Lab3

{

using Geometry;

using Lst;

using SparseMatrix;

class Program

{

static void Main(string[] args)

{

Rectangle rec1 = new Rectangle(2, 4);

Square sq1 = new Square(2);

Circle cir1 = new Circle(9);

ArrayList arrayList = new ArrayList

{

rec1, sq1, cir1

};

Console.WriteLine("До сортировки");

foreach (var i in arrayList)

Console.WriteLine(i);

arrayList.Sort();

Console.WriteLine("\nПосле сортировки");

foreach (var i in arrayList)

Console.WriteLine(i);

List<GeomPhygure> geomPhygures = new List<GeomPhygure>();

geomPhygures.Add(rec1);

geomPhygures.Add(cir1);

geomPhygures.Add(sq1);

Console.WriteLine("\nДо сортировки");

foreach (var i in geomPhygures)

Console.WriteLine(i);

geomPhygures.Sort();

Console.WriteLine("\nПосле сортировки");

foreach (var i in geomPhygures)

Console.WriteLine(i);

Matrix<GeomPhygure> cube = new Matrix<GeomPhygure>(3, 3, 3, new FigureMatrixCheckEmpty());

cube[0, 0, 0] = rec1;

cube[1, 1, 1] = sq1;

cube[2, 2, 2] = cir1;

Console.WriteLine(cube.ToString());

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

stack.Push(rec1); stack.Push(sq1); stack.Push(cir1); //добавление данных в стек

while (stack.Count > 0) //чтение данных из стека

{

GeomPhygure f = stack.Pop();

Console.WriteLine(f);

}

Console.ReadLine();

}

}

}

namespace Geometry

{

/// <summary>

/// Класс геометрическая фигура

/// </summary>

abstract class GeomPhygure: IComparable

{

/// <summary>

/// Название фигуры

/// </summary>

public string Name { get; protected set; }

/// <summary>

/// Площадь фигуры

/// </summary>

public abstract double Area();

public int CompareTo(object obj)

{

GeomPhygure f = (GeomPhygure)obj;

if (Area() < f.Area()) return -1;

else if (Area() == f.Area()) return 0;

else return 1;

}

/// <summary>

/// Переопределение метода ToString(), Краткая информация о фигуре

/// </summary>

public override string ToString()

{

return Name + " имеет площадь " + Area();

}

}

interface IPrint

{

void Print();

}

/// <summary>

/// Класс прямоугольник

/// </summary>

class Rectangle : GeomPhygure, IPrint

{

/// <summary>

/// Длина

/// </summary>

public double Length { get; set; }

/// <summary>

/// Ширина

/// </summary>

public double Width { get; set; }

public Rectangle(double p1, double p2)

{

Length = p1;

Width = p2;

Name = "Прямоугольник";

}

public override double Area()

{

return Length \* Width;

}

public void Print()

{

Console.WriteLine(ToString());

}

}

/// <summary>

/// Класс квадрат

/// </summary>

class Square : Rectangle, IPrint

{

public Square(double size) : base(size, size)

{

Name = "Квадрат";

}

}

/// <summary>

/// Класс круг

/// </summary>

class Circle : GeomPhygure, IPrint

{

/// <summary>

/// Радиус

/// </summary>

public double Radius { get; set; }

public Circle(double p)

{

Radius = p;

Name = "Круг";

}

public override double Area()

{

return Math.PI \* Radius \* Radius;

}

public void Print()

{

Console.WriteLine(ToString());

}

}

}

namespace SparseMatrix

{

public class Matrix<T>

{

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

int maxX;

int maxY;

int maxZ;

IMatrixCheckEmpty<T> nullElement;

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

{

this.maxX = px;

this.maxY = py;

this.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 (\_matrix.ContainsKey(key))

{

return \_matrix[key];

}

else

{

return nullElement.getEmptyElement();

}

}

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

//большого количества обычных строк

StringBuilder b = new StringBuilder();

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

{

b.Append("Таблица " + (k+1) + '\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.nullElement.checkEmptyElement(this[i, j, k]))

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

else

b.Append("-");

}

b.Append("]\n");

}

b.Append('\n');

}

return b.ToString();

}

}

public interface IMatrixCheckEmpty<T>

{

T getEmptyElement();

bool checkEmptyElement(T element);

}

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<Geometry.GeomPhygure>

{

public Geometry.GeomPhygure getEmptyElement()

{

return null;

}

public bool checkEmptyElement(Geometry.GeomPhygure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

namespace Lst

{

public class SimpleListItem<T>

{

public T Data { get; set; }

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

public SimpleListItem(T param)

{

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; protected set; }

public void Add(T element)

{

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

Count++;

if (last == null)

{

first = newItem;

last = newItem;

}

else

{

last.next = newItem;

last = newItem;

}

}

public SimpleListItem<T> GetItem(int number)

{

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

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

SimpleListItem<T> current = 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;

}

}

IEnumerator

IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public void Sort()

{

Sort(0, 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 (Count == 0) return Result;

if (Count == 1)

{

Result = first.Data;

first = null;

last = null;

}

else

{

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

Result = newLast.next.Data;

last = newLast;

newLast.next = null;

}

Count--;

return Result;

}

}

}

1. Экранные формы с примерами выполнения программы

