|  |  |
| --- | --- |
| 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)** |

**Отчёт**

***Лабораторная работа №1***

Выполнил студент: \_\_***Сафиуллин Антон Павлович\_\_\_\_\_\_\_\_\_\_***

*фамилия, имя, отчество*

Группа: \_\_\_\_***РТ5-31Б\_\_\_\_\_\_\_\_***

**Описание задания**

**Лабораторная работа №3**

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**

**Program.cs:**

using System;

using System.Collections.Generic;

namespace LabRabota\_3

{

class Program

{

public static void cout(string per)

{

Console.WriteLine(per);

}

public static double dcin()

{

int error = 1;

double pev;

while (error == 1)//Вввод переменной

{

try

{

string per = Console.ReadLine();

pev = double.Parse(per);

return pev;

}

catch

{

cout("Введите корректную переменную: ");

}

finally { }

}

return 0;

}

static void Main(string[] args)

{

cout("Программу создал Сафиуллин А.П. РТ5-31Б");

Rectangle KK, KK1;

Sqare TT,TT1;

Circle KR1, KR2, KR3;

TT = new Sqare(4);

TT1 = new Sqare(10);

KK = new Rectangle(5, 4);

KK1 = new Rectangle(6, 4, "2");

KR1 = new Circle(4);// Добавим круг

KR2 = new Circle(5, "2");// Добавим круг 2

KR3 = new Circle(7, "3");// Добавим круг 3

KK.Print();

TT.Print();

List<GeomFigure> geomFigures= new List<GeomFigure> {TT,KK};

geomFigures.Add(KK1);

geomFigures.Add(KK);

geomFigures.Add(TT);

geomFigures.Add(TT1);

geomFigures.Add(KR1);

geomFigures.Add(KR2);

geomFigures.Add(KR3);

cout("До сортировки: ");

IPrint.LPrint(geomFigures,-6);

IComparablee.Sort(geomFigures);

cout(""); cout("");

cout("После сортировки: ");

IPrint.LPrint(geomFigures,8);

Console.WriteLine("\nМатрица");

Matrix<GeomFigure> matrix = new Matrix<GeomFigure>(5,5,5, new FigureMatrixCheckEmpty() );

int x = 1, y = 1, z = 1;

foreach (var item in geomFigures)

{

if (item.name == "Квадрат") {

matrix[x, 0, 0] = item;

x++;

}

if (item.name == "Прямоугольник") {

matrix[0, y, 0] = item;

y++;

}

if (item.name == "Круг") {

matrix[0, 0, z] = item;

z++;

}

}

Console.WriteLine(matrix.ToString());

cout(""); cout(""); cout("");

cout("СТЭК");

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

stack.Push(TT);

stack.Push(TT1);

Console.WriteLine(stack.Pop().ToString());

stack.Push(KK1);

stack.Push(KK);

Console.WriteLine(stack.Pop().ToString());

Console.WriteLine(stack.Pop().ToString());

Console.WriteLine(stack.Pop().ToString());

Console.WriteLine("Элементов в стеке: {0}", stack.Count);

}

}

}

**Sqare.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

class Sqare : Rectangle, IPrint

{

public Sqare(double W) : base(W, W)

{

Name = "Квадрат";

Nameles = "Квадрат";

}

public Sqare(double W,string n) : base(W, W)

{

Name = "Квадрат";

Nameles = "Квадрат "+n;

}

public override string ToString()

{

return Name.ToString() + " Ширина && Длина фигуры: " + Width.ToString() + " Площадь фигуры: " + Area.ToString();

}

}

}

**Circle.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

class Circle : GeomFigure, IPrint

{

private double radius;

protected string newArea;

public override double Areaa()

{

return Math.PI \* radius \* radius;

}

/// <summary>

/// Создать фигуру круг

/// </summary>

/// <param name="R">Радиус</param>

public Circle(double R)

{

radius = R;

newArea = $"{radius \* radius}\*Pi";

Area = R \* R \* Math.PI;

Name = "Круг";

Nameles = "Круг";

}

public Circle(double R,string n)

{

radius = R;

newArea = $"{radius \* radius}\*Pi";

Area = R \* R \* Math.PI;

Name = "Круг";

Nameles = "Круг "+n;

}

/// <summary>

/// Вывод Радиуса

/// </summary>

public double Radius

{

get

{

return radius;

}

}

/// <summary>

/// Вывод названия фигуры

/// </summary>

public string name

{

get

{

return Name;

}

}

public void display()

{

cout($"Радиус фигуры {Name}: {radius}");

}

public override string ToString()

{

return Name.ToString() + " Радиус фигуры: " + Radius.ToString() + " Площадь фигуры: " + newArea.ToString();

}

}

}

**Figure.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

/// <summary>

/// Класс геометрическая фигура

/// </summary>

abstract class GeomFigure : IComparable

{

public abstract double Areaa();

protected double Area;

protected string Name;

protected string Nameles;

public double area

{

get

{

return Area;

}

}

public string name

{

get

{

return Name;

}

set { Name = value; }

}

public string nameles

{

get

{

return Nameles;

}

set { Nameles = value; }

}

public static bool operator > (GeomFigure F1, GeomFigure F2) // операторы сравнения

{

if (F1.area > F2.area)

{

return true;

}

else { return false;}

}

public static bool operator <(GeomFigure F1, GeomFigure F2)// операторы сравнения

{

if (F1.area < F2.area)

{

return true;

}

else { return false; }

}

public int CompareTo(object obj)

{

//Приведение параметра к типу "фигура"

GeomFigure p = (GeomFigure)obj;

//Сравнение

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

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

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

}

/// <summary>

/// Вывод

/// </summary>

/// <param name="per">выводимая переменная</param>

public static void cout(string per)

{

Console.WriteLine(per);

}

/// <summary>

/// Ввод переменой типа double

/// </summary>

/// <returns></returns>

public static double dcin()

{

int error = 1;

double pev;

while (error == 1)//Вввод переменной

{

try

{

string per = Console.ReadLine();

pev = double.Parse(per);

return pev;

}

catch

{

cout("Введите корректную переменную: ");

}

finally { }

}

return 0;

}

}

}

**IComparablee.cs:**

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Runtime.InteropServices.WindowsRuntime;

using System.Text;

namespace LabRabota\_3

{

interface IComparablee

{

public static GeomFigure[] Sort(GeomFigure[]mas) //Сортировка фигур по площади в массиве

{

{

GeomFigure temp;

for (int i = 0; i < mas.Length; i++)

{

for (int j = i + 1; j < mas.Length; j++)

{

if (mas[i].area > mas[j].area)

{

temp = mas[i];

mas[i] = mas[j];

mas[j] = temp;

}

}

}

/\*Console.WriteLine();

Console.WriteLine("Площади по возрастанию: ");

for (int i=0; i<mas.Length;i++)

{

if (mas[i].name=="Квадрат") Console.WriteLine($"{i+1})Фигура: {mas[i].name} Площадь:{mas[i].area}");

else if (mas[i].name == "Прямоугольник") Console.WriteLine($"{i + 1})Фигура: {mas[i].name} Площадь:{mas[i].area}");

else if (mas[i].name == "Круг") Console.WriteLine($"{i + 1})Фигура: {mas[i].name} Площадь:{mas[i].area:f3}");

}\*/

return mas;

}

}

public static List<GeomFigure> Sort(List<GeomFigure> geomFigures) //Сортировка фигур в списке типа Лист

{

GeomFigure [] mas= new GeomFigure[geomFigures.Count]; //Создание массива размера массива лист

int i = 0;

foreach (var item in geomFigures)

{

mas[i] = item;// Запись элементов листа в массив

i++;

}

Sort(mas);// Сортировка массива

geomFigures.Clear();//Очистка изначального массива

for (int j = 0; j < i; j++)// Перезапись массива отсортированными фигурами

{

geomFigures.Add(mas[j]);

}

return geomFigures;

}

}

}

**IPrint.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

interface IPrint

{

private static void cout(string per)

{

Console.WriteLine(per);

}

public void Print()

{

this.ToString();

}

public static void LPrint(List<GeomFigure> list, int k)

{

int i = 0;

cout("\n");

Console.SetCursorPosition(10, k+10);

Console.WriteLine("Фигуры в массиве: ");

Console.Write("Номер в массиве: "); ; Console.Write("Название фигуры: ");Console.WriteLine(" Площадь: ");

foreach (var item in list)

{

Console.Write(i + 1);Console.SetCursorPosition(17, k+11+i+1);Console.Write(item.nameles); Console.SetCursorPosition(36, k+11+i+1); Console.WriteLine($"{item.area:f3}");

i++;

}

}

}

}

**Matrix.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace LabRabota\_3

{

class FigureMatrixCheckEmpty : IMatrixCheckEmpty<GeomFigure>

{

/// <summary>

/// В качестве пустого элемента возвращается null

/// </summary>

public GeomFigure getEmptyElement()

{

return null;

}

/// <summary>

/// Проверка что переданный параметр равен null

/// </summary>

public bool checkEmptyElement(GeomFigure element)

{

bool Result = false;

if (element == null)

{

Result = true;

}

return Result;

}

}

}

public interface IMatrixCheckEmpty<T>

{

/// <summary>

/// Возвращает пустой элемент

/// </summary>

T getEmptyElement();

/// <summary>

/// Проверка что элемент является пустым

/// </summary>

bool checkEmptyElement(T element);

}

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 вертикали .....(максимальное количество строк)

/// </summary>

int maxZ;

/// <summary>

/// Реализация интерфейса для проверки пустого элемента

/// </summary>

IMatrixCheckEmpty<T> сheckEmpty;

/// <summary>

/// Конструктор

/// </summary>

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

{

this.maxX = px;

this.maxY = py;

this.maxZ = pz;

this.сheckEmpty = сheckEmptyParam;

}

/// <summary>

/// Индексатор для доступа к данных

/// </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.сheckEmpty.getEmptyElement();

}

}

}

/// <summary>

/// Проверка границ

/// </summary>

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");

}

}

/// <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()

{

StringBuilder b = new StringBuilder();

Console.WriteLine("XY");

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

{

int k = 0;

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

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

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

Console.WriteLine("XZ");

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

{

int k = 0;

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

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

{

//Добавить приведенный к строке текущий элемент

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

}

else

{

//Иначе добавить признак пустого значения

b.Append(" - ");

}

}

b.Append("]\n");

}

Console.WriteLine("ZY");

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

{

int k = 0;

b.Append("[");

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

{

//Добавление разделителя-табуляции

if (i > 0)

{

b.Append("\t");

}

//Если текущий элемент не пустой

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("]\n");

}

return b.ToString();

}

}

**Rectangle.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

/// <summary>

/// Класс прямоугольник, наследник класса фигура

/// </summary>

class Rectangle : GeomFigure, IPrint

{

public override double Areaa()

{

return Width \* Length;

}

protected double Width;

protected double Length;

protected Rectangle()

{

Area = Width \* Length;

}

/// <summary>

/// Создание фигуры

/// </summary>

/// <param name="W">Ширина фигуры</param>

/// <param name="L">Длина фигуры</param>

public Rectangle(double W, double L)

{

Width = W;

Length = L;

Area = Width \* Length;

Name = "Прямоугольник";

Nameles = "Прямоугольник";

}

public Rectangle(double W, double L,string s)

{

Width = W;

Length = L;

Area = Width \* Length;

Name = "Прямоугольник";

Nameles = "Прямоугольник "+s;

}

public void Display()

{

cout($"Ширина фигуры {Name} равна:{Width}");

cout($"Длина фигуры {Name} равна:{Length}");

cout($"Площадь фигуры {Name} равна:{Area}");

}

public override string ToString()

{

if (Name == "Прямоугольник") return Name.ToString() + " Ширина: " + Width.ToString() + " Длина: " + Length.ToString() + " Площадь фигуры: " + Area.ToString();

else return "";

}

public void Print()

{

Console.WriteLine(this.ToString());

}

}

}

**SimpleListItem.cs:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

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

{

//Присоединение элемента к цепочке

this.last.next = newItem;

//Присоединенный элемент считается последним

this.last = newItem;

}

}

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;

}

public T Get(int number)

{

return GetItem(number).data;

}

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;

}

}

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;

}

}

}

**SimplseStack:**

using System;

using System.Collections.Generic;

using System.Text;

namespace LabRabota\_3

{

public 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;

}

}

}

**Экранные формы с примерами выполнения программы.**



