**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н.Э.БАУМАНА**

студента 2-го курса

РТ факультета группы РТ5-31

Макарова Алексея

Москва – 2016г.

Лабораторная работа №1

**Цель работы:** разработать программу для решения квадратного уравнения.

**Описание программы:**

Программа, разработанная в виде консольного приложения на языке С#, осуществляет ввод с клавиатуры коэффициентов **А**, **В**, **С** , вычисляет дискриминант по формуле и корни уравнения (в зависимости от дискриминанта). При некорректном вводе коэффициентов или дискриминанте, равном нулю, программа выдаёт ошибку и предлагает ввести коэффициенты заново.

**Текст программы (язык С#):**

using System; //вскрытие библиотек

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Kvadratnoe\_Makarov\_A\_RT\_5\_31//начало программы

{

class Program

{

static void Main(string[] args) //объявление главной части

{

//ax^2+bx+с

//D=b^2-4ac

//double a, b, c;

Console.WriteLine("Введите a: "); //вывод запроса в ввод а

double a = InputDouble(); //ввод даных

//a = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Введите b: "); //вывод запроса в ввод b

double b = InputDouble(); //ввод даных

//b = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Введите c: "); //вывод запроса в ввод c

double c = InputDouble(); //ввод даных

// c = Convert.ToDouble(Console.ReadLine());

Reshenie myclass = new Reshenie(a, b, c); //классифицируем

myclass.raschet(); //выполняем классную функцию

Console.ReadLine(); //остановка консоли для просмотра результатов

}

private static double InputDouble() //функция ввода чисел

{

double number; //допустим введем число

string value; //и некоторую строковую данную, чтоб ее преобразовать

do //поехали

{

value = Console.ReadLine(); //вводим строку

if (Double.TryParse(value, out number)) return number; //если строка число, то возвращаем это число

Console.WriteLine(" Данные не являются вещественным числом! ");//иначе выводим предупреждение

} while (true);

}

}

class Reshenie

{

private double a, b, c, D, x1, x2; //вводим частные дублированные данные

public Reshenie(double a, double b, double c) //вводим публичные данные

{

this.a = a;

this.b = b;

this.c = c;

}

public void raschet() //само решение уравнения

{

D = Math.Pow(b, 2) - 4 \* a \* c; //дискриминант

if (D > 0 || D == 0) //если д не отрицателен

{

x1 = (-b + Math.Sqrt(D)) / (2 \* a); //ур-е х1

x2 = (-b - Math.Sqrt(D)) / (2 \* a); //ур-е х2

Console.WriteLine("D={0}", D); //вывод дискриминанта

Console.WriteLine("x1= {0}\n x2= {1}", x1, x2);//вывод решений

Console.ReadKey();

}

else

{

Console.WriteLine("Действительных корней нет"); //если нет корней

Console.ReadKey();

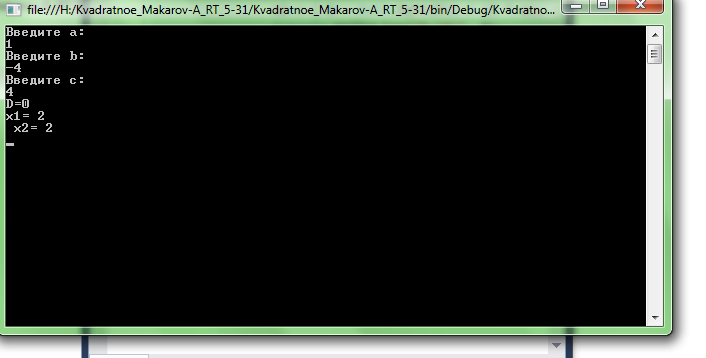
}

}

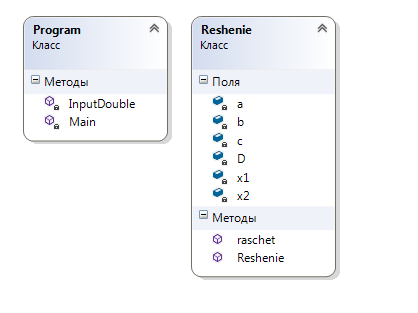
}

}

**Результат работы программы**



**Дерево классов.**



Лабораторная работа №2

**Цель работы:** Разработать программу, реализующую работу с классами.

**Описание программы:**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Абстрактный класс «Геометрическая фигура» содержит виртуальный метод для вычисления площади фигуры.
3. Класс «Прямоугольник» наследуется от «Геометрическая фигура». Ширина и высота объявляются как свойства (property). Класс должен содержать конструктор по параметрам «ширина» и «высота».
4. Класс «Квадрат» наследуется от «Прямоугольник». Класс должен содержать конструктор по длине стороны.
5. Класс «Круг» наследуется от «Геометрическая фигура». Радиус объявляется как свойство (property). Класс должен содержать конструктор по параметру «радиус».
6. Для классов «Прямоугольник», «Квадрат», «Круг» переопределить виртуальный метод Object.ToString(), который возвращает в виде строки основные параметры фигуры и ее площадь.
7. Разработать интерфейс IPrint. Интерфейс содержит метод Print(), который не принимает параметров и возвращает void. Для классов «Прямоугольник», «Квадрат», «Круг» реализовать наследование от интерфейса IPrint. Переопределяемый метод Print() выводит на консоль информацию, возвращаемую переопределенным методом ToString().

**Текст программы (язык С#):**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApplication17

{/// <summary>

/// класс - геометрическая фигура

/// </summary>

class gemeticheskayfigura

{

public double a, b;

public gemeticheskayfigura(double a, double b)

{

this.a = a;

this.b = b;

}

}

/// <summary>

/// класс круг, наследник геометрической Фигуры

/// </summary>

class krug : gemeticheskayfigura

{

public krug(double a, double b)

: base(a, b)

{ }

public void raschet(krug qw)

{

qw.a = (a\*a) \* 3.14;

Console.WriteLine("Площадь круга: {0}", a);

}

}

/// <summary>

/// класс прямоугольник наследник геометрической фигуры

/// </summary>

class pramougolnik : gemeticheskayfigura

{

public pramougolnik(double a, double b)

: base(a, b)

{ }

public void raschet(pramougolnik qw)

{

qw.a = (a \* b);

Console.WriteLine("Площадь прямоугольника: {0}", a);

}

}

/// <summary>

/// класс квадрата, наследник прямоугольника

/// </summary>

class kvadrat : pramougolnik

{

public kvadrat(double a,double b)

: base(a,b)

{ }

public void raschet(kvadrat qw)

{

qw.a = (a \* a);

Console.WriteLine("Площадь квадрата: {0}", a);

}

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Выберете фигуру, введя цифру цифре:прямоугольник- 1, квадрат- 2, круг- 3");

double c = InputDouble();

Console.WriteLine("Введите данные-длину и ширину прямоугольника, сторону квадрата,радиус круга (^-^)");

if (c == 1)

{

pramougolnik qw = new pramougolnik(InputDouble(), InputDouble());

qw.raschet(qw);

}

if (c == 2)

{

kvadrat qw = new kvadrat(InputDouble(),1);

qw.raschet(qw);

}

if (c == 3) {

krug qw = new krug(InputDouble(), 1);

qw.raschet(qw);

}

Console.ReadLine();

}

/// <summary>

/// метод ввода чисел

/// </summary>

private static double InputDouble() //метод ввода чисел

{

double number; //допустим введем число

string value; //и некоторую строковую данную, чтоб ее преобразовать

do //поехали

{

value = Console.ReadLine(); //вводим строку

if (Double.TryParse(value, out number)) return number; //если строка число, то возвращаем это число

Console.WriteLine(" Данные не являются вещественным числом! ");//иначе выводим предупреждение

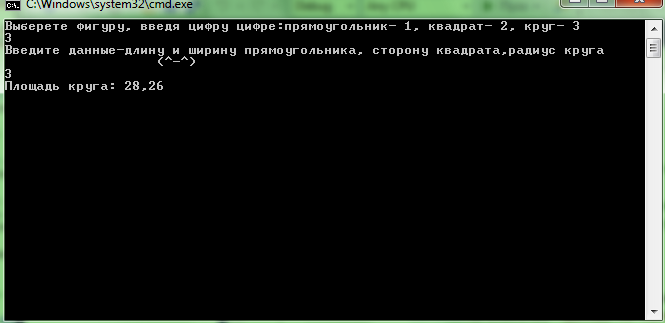
} while (true);

}

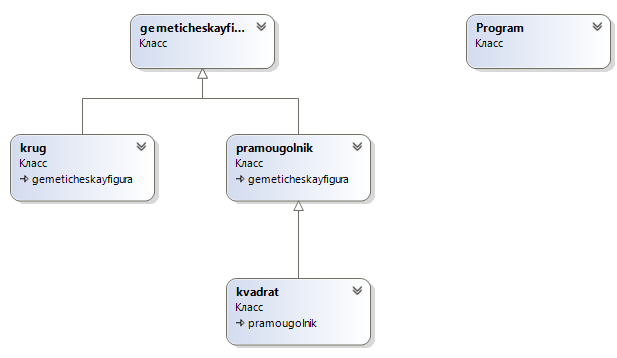
}

}

**Результат работы программы**



**Дерево классов.**



Лабораторная работа №3

**Цель работы:** Разработать программу, реализующую работу с коллекциями.

**Описание программы:**

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 (язык С#):**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using SimpleListProject;

namespace Collections

{

/// <summary>

/// класс - геометрическая фигура

/// </summary>

class gemeticheskayfigura

{

public double a, b;

public gemeticheskayfigura(double a, double b)

{

this.a = a;

this.b = b;

}

}

/// <summary>

/// класс круг, наследник геометрической Фигуры

/// </summary>

class krug : gemeticheskayfigura

{

public krug(double a, double b)

: base(a, b)

{ }

public double raschet(krug qr)

{

return (Math.Pow(a,2) \* Math.PI);

}

}

/// <summary>

/// класс прямоугольник наследник геометрической фигуры

/// </summary>

class pramougolnik : gemeticheskayfigura

{

public pramougolnik(double a, double b)

: base(a, b)

{ }

public double raschet(pramougolnik qw)

{

return (a \* b);

}

}

/// <summary>

/// класс квадрата, наследник прямоугольника

/// </summary>

class kvadrat : pramougolnik

{

public kvadrat(double a, double b)

: base(a, b)

{ }

public double raschet(kvadrat qe)

{

return (a \* a);

}

}

class MyCollection

{

public static void Writingcollection(ArrayList list)// вывод коллекциии цикл

{

foreach (double a in list)

Console.Write("{0}\t", a);

Console.WriteLine("\n");

}

}

class Program

{

static void Main(string[] args)

{

ArrayList list = new ArrayList();//коллекция эррэй

Console.WriteLine("Введите данные-длину и ширину прямоугольника, сторону квадрата,радиус круга");

pramougolnik qw = new pramougolnik(InputDouble(), InputDouble());

list.Add(qw.raschet(qw)); //добавляем прямоугольник

kvadrat qe = new kvadrat(InputDouble(), 1);

list.Add(qe.raschet(qe)); //квадрат

krug qr = new krug(InputDouble(), 1);

list.Add(qr.raschet(qr)); //круг

foreach (object o in list) //вывод обычный

{

Console.WriteLine("{0}\t", o);

}

Console.WriteLine("\n");

list.Sort(); //сортируем

MyCollection.Writingcollection(list);

List<double> collections2 = new List<double>() { qw.raschet(qw), qe.raschet(qe), qr.raschet(qr) };//коллекция лист

collections2.Sort();

foreach (double i in collections2)

{

Console.WriteLine(i);

}

Console.ReadLine(); }

/// <summary>

/// метод ввода чисел

/// </summary>

private static double InputDouble() //метод ввода чисел

{

double number; //допустим введем число

string value; //и некоторую строковую данную, чтоб ее преобразовать

do //поехали

{

value = Console.ReadLine(); //вводим строку

if (Double.TryParse(value, out number)) return number; //если строка число, то возвращаем это число

Console.WriteLine(" Данные не являются вещественным числом! ");//иначе выводим предупреждение

} while (true);

}

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;

int maxZ;

/// <summary>

/// Пустой элемент, который возвращается если элемент с нужными координатами не был задан

/// </summary>

T nullElement;

/// <summary>

/// Конструктор

/// </summary>

public Matrix(int px, int py, int pz, 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 (this.\_matrix.ContainsKey(key))

{

return this.\_matrix[key];

}

else

{

return this.nullElement;

}

}

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

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

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");

}

/// <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 склеиванием

//большого количества обычных строк

StringBuilder b = new StringBuilder();

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

{

b.Append("[");

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

{

b.Append("[");

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

{

if (i > 0) b.Append("\t");

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

}

b.Append("]\n");

}

b.Append("]\n");

}

return b.ToString();

}

}

}

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

{

/// <summary>

/// Добавление в стэк

/// </summary>

/// <param name="element"></param>

public void Push(T element)

{

//Добавление в конец списка уже реализовано

Add(element);

}

/// <summary>

/// Удаление и чтение из стека

/// </summary>

public T Pop()

{

//default(T) - значение для типа T по умолчанию

T Result = default(T);

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

//Если элемент единственный

if (this.Count == 1)

{

//то из него читаются данные

Result = this.first.data;

this.first = null;

this.last = null;

}

else

{

//В списке более одного элемента

//Поиск предпоследнего элемента

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

//Чтение значения из последнего элемента

Result = newLast.next.data;

//предпоследний элемент считается последним

this.last = newLast;

//последний элемент удаляется из списка

newLast.next = null;

}

//Уменьшение количества элементов в списке

this.Count--;

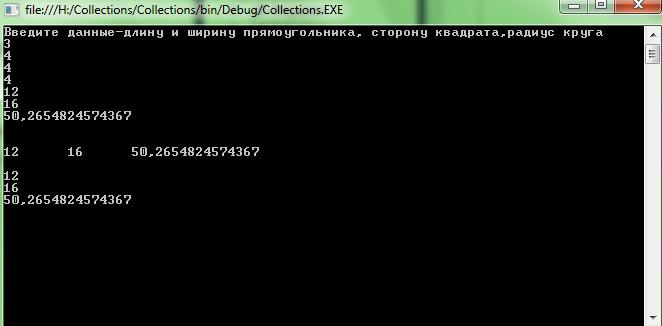
return Result;

}

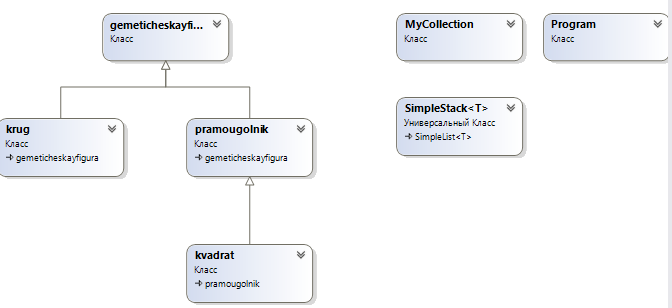
}

}

**Результат работы программы.**

****

**Дерево классов.**



Лабораторная работа №4,5,8

**Цель работы:** -Разработать программу, реализующую работу с файлами.

- Разработать программу, реализующую вычисление расстояния Левенштейна с использованием алгоритма Вагнера-Фишера.

- Разработать программу, реализующую многопоточный поиск в файле.

**Описание программы:**

-

1. Программа должна быть разработана в виде библиотеки классов на языке C#.
2. Использовать самый простой вариант алгоритма без оптимизации.
3. Дополнительно возможно реализовать вычисление расстояния Дамерау-Левенштейна (с учетом перестановок соседних символов).
4. Модифицировать предыдущую лабораторную работу, вместо поиска подстроки используется вычисление расстояния Левенштейна.
5. Предусмотреть отдельное поле ввода для максимального расстояния. Если расстояние Левенштейна между двумя строками больше максимального, то строки считаются несовпадающими и не выводятся в список результатов.

-

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   * метод, разработанный в пункте 3;
   * лямбда-выражение.
5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

-

1. Программа должна быть разработана в виде приложения Windows Forms на языке C#. По желанию вместо Windows Forms возможно использование WPF.
2. В качестве основы используется макет, разработанный в лабораторных работах №4 и №5.
3. Реализуйте функцию поиска с использованием расстояния Левенштейна в многопоточном варианте. Количество потоков для запуска функции поиска вводится на форме в поле ввода (TextBox).
4. Реализуйте функцию записи результатов поиска в файл отчета. Файл отчета создается в формате .txt или .html.

**Текст программы (язык С#):**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.IO;

using System.Diagnostics;

using EditDistanceProject;

using System.Threading.Tasks;

namespace WindowsFormsFiles

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Список слов

/// </summary>

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

private void buttonLoadFile\_Click(object sender, EventArgs e)

{

OpenFileDialog fd = new OpenFileDialog();

fd.Filter = "текстовые файлы|\*.txt";

if (fd.ShowDialog() == DialogResult.OK)

{

Stopwatch t = new Stopwatch();

t.Start();

//Чтение файла в виде строки

string text = File.ReadAllText(fd.FileName);

//Разделительные символы для чтения из файла

char[] separators = new char[] {' ','.',',','!','?','/','\t','\n'};

string[] textArray = text.Split(separators);

foreach (string strTemp in textArray)

{

//Удаление пробелов в начале и конце строки

string str = strTemp.Trim();

//Добавление строки в список, если строка не содержится в списке

if (!list.Contains(str)) list.Add(str);

}

t.Stop();

this.textBoxFileReadTime.Text = t.Elapsed.ToString();

this.textBoxFileReadCount.Text = list.Count.ToString();

}

else

{

MessageBox.Show("Необходимо выбрать файл");

}

}

private void buttonExact\_Click(object sender, EventArgs e)

{

//Слово для поиска

string word = this.textBoxFind.Text.Trim();

//Если слово для поиска не пусто

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)

{

//Слово для поиска в верхнем регистре

string wordUpper = word.ToUpper();

//Временные результаты поиска

List<string> tempList = new List<string>();

Stopwatch t = new Stopwatch();

t.Start();

foreach (string str in list)

{

if (str.ToUpper().Contains(wordUpper))

{

tempList.Add(str);

}

}

t.Stop();

this.textBoxExactTime.Text = t.Elapsed.ToString();

this.listBoxResult.BeginUpdate();

//Очистка списка

this.listBoxResult.Items.Clear();

//Вывод результатов поиска

foreach (string str in tempList)

{

this.listBoxResult.Items.Add(str);

}

this.listBoxResult.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

private void buttonApprox\_Click(object sender, EventArgs e)

{

//Слово для поиска

string word = this.textBoxFind.Text.Trim();

//Если слово для поиска не пусто

if (!string.IsNullOrWhiteSpace(word) && list.Count > 0)

{

int maxDist;

if(!int.TryParse(this.textBoxMaxDist.Text.Trim(), out maxDist))

{

MessageBox.Show("Необходимо указать максимальное расстояние");

return;

}

if (maxDist < 1 || maxDist > 5)

{

MessageBox.Show("Максимальное расстояние должно быть в диапазоне от 1 до 5");

return;

}

int ThreadCount;

if (!int.TryParse(this.textBoxThreadCount.Text.Trim(), out ThreadCount))

{

MessageBox.Show("Необходимо указать количество потоков");

return;

}

Stopwatch timer = new Stopwatch();

timer.Start();

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

// Начало параллельного поиска

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

//Результирующий список

List<ParallelSearchResult> Result = new List<ParallelSearchResult>();

//Деление списка на фрагменты для параллельного запуска в потоках

List<MinMax> arrayDivList = SubArrays.DivideSubArrays(0, list.Count, ThreadCount);

int count = arrayDivList.Count;

//Количество потоков соответствует количеству фрагментов массива

Task<List<ParallelSearchResult>>[] tasks = new Task<List<ParallelSearchResult>>[count];

//Запуск потоков

for (int i = 0; i < count; i++)

{

//Создание временного списка, чтобы потоки не работали параллельно с одной коллекцией

List<string> tempTaskList = list.GetRange(arrayDivList[i].Min, arrayDivList[i].Max - arrayDivList[i].Min);

tasks[i] = new Task<List<ParallelSearchResult>>(

//Метод, который будет выполняться в потоке

ArrayThreadTask,

//Параметры потока

new ParallelSearchThreadParam()

{

tempList = tempTaskList,

maxDist = maxDist,

ThreadNum = i,

wordPattern = word

});

//Запуск потока

tasks[i].Start();

}

Task.WaitAll(tasks);

timer.Stop();

//Объединение результатов

for (int i = 0; i < count; i++)

{

Result.AddRange(tasks[i].Result);

}

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

// Завершение параллельного поиска

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

timer.Stop();

//Вывод результатов

//Время поиска

this.textBoxApproxTime.Text = timer.Elapsed.ToString();

//Вычисленное количество потоков

this.textBoxThreadCountAll.Text = count.ToString();

//Начало обновления списка результатов

this.listBoxResult.BeginUpdate();

//Очистка списка

this.listBoxResult.Items.Clear();

//Вывод результатов поиска

foreach (var x in Result)

{

string temp = x.word + "(расстояние=" + x.dist.ToString() + " поток=" + x.ThreadNum.ToString() + ")";

this.listBoxResult.Items.Add(temp);

}

//Окончание обновления списка результатов

this.listBoxResult.EndUpdate();

}

else

{

MessageBox.Show("Необходимо выбрать файл и ввести слово для поиска");

}

}

/// <summary>

/// Выполняется в параллельном потоке для поиска строк

/// </summary>

/// <param name="param"></param>

public static List<ParallelSearchResult> ArrayThreadTask(object paramObj)

{

ParallelSearchThreadParam param = (ParallelSearchThreadParam)paramObj;

//Слово для поиска в верхнем регистре

string wordUpper = param.wordPattern.Trim().ToUpper();

//Результаты поиска в одном потоке

List<ParallelSearchResult> Result = new List<ParallelSearchResult>();

//Перебор всех слов во временном списке данного потока

foreach (string str in param.tempList)

{

//Вычисление расстояния Дамерау-Левенштейна

int dist = EditDistance.Distance(str.ToUpper(), wordUpper);

//Если расстояние меньше порогового, то слово добавляется в результат

if (dist <= param.maxDist)

{

ParallelSearchResult temp = new ParallelSearchResult()

{

word = str,

dist = dist,

ThreadNum = param.ThreadNum

};

Result.Add(temp);

}

}

return Result;

}

private void buttonExit\_Click(object sender, EventArgs e)

{

this.Close();

//Application.Exit();

}

private void buttonSaveReport\_Click(object sender, EventArgs e)

{

//Имя файла отчета

string TempReportFileName = "Report\_" + DateTime.Now.ToString("dd\_MM\_yyyy\_hhmmss");

//Диалог сохранения файла отчета

SaveFileDialog fd = new SaveFileDialog();

fd.FileName = TempReportFileName;

fd.DefaultExt = ".html";

fd.Filter = "HTML Reports|\*.html";

if (fd.ShowDialog() == DialogResult.OK)

{

string ReportFileName = fd.FileName;

//Формирование отчета

StringBuilder b = new StringBuilder();

b.AppendLine("<html>");

b.AppendLine("<head>");

b.AppendLine("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'/>");

b.AppendLine("<title>" + "Отчет: " + ReportFileName + "</title>");

b.AppendLine("</head>");

b.AppendLine("<body>");

b.AppendLine("<h1>" + "Отчет: " + ReportFileName + "</h1>");

b.AppendLine("<table border='1'>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время чтения из файла</td>");

b.AppendLine("<td>" + this.textBoxFileReadTime.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Количество уникальных слов в файле</td>");

b.AppendLine("<td>" + this.textBoxFileReadCount.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Слово для поиска</td>");

b.AppendLine("<td>" + this.textBoxFind.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Максимальное расстояние для нечеткого поиска</td>");

b.AppendLine("<td>" + this.textBoxMaxDist.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время четкого поиска</td>");

b.AppendLine("<td>" + this.textBoxExactTime.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr>");

b.AppendLine("<td>Время нечеткого поиска</td>");

b.AppendLine("<td>" + this.textBoxApproxTime.Text + "</td>");

b.AppendLine("</tr>");

b.AppendLine("<tr valign='top'>");

b.AppendLine("<td>Результаты поиска</td>");

b.AppendLine("<td>");

b.AppendLine("<ul>");

foreach (var x in this.listBoxResult.Items)

{

b.AppendLine("<li>" + x.ToString() + "</li>");

}

b.AppendLine("</ul>");

b.AppendLine("</td>");

b.AppendLine("</tr>");

b.AppendLine("</table>");

b.AppendLine("</body>");

b.AppendLine("</html>");

//Сохранение файла

File.AppendAllText(ReportFileName, b.ToString());

MessageBox.Show("Отчет сформирован. Файл: " + ReportFileName);

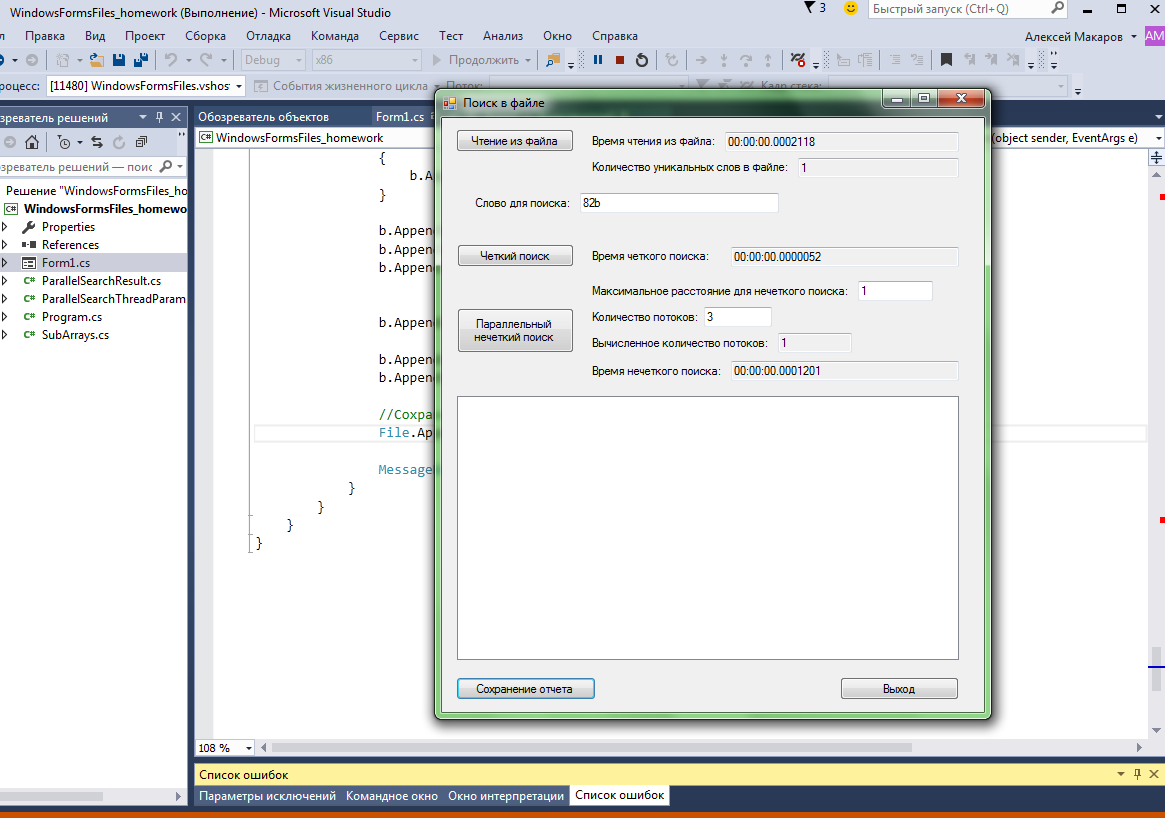
}

}

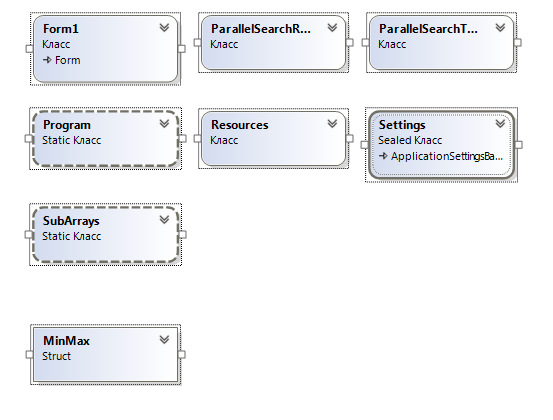
}

}

**Один из выводов программы**

****

**Дерево классов**

****

Лабораторная работа №6

**Цель работы:** **Часть 1. Разработать программу, использующую делегаты.**

**Часть 2. Разработать программу, реализующую работу с рефлексией.**

**Описание программы:**

**-**

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, в качестве одного из входным параметров. Осуществите вызов метода, передавая в качестве параметра-делегата:
   * метод, разработанный в пункте 3;
   * лямбда-выражение.
5. Повторите пункт 4, используя вместо разработанного Вами делегата, обобщенный делегат Func< > или Action< >, соответствующий сигнатуре разработанного Вами делегата.

-

1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Создайте класс, содержащий конструкторы, свойства, методы.
3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
4. Создайте класс атрибута (унаследован от класса System.Attribute).
5. Назначьте атрибут некоторым свойствам классам. Выведите только те свойства, которым назначен атрибут.
6. Вызовите один из методов класса с использованием рефлексии.

Программа, разработанная в виде консольного приложения на языке С#, осуществляет ввод с клавиатуры коэффициентов **А**, **В**, **С** , вычисляет дискриминант по формуле и корни уравнения (в зависимости от дискриминанта). При некорректном вводе коэффициентов или дискриминанте, равном нулю, программа выдаёт ошибку и предлагает ввести коэффициенты заново.

**Текст программы (язык С#):**

**-**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Delegates

{

//Делегаты - аналог процедурного типа в Паскале.

//Делегат - это не тип класса, а тип метода.

//Делегат определяет сигнатуру метода (типы параметров и возвращаемого значения).

//Если создается метод типа делегата, то у него должна быть сигнатура как у делегата.

//Метод типа делегата можно передать как параметр другому методу.

//Название делегата при объявлении указывается "вместо" названия метода

delegate int PlusOrMinus(int p1, int p2);

class Program

{

//Методы, реализующие делегат (методы "типа" делегата)

static int Plus(int p1, int p2) { return p1 + p2; }

static int Minus(int p1, int p2) { return p1 - p2; }

/// <summary>

/// Использование обощенного делегата Func<>

/// </summary>

static void PlusOrMinusMethodFunc(string str, int i1, int i2, Func<int, int, int> PlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

// Func<int, string, bool> - делегат принимает параметры типа int и string и возвращает bool

// Если метод должен возвращать void, то используется делегат Action

// Action<int, string> - делегат принимает параметры типа int и string и возвращает void

// Action как правило используется для разработки групповых делегатов, которые используются в событиях

}

/// <summary>

/// Использование делегата

/// </summary>

static void PlusOrMinusMethod(string str, int i1, int i2, PlusOrMinus PlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2);

Console.WriteLine(str + Result.ToString());

}

static void Main(string[] args)

{

int i1 = 3;

int i2 = 2;

PlusOrMinusMethod("Плюс: ", i1, i2, Plus);

PlusOrMinusMethod("Минус: ", i1, i2, Minus);

//Создание экземпляра делегата на основе метода

PlusOrMinus pm1 = new PlusOrMinus(Plus);

PlusOrMinusMethod("Создание экземпляра делегата на основе метода: ", i1, i2, pm1);

//Создание экземпляра делегата на основе 'предположения' делегата

//Компилятор 'пердполагает' что метод Plus типа делегата

PlusOrMinus pm2 = Plus;

PlusOrMinusMethod("Создание экземпляра делегата на основе 'предположения' делегата: ", i1, i2, pm2);

//Создание анонимного метода

PlusOrMinus pm3 = delegate(int param1, int param2)

{

return param1 + param2;

};

PlusOrMinusMethod("Создание экземпляра делегата на основе анонимного метода: ", i1, i2, pm2);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 1: ", i1, i2,

(int x, int y) =>

{

int z = x + y;

return z;

}

);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 2: ", i1, i2,

(x, y) =>

{

return x+y;

}

);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения 3: ", i1, i2, (x, y) => x+y );

////////////////////////////////////////////////////////////////

Console.WriteLine("\n\nИспользование обощенного делегата Func<>");

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе метода: ", i1, i2, Plus);

string OuterString = "ВНЕШНЯЯ ПЕРЕМЕННАЯ";

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-выражения 1: ", i1, i2,

(int x, int y) =>

{

Console.WriteLine("Эта переменная объявлена вне лямбда-выражения: " + OuterString);

int z = x + y;

return z;

}

);

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-выражения 2: ", i1, i2,

(x, y) =>

{

return x + y;

}

);

PlusOrMinusMethodFunc("Создание экземпляра делегата на основе лямбда-выражения 3: ", i1, i2, (x, y) => x + y);

//////////////////////////////////////////////////////////////

//Групповой делегат всегда возвращает значение типа void

Console.WriteLine("Пример группового делегата");

Action<int, int> a1 = (x, y) => { Console.WriteLine("{0} + {1} = {2}", x, y, x + y); };

Action<int, int> a2 = (x, y) => { Console.WriteLine("{0} - {1} = {2}", x, y, x - y); };

Action<int, int> group = a1 + a2;

group(5, 3);

Action<int, int> group2 = a1;

Console.WriteLine("Добавление вызова метода к групповому делегату");

group2 += a2;

group2(10, 5);

Console.WriteLine("Удаление вызова метода из группового делегата");

group2 -= a1;

group2(20, 10);

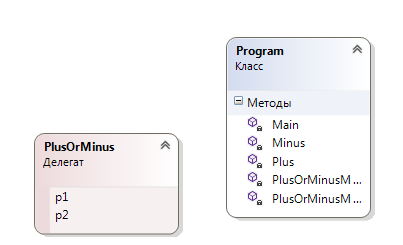
Console.ReadLine();

}

}

}

**Дерево классов.**

****

**-**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Reflection;

namespace Reflection

{

class Program

{

/// <summary>

/// Проверка, что у свойства есть атрибут заданного типа

/// </summary>

/// <returns>Значение атрибута</returns>

public static bool GetPropertyAttribute(PropertyInfo checkType, Type attributeType, out object attribute)

{

bool Result = false;

attribute = null;

//Поиск атрибутов с заданным типом

var isAttribute = checkType.GetCustomAttributes(attributeType, false);

if (isAttribute.Length > 0)

{

Result = true;

attribute = isAttribute[0];

}

return Result;

}

static void Main(string[] args)

{

Type t = typeof(ForInspection);

Console.WriteLine("Тип " + t.FullName + " унаследован от " + t.BaseType.FullName);

Console.WriteLine("Пространство имен " + t.Namespace);

Console.WriteLine("Находится в сборке " + t.AssemblyQualifiedName);

Console.WriteLine("\nКонструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("\nМетоды:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("\nПоля данных (public):");

foreach (var x in t.GetFields())

{

Console.WriteLine(x);

}

Console.WriteLine("\nСвойства, помеченные атрибутом:");

foreach (var x in t.GetProperties())

{

object attrObj;

if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))

{

NewAttribute attr = attrObj as NewAttribute;

Console.WriteLine(x.Name + " - " + attr.Description);

}

}

Console.WriteLine("\nВызов метода:");

//Создание объекта

//ForInspection fi = new ForInspection();

//Можно создать объект через рефлексию

ForInspection fi = (ForInspection)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] { });

//Параметры вызова метода

object[] parameters = new object[] { 3, 2 };

//Вызов метода

object Result = t.InvokeMember("Plus", BindingFlags.InvokeMethod, null, fi, parameters);

Console.WriteLine("Plus(3,2)={0}", Result);

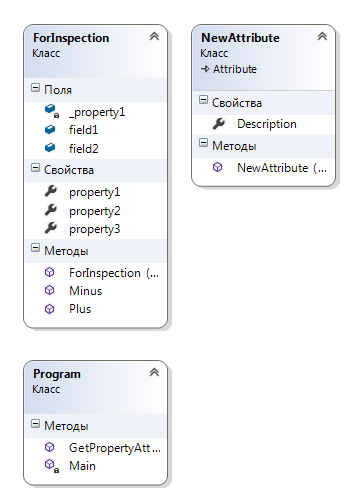
Console.ReadLine();

}

}

}

**Дерево классов.**

****