Отчёт по лабораторной работе №6 по курсу БКИТ

«Работа с делегатами и рефлексией»

Подготовил

Ионов С.А.

ИУ5-31Б

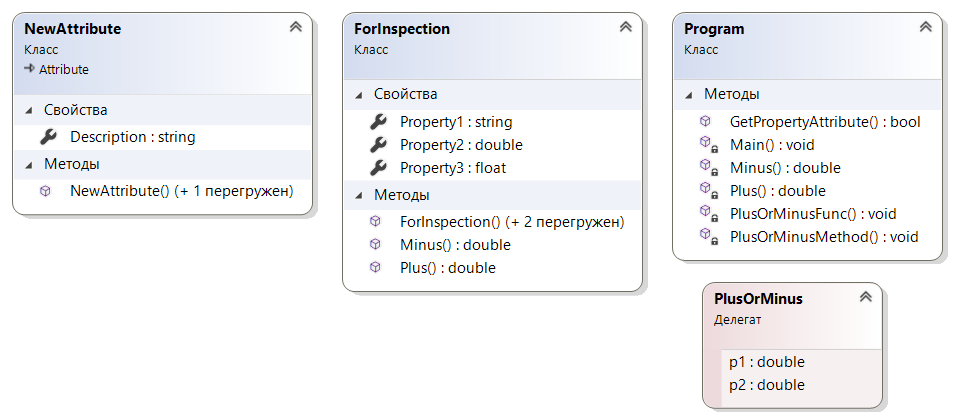
1. Описание задания

**Часть 1. Разработать программу, использующую делегаты.**

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 входных 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.

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

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.
7. Диаграмма классов



1. Текст программы

1)Program.cs

using System;

using System.Reflection;

namespace Lab6

{

delegate double PlusOrMinus(double p1, double p2);//Обычный делегат

class Program

{

static double Plus(double p1, double p2)//Метод типа делегата PlusOrMinus

{

return p1 + p2;

}

static double Minus(double p1, double p2)//Метод типа делегата PlusOrMinus

{

return p1 - p2;

}

//Метод, принимающий разработанный делегат PlusOrMinus

static void PlusOrMinusMethod(string str, double d1, double d2, PlusOrMinus PlusOrMinusParam)

{

double result = PlusOrMinusParam(d1, d2);

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

}

//Использования обобщенного делегата Func

static void PlusOrMinusFunc(string str, double d1, double d2, Func<double, double, double> PlusOrMinusParam)

{

double result = PlusOrMinusParam(d1, d2);

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

}

/// <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)

{

Console.WriteLine("\t\t1)Работа с делегатами");

double d1 = 1.1, d2 = 2;

Console.WriteLine("Числа: " + d1.ToString() + " и " + d2.ToString());

Console.WriteLine("\t1.Вызов метода с разработанным делегатом");

PlusOrMinusMethod("Сумма: ", 1.1, 2, Plus);

PlusOrMinus pm1 = Minus;//экземпляр делегата

PlusOrMinusMethod("Разность: ", 1.1, 2, pm1);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения\n", d1, d2, (x, y) => x + y);

Console.WriteLine("\t2.Вызов метода с обобщенным делегатом Func");

PlusOrMinusFunc("Сумма: ", 1.1, 2, Plus);

Func<double, double, double> pm2 = Minus;//экземпляр делегата

PlusOrMinusFunc("Разность: ", 1.1, 2, pm2);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения\n", d1, d2, (x, y) => x + y);

Console.WriteLine("\t\t2)Работа с рефлексией");

Console.WriteLine("\t1.Информация о конструкторах, свойствах и методах");

Type t = typeof(ForInspection);

Console.WriteLine("Конструкторы:");

foreach (var x in t.GetConstructors())

{

Console.WriteLine(x);

}

Console.WriteLine("Свойства:");

foreach (var x in t.GetProperties())

{

Console.WriteLine(x);

}

Console.WriteLine("Методы:");

foreach (var x in t.GetMethods())

{

Console.WriteLine(x);

}

Console.WriteLine("\t2.Свойства, 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("\t3.Вызов метода класса:");

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

object[] parameters = new object[] { 1.2, 3.5 };

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

Console.WriteLine("Plus(1.2,3.5)={0}", Result);

Console.ReadLine();

}

}

}

2)ForIspection.cs

namespace Lab6

{

public class ForInspection

{

public ForInspection() { }

public ForInspection(int i) { }

public ForInspection(string str) { }

public double Plus(double x, double y) { return x + y; }

public double Minus(double x, double y) { return x - y; }

[NewAttribute(Description = "Описание для Property1")]

public string Property1 { get; set; }

[NewAttribute(Description = "Описание для Property2")]

public double Property2 { get; private set; }

public float Property3 { get; private set; }

}

}

3)NewAttribute.cs

using System;

namespace Lab6

{

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

}

1. Экранные формы с примерами выполнения программы

