|  |  |
| --- | --- |
|  | **Министерство образования и науки 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)** |

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6**

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Мокренко Никита Валерьевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*фамилия, имя, отчество*

Группа: ИУ5 - 34

Название предмета**: Базовые компоненты интернет-технологий**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** \_\_\_\_Мокренко Н.В.\_\_\_\_

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Гапанюк Ю.Е.**\_\_\_\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2017 г.*

**Задание**

**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 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.

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

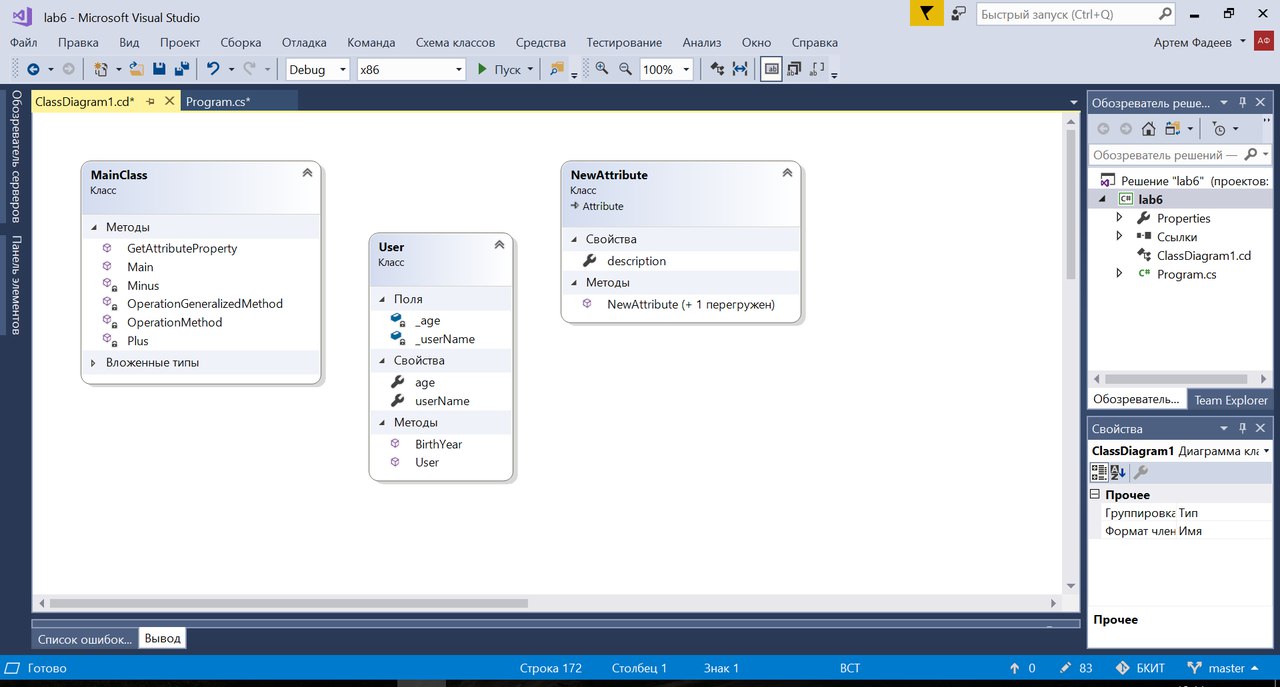
* + 1. Программа должна быть разработана в виде консольного приложения на языке C#.
  + 2. Создайте класс, содержащий конструкторы, свойства, методы.
  + 3. С использованием рефлексии выведите информацию о конструкторах, свойствах, методах.
  + 4. Создайте класс атрибута (унаследован от класса System.Attribute).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

**Код программы**

Program.cs

|  |
| --- |
| using System; |
|  | using System.Reflection; |
|  |  |
|  | namespace lab6 |
|  | { |
|  | class MainClass |
|  | { |
|  | //Объявление делегата |
|  | delegate double Operation(int a, double b); |
|  |  |
|  | //Методы, соответствующие делегату |
|  | static double Plus (int a, double b) |
|  | { |
|  | return a + b; |
|  | } |
|  |  |
|  | static double Minus(int a, double b) |
|  | { |
|  | return a - b; |
|  | } |
|  |  |
|  | //Метод с параметром в виде делегата |
|  | static void OperationMethod(string str, int a, double b, Operation OperationParam) |
|  | { |
|  | double result = OperationParam(a, b); |
|  | Console.WriteLine(str + result.ToString()); |
|  | } |
|  |  |
|  | //Метод с параметром в виде обобщенного делегата |
|  | static void OperationGeneralizedMethod(string str, int a, double b, |
|  | Func <int, double, double> OperationParam) |
|  | { |
|  | double result = OperationParam(a, b); |
|  | Console.WriteLine(str + result.ToString()); |
|  | } |
|  |  |
|  | // Проверка, что у свойства есть атрибут заданного типа |
|  | //Параметры - проверяемое свойство, тип проверяемого атрибута |
|  | public static bool GetAttributeProperty(PropertyInfo checkType, Type attributeType, |
|  | out object attribute) |
|  | { |
|  | attribute = null; |
|  |  |
|  | //Поиск атрибутов с заданным типом |
|  | var isAttribute = checkType.GetCustomAttributes(attributeType, false); |
|  | if (isAttribute.Length > 0) |
|  | { |
|  | attribute = isAttribute[0]; |
|  | return true; |
|  | } |
|  |  |
|  | return false; |
|  | } |
|  |  |
|  | public static void Main(string[] args) |
|  | { |
|  | Console.WriteLine("=========================================================="); |
|  | Console.WriteLine("1.4) Простой вызов:"); |
|  | Console.WriteLine("С помощью метода:"); |
|  | OperationMethod("Сумма: ", 5, 2.1, Plus); |
|  | Console.WriteLine("С помощью лямбда-выражения:"); |
|  | OperationMethod("Произведение: ", 5, 2.1, (a, b) => { return a \* b; }); |
|  | Console.WriteLine("1.5) С использованием обобщенного делегата 'Func <...>':"); |
|  | Console.WriteLine("С помощью метода:"); |
|  | OperationGeneralizedMethod("Разность: ", 5, 2.1, Minus); |
|  | Console.WriteLine("С помощью лямбда-выражения:"); |
|  | OperationGeneralizedMethod("Частное: ", 5, 2.1, (a, b) => { return a / b; }); |
|  | Console.WriteLine("=========================================================="); |
|  | User person = new User("Артём", 19); |
|  | Type t = person.GetType(); |
|  | 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); |
|  | } |
|  | Type type = typeof(User); |
|  | Console.WriteLine(("Свойства помеченные аттрибутом:")); |
|  | //Перебор всех свойств атрибутов |
|  | //Если свойство снабжено атрибутом, |
|  | //то выводится название свойства и поле Description атрибута |
|  | foreach (var x in t.GetProperties()) |
|  | { |
|  | Object attrObj; |
|  | if (GetAttributeProperty(x, typeof(NewAttribute), out attrObj)) |
|  | { |
|  | NewAttribute attr = attrObj as NewAttribute; |
|  | Console.WriteLine(x.Name + " - " + attr.description); |
|  | } |
|  | } |
|  |  |
|  | //Метод InvokeMember класса Type позволяет выполнять |
|  | //динамические действия с объектами классов: |
|  | //создавать объекты, вызывать методы, получать и присваивать значения свойств и др. |
|  | //Особенность в том, что он передает параметры в виде строк (имена свойств, классов) |
|  | Console.WriteLine(("Вызов метода с использованием рефлексии:")); |
|  | Console.WriteLine(("Cоздан объект Человек с параметрами {Артём, 19}")); |
|  |  |
|  | //Создание объекта |
|  | object[] parameters = new object[] { "Артём", 19 }; |
|  |  |
|  | //Создание объекта через рефлексию |
|  | User user = (User)type.InvokeMember |
|  | (null, BindingFlags.CreateInstance, null, null, parameters); |
|  |  |
|  | //Вызов метода без параметров |
|  | object result = type.InvokeMember |
|  | ("BirthYear", BindingFlags.InvokeMethod, null, user, null); |
|  |  |
|  | //BindingFlags – это перечисление, которое определяет выполняемое действие: |
|  | //создание объекта, обращение к методу или свойству и т.д. |
|  | Console.WriteLine("Год рождения:" + result.ToString()); |
|  | Console.WriteLine("##########################################################\n\n" + |
|  | "Нажмите любую клавишу для выхода из программы...\n"); |
|  | Console.Read(); |
|  | } |
|  | } |
|  |  |
|  | //Класс атрибута |
|  | [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)] |
|  | public class NewAttribute : Attribute |
|  | { |
|  | //Создание нового атрибута (и его описания) на основе стандартного класса |
|  | public NewAttribute() { } |
|  | public NewAttribute(string descriptionParam) |
|  | { |
|  | description = descriptionParam; |
|  | } |
|  | public string description { get; set; } |
|  | } |
|  |  |
|  |  |
|  | public class User |
|  | { |
|  | private string \_userName; |
|  |  |
|  | private int \_age; |
|  |  |
|  | [NewAttribute("Username пользователя")] |
|  | public string userName |
|  | { |
|  | get { return \_userName; } |
|  | private set { \_userName = value; } |
|  | } |
|  |  |
|  | public int age |
|  | { |
|  | get { return \_age; } |
|  | private set { \_age = value; } |
|  | } |
|  |  |
|  | public User(string name, int number) { |
|  | userName = name + age.ToString(); |
|  | age = number; |
|  | } |
|  |  |
|  | public int BirthYear() |
|  | { |
|  | return 2017 - age; |
|  | } |
|  | } |
|  | } |

**Диаграмма классов**



**Примеры работы**

