**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-35б |  | преподаватель каф. ИУ5 |
| Хижняков Вадим |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

# Задание.

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

(В качестве примера можно использовать проект «Delegates»).

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.

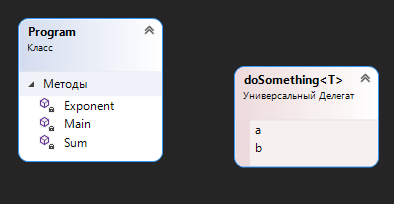
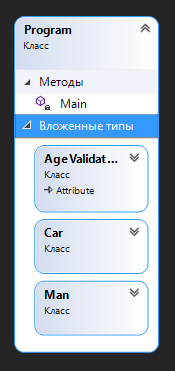
**Часть 2. Разработать программу, реализующую работу с рефлексией.**

(В качестве примера можно использовать проект «Reflection»).

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.

# Диаграмма классов

Delegates:

Reflection:

# Текст программы

**Delegates:**

Program.cs

using System;

using System.Runtime.InteropServices.ComTypes;

namespace Sem3Lab6\_Delegates

{

delegate T doSomething<T>(int a, double b);

class Program

{

static double Sum(int a, double b) { return a + b; }

static double Exponent(int a, double b, doSomething<double> del){ return Math.Pow(Math.E, del(a, b)); }

static void Main(string[] args)

{

doSomething<double> del = Sum;

Console.WriteLine(del?.Invoke(3, 2.5));

Console.WriteLine(Exponent(1, 4.2, Sum));

Console.WriteLine(Exponent(1, 4.2, del));

Console.WriteLine(Exponent(1, 4.2, (a, b) => a + b));//если параметры передаются через ref/out обязательно указывать тип

Console.WriteLine(Exponent(1, 4.2, (a, b) => Sum(a, b)));

Func<int, double, double> func = Sum;

Console.WriteLine(Exponent(1, 4.2, func.Invoke));

}

}

}

**Reflection:**

Program.cs

using System;

using System.Reflection;

namespace Sem3Lab6\_Reflection

{

public class Program

{

public class AgeValidationAttribute : System.Attribute

{

public int Age { get; set; }

public AgeValidationAttribute()

{

}

public AgeValidationAttribute(int age)

{

this.Age = age;

}

}

[AgeValidation(18)]

public class Man

{

public string Name { get; set; }

public uint Age { get; set; }

public double Money { get; set; }

public Man(string name, uint age, double money)

{

this.Age = age;

this.Name = name;

this.Money = money;

}

}

public class Car

{

public string model { get; private set; }

public double price { get; private set; }

public string bodyType { get; private set; }

public int power { get; private set; }

public double engineCapacity { get; private set; }

public static uint count { get; private set; }

public Car(string model, double price, string bodyType, int power, double engineCapacity)

{

this.model = model;

this.price = price; ;

this.bodyType = bodyType;

this.power = power;

this.engineCapacity = engineCapacity;

count = 10;

}

static bool validateCustomer(Man customer)

{

Type t = customer.GetType();

var attributes = t.GetCustomAttributes(typeof(AgeValidationAttribute), false);

foreach (AgeValidationAttribute attribute in attributes)

{

return customer.Age >= attribute.Age;

}

return true;

}

public bool Buy(Man customer)

{

if (count <= 0)

return false;

if (validateCustomer(customer) && customer.Money >= this.price)

{

customer.Money -= this.price;

count--;

return true;

}

return false;

}

public void Drive()

{

Console.WriteLine("Wroooooom wrooooooom I'm Prado with gas");

}

public void Display()

{

Console.WriteLine($"Model: {this.model} | Price: {this.price} \n" +

$" Body Type: {this.bodyType} | Power: {this.power} \n" +

$" Engine Capacity: {this.engineCapacity}");

}

}

static void Main(string[] args)

{

try

{

Type classCar = typeof(Car);

//var classCar = Type.GetType("Sem3Lab6\_Reflection.Car");

//shortcut for written later

/\*foreach (var item in classCar.GetMembers())

{

Console.WriteLine($"{item.DeclaringType} {item.MemberType} {item.Name}");

}\*/

Console.WriteLine("Fields:");

foreach (FieldInfo field in classCar.GetFields())

{

Console.WriteLine($"{field.FieldType} {field.Name}");

}

Console.WriteLine("Properties:");

foreach (PropertyInfo prop in classCar.GetProperties())

{

Console.WriteLine($"{prop.PropertyType} {prop.Name}");

}

Console.WriteLine("Methods:");

foreach (MethodInfo method in classCar.GetMethods())

{

string modificator = string.Empty;

if (method.IsStatic)

modificator += " static";

if (method.IsVirtual)

modificator += " virtual";

Console.Write($"{modificator} {method.ReturnType.Name} {method.Name} (");

var parameters = method.GetParameters();

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

{

Console.Write($"{parameters[i].ParameterType.Name} {parameters[i].Name}");

if (i + 1 < parameters.Length) Console.Write(", ");

}

Console.WriteLine(")");

}

}

catch (Exception ex)

{

throw ex;

}

Car lancer = new Car("Lancer X", 300000, "Sedan", 150, 1.8);

Man vadik = new Man("Vadik", 19, 10000000000);

Car jija = new Car("Zhiguli", 16000, "таз", 77, 1.8);

Man max = new Man("Maxim", 16, 40000);

Console.WriteLine("----------------------------------------\nVadim buying Lancer X: " + lancer.Buy(vadik));

Console.WriteLine("Max buying ZHIGA: " + jija.Buy(max));

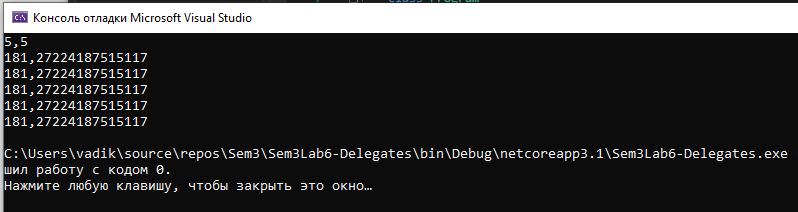
}

}

}

# Результаты

Delegates:



Reflection:

