**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Радиотехнический»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет технологий»

Отчет по лабораторной работе №6

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы РТ5-31Б |  | преподаватель каф. ИУ5 |
| Грызин Алексей |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

# Описание задания

**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

# 

# Текст программы

public Car() { }

public Car(string Name) { this.Name = Name; }

public double GetFuelConsumption(double liters, double kilometers)

{

return liters / kilometers \* 100;

}

public string Name

{

get { return Name; }

set { Name = value; }

}

[NewAttribute("Цена")]

public double Cost

{

get { return Cost; }

set { Cost = value; }

}

[NewAttribute("Тип кузова")]

public string Body

{

get { return Body; }

set { Body = value; }

}

[NewAttribute("Год выпуска")]

public int Date

{

get { return Date; }

set { Date = value; }

}

public void GetInfo()

{

Console.WriteLine("Название: {0}\nДата выпуска: {1}\n Кузов: {2}\nЦена: {3} рублей",

this.Name == null ? "-" : this.Name,

this.Date == null ? 0 : this.Date,

this.Cost == null ? 0 : this.Cost,

this.Body == null ? "-" : this.Body);

}

public int CompareTo(object obj)

{

return 0;

}

}

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

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam)

{

Description = DescriptionParam;

}

public string Description { get; set; }

}

//Делегат сравнения

delegate string Comparing(double x, double y);

partial class Program

{

//1-я часть

//Методы соответстующие делегату

static string Bigger(double x, double y)

{

if (x > y) return $"{x} больше, чем {y}";

else if (y > x) return $"{y} больше, чем {x}";

else return $"{x} и {y} равны";

}

static string Lower(double x, double y)

{

if (x < y) return $"{x} меньше, чем {y}";

else if (y < x) return $"{y} меньше, чем {x}";

else return $"{x} и {y} равны";

}

//Метод с делегатным параметром

static void ComparingMethod(double x, double y, Comparing parameter)

{

string result = parameter(x, y);

Console.WriteLine(result);

}

//Метод с обобщенным делегатом

static void ComparingMethodFunc(double x, double y, Func<double, double, string> parameter)

{

string result = parameter(x, y);

Console.WriteLine(result);

}

//2-я часть

// Проверка, что у свойства есть атрибут заданного типа

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;

}

// Получение информации о текущей сборке

static void AssemblyInfo()

{

Console.WriteLine("Вывод информации о сборке:");

Assembly i = Assembly.GetExecutingAssembly();

Console.WriteLine("Полное имя:" + i.FullName);

Console.WriteLine("Исполняемый файл:" + i.Location);

}

// Получение информации о типе

static void TypeInfo()

{

Car obj = new Car();

Type t = obj.GetType();

//другой способ

//Type t = typeof(Car);

Console.WriteLine("\nИнформация о типе:");

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("\nCar реализует IComparable -> " +

t.GetInterfaces().Contains(typeof(IComparable))

);

}

// Пример использования метода InvokeMember

static void InvokeMemberInfo()

{

Type t = typeof(Car);

Console.WriteLine("\nВызов метода:");

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

//Параметры вызова метода

object[] parameters = new object[] {40, 500};

//Вызов метода

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

Console.WriteLine("GetFuelConsumption(40,500) - {0}", Result);

}

/// Работа с атрибутами

static void AttributeInfo()

{

Type t = typeof(Car);

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);

}

}

}

static void Main(string[] args)

{

double x, y;

Console.WriteLine("Введите Х");

x = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Введите Y");

y = Convert.ToDouble(Console.ReadLine());

ComparingMethod(x, y, Bigger);

ComparingMethod(x, y, Lower);

//Лямбда-выражение

Comparing biggerlambda = (double x, double y) =>

{

if (x > y) return $"{x} больше, чем {y}";

else if (y > x) return $"{y} больше, чем {x}";

else return $"{x} и {y} равны";

};

ComparingMethod(x, y, biggerlambda);

Console.WriteLine("Нажмите любую клавишу для продолжения");

Console.ReadLine();

//Обобщенные делегаты

ComparingMethodFunc(x, y, Lower);

Console.WriteLine("----------------------------------------------------------");

AssemblyInfo();

TypeInfo();

InvokeMemberInfo();

AttributeInfo();

}

}

# Результат работы программы

