**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИН. Э.БАУМАНА**

Лабораторная работа №6 по курсу

*Базовые компоненты Интернет технологий*

РАБОТА С ДЕЛЕГАТАМИ И РЕФЛЕКСИЕЙ

НА ЯЗЫКЕ C#

студента 2-го курса

РТ факультета группы РТ5-31

Калина Владимира

Москва – 2016г.

**Цель работы:**   
1.Разработать программу для работы с делегатами

2.Разработать программу для работы с рефлексией

**Описание программы:**  
**1.**Программа разработана в виде консольного приложения на языке C#. Определить делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа**.** Написать метод, соответствующий данному делегату.Написать метод, принимающий разработанный делегат, в качестве одного из входным параметров. Осуществить вызов метода, передавая в качестве параметра-делегата:

• метод написанный раннее;

• лямбда-выражение.

Осуществить данный метод повторно, используя вместо разработанного делегата, обобщенный делегат Func<> или Action<>, соответствующий сигнатуре разработанного делегата.

**2.**Программа разработана в виде консольного приложения на языке C#.Создать класс, содержащий конструкторы, свойства, методы. С использованием рефлексии вывести информацию о конструкторах, свойствах, методах. Создать класс атрибута (унаследован от класса System.Attribute). Назначить атрибут некоторым свойствам классам. Вывести только те свойства, которым назначен атрибут. Вызвать один из методов класса с использованием рефлексии

**Текстпрограммы**

1. **ДЕЛЕГАТЫ**

using System;

usingSystem.Collections.Generic;

usingSystem.Linq;

usingSystem.Text;

usingSystem.Threading.Tasks;

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

delegateintPlusOrMinus(int p1, int p2);

classProgram

{

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

staticintPlus(int p1, int p2) { return p1 + p2; }

staticintMinus(int p1, int p2) { return p1 - p2; }

///<summary>

/// Использование обощенного делегата Func<>

///</summary>

staticvoidPlusOrMinusMethodFunc(stringstr, 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>

///Использованиеделегата

///</summary>

staticvoidPlusOrMinusMethod(stringstr, int i1, int i2, PlusOrMinusPlusOrMinusParam)

{

int Result = PlusOrMinusParam(i1, i2);

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

}

staticvoid 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

PlusOrMinuspm1 = newPlusOrMinus(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);

//Созданиеанонимногометода

PlusOrMinus pm3 = delegate(int param1, int param2)

{

return param1 + param2;

};

PlusOrMinusMethod("Созданиеэкземпляраделегатанаосновеанонимногометода: ", i1, i2, pm2);

//Лямбда-выражение в виде переменной

PlusOrMinus pm4 = (int x, int y) =>

{

int z = x + y;

return z;

};

inttest = pm4(1, 2);

PlusOrMinusMethod("Создание экземпляра делегата на основе лямбда-выражения в виде переменной: ", i1, i2, pm4);

//Пример использования внешней переменной

intouter = 100;

PlusOrMinus pm5 = (int x, int y) =>

{

int z = x + y + outer;

return z;

};

int test2 = pm5(1, 2);

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\nИспользованиеобощенного делегата Func<>");

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

stringOuterString = "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();

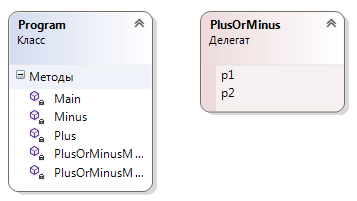
}

}

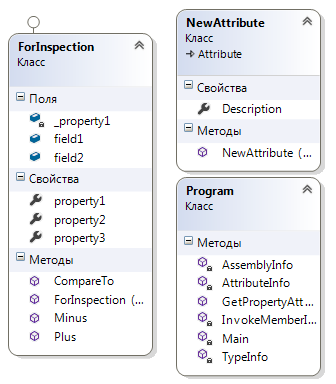
}

1. **РЕФЛЕКСИЯ**
2. usingSystem;
3. usingSystem.Collections.Generic;
4. usingSystem.Linq;
5. usingSystem.Reflection;
6. usingSystem.Text;
7. publicclassForInspection :IComparable
8. {
9. publicForInspection() { }
10. publicForInspection(int i) { }
11. publicForInspection(stringstr) { }
12. publicintPlus(int x, int y) { return x + y; }
13. publicintMinus(int x, int y) { return x - y; }
14. [NewAttribute("Описание для property1")]
15. publicstring property1
16. {
17. get{ return \_property1; }
18. set{ \_property1 = value; }
19. }
20. privatestring \_property1;
21. publicint property2 { get; set; }
22. [NewAttribute(Description = "Описаниедля property3")]
23. publicdouble property3 { get; privateset; }
24. publicint field1;
25. publicfloat field2;
26. ///<summary>
27. /// Реализация интерфейса IComparable
28. ///</summary>
29. publicintCompareTo(objectobj)
30. {
31. return 0;
32. }
33. }
34. namespaceReflection
35. {
36. ///<summary>
37. /// Класс атрибута
38. ///</summary>
39. [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
40. publicclassNewAttribute :Attribute
41. {
42. publicNewAttribute() { }
43. publicNewAttribute(stringDescriptionParam)
44. {
45. Description = DescriptionParam;
46. }
47. publicstring Description { get; set; }
48. }
49. }
50. namespaceReflection
51. {
52. classProgram
53. {
54. ///<summary>
55. /// Проверка, что у свойства есть атрибут заданного типа
56. ///</summary>
57. ///<returns>Значение атрибута</returns>
58. publicstaticboolGetPropertyAttribute(PropertyInfocheckType, TypeattributeType, outobject attribute)
59. {
60. boolResult = false;
61. attribute = null;
62. //Поиск атрибутов с заданным типом
63. varisAttribute = checkType.GetCustomAttributes(attributeType, false);
64. if (isAttribute.Length> 0)
65. {
66. Result = true;
67. attribute = isAttribute[0];
68. }
69. returnResult;
70. }
71. ///<summary>
72. /// Получение информации о текущей сборке
73. ///</summary>
74. staticvoidAssemblyInfo()
75. {
76. Console.WriteLine("Вывод информации о сборке:");
77. Assembly i = Assembly.GetExecutingAssembly();
78. Console.WriteLine("Полноеимя:" + i.FullName);
79. Console.WriteLine("Исполняемыйфайл:" + i.Location);
80. }
81. ///<summary>
82. /// Получение информации о типе
83. ///</summary>
84. staticvoidTypeInfo()
85. {
86. ForInspectionobj = newForInspection();
87. Type t = obj.GetType();
88. //другой способ
89. //Type t = typeof(ForInspection);
90. Console.WriteLine("\nИнформация о типе:");
91. Console.WriteLine("Тип " + t.FullName + " унаследован от " + t.BaseType.FullName);
92. Console.WriteLine("Пространство имен " + t.Namespace);
93. Console.WriteLine("Находитсявсборке " + t.AssemblyQualifiedName);
94. Console.WriteLine("\nКонструкторы:");
95. foreach (var x int.GetConstructors())
96. {
97. Console.WriteLine(x);
98. }
99. Console.WriteLine("\nМетоды:");
100. foreach (var x int.GetMethods())
101. {
102. Console.WriteLine(x);
103. }
104. Console.WriteLine("\nСвойства:");
105. foreach (var x int.GetProperties())
106. {
107. Console.WriteLine(x);
108. }
109. Console.WriteLine("\nПоляданных (public):");
110. foreach (var x int.GetFields())
111. {
112. Console.WriteLine(x);
113. }
114. Console.WriteLine("\nForInspectionреализуетIComparable -> " +
115. t.GetInterfaces().Contains(typeof(IComparable))
116. );
117. }
118. ///<summary>
119. /// Пример использования метода InvokeMember
120. ///</summary>
121. staticvoidInvokeMemberInfo()
122. {
123. Type t = typeof(ForInspection);
124. Console.WriteLine("\nВызов метода:");
125. //Создание объекта
126. //ForInspectionfi = newForInspection();
127. //Можно создать объект через рефлексию
128. ForInspection fi = (ForInspection)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, newobject[] { });
129. //Параметры вызова метода
130. object[] parameters = newobject[] { 3, 2 };
131. //Вызов метода
132. object Result = t.InvokeMember("Plus", BindingFlags.InvokeMethod, null, fi, parameters);
133. Console.WriteLine("Plus(3,2)={0}", Result);
134. }
135. ///<summary>
136. /// Работа с атрибутами
137. ///</summary>
138. staticvoidAttributeInfo()
139. {
140. Type t = typeof(ForInspection);
141. Console.WriteLine("\nСвойства, помеченные атрибутом:");
142. foreach (var x int.GetProperties())
143. {
144. objectattrObj;
145. if (GetPropertyAttribute(x, typeof(NewAttribute), outattrObj))
146. {
147. NewAttributeattr = attrObjasNewAttribute;
148. Console.WriteLine(x.Name + " - " + attr.Description);
149. }
150. }
151. }
152. staticvoid Main(string[] args)
153. {
154. AssemblyInfo();
155. TypeInfo();
156. InvokeMemberInfo();
157. AttributeInfo();
158. Console.ReadLine();
159. }
160. }
161. }

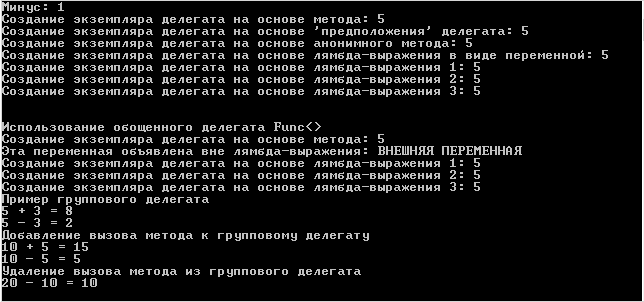
**Дерево классов  
1.**

****

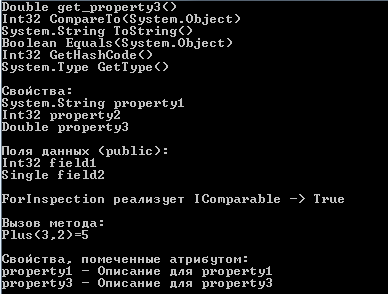
**2.**

****

**Результат работы программы**

**1.**

**2.**

****