**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №6

«Делегаты и рефлексия»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-33 |  | Гапанюк Ю.Е. |
| Миронова Александра |  |  |
| Подпись и дата:  22.12.2020 |  | Подпись и дата: |

Москва, 2020 г.

**Задание:**

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

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

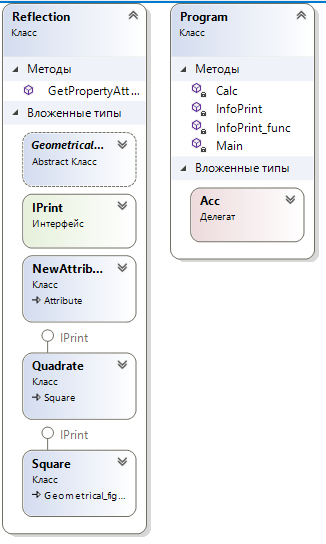
1. Программа должна быть разработана в виде консольного приложения на языке C#.
2. Определите делегат, принимающий несколько параметров различных типов и возвращающий значение произвольного типа.
3. Напишите метод, соответствующий данному делегату.
4. Напишите метод, принимающий разработанный Вами делегат, 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:**



**Текст программы:**

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Delegates

{

class Program

{

delegate double Acc(string billnum, int bill, double percentage);

static double Calc(string billnum, int bill, double percentage)

{

Console.WriteLine("Номер счета: {0}", billnum);

Console.Write("Сумма на счету после начисления процентов: ");

return bill\* (1 + percentage);

}

static void InfoPrint(string name, string fam, string otch, string billnum, int bill,

double percentage, Acc billinfo)

{

Console.WriteLine("Информация о клиенте");

Console.WriteLine("Имя {0}", name);

Console.WriteLine("Фамилия {0}", fam);

Console.WriteLine("Отчество {0}", otch);

Console.WriteLine("{0}", billinfo(billnum, bill, percentage));

}

static void InfoPrint\_func(string name, string fam, string otch, string billnum, int bill,

double percentage, Func<string, int, double, double> billinfo)

{

Console.WriteLine("Информация о клиенте");

Console.WriteLine("Имя {0}", name);

Console.WriteLine("Фамилия {0}", fam);

Console.WriteLine("Отчество {0}", otch);

Console.WriteLine("{0}", billinfo(billnum, bill, percentage));

}

static void Main(string[] args)

{

Console.WriteLine("Program is done by Mironova Alexandra Romanovna group IU5-33b");

Console.WriteLine("\n");

// вызов метода для определения суммы на счету после определения процентов

// 2способа

Console.WriteLine("\n");

Console.WriteLine("Вызов метода с передачей метода Сalc в качестве параметра");

InfoPrint("Меркулова", "Ирина", "Владимировна", "0445678", 52000, 0.15, Calc);

Console.WriteLine("\n");

Console.WriteLine("Вызов метода с передачей лямбда-выражения в качестве параметра");

InfoPrint("Меркулова", "Ирина", "Владимировна", "0445678", 52000, 0.15,

(string billnum, int bill, double percentage) =>

{

Console.WriteLine("Номер счета: {0}", billnum);

Console.Write("Сумма на счету после начисления процентов: ");

return bill \* (1 + percentage);

});

Console.WriteLine("\n");

// объявление лямбда-выражения

Func<string, int, double, double> NewBill\_3;

NewBill\_3 = (string billnum, int bill, double percentage) =>

{

Console.WriteLine("Номер счета: {0}", billnum);

Console.Write("Сумма на счету после начисления процентов: ");

return bill \* (1 + percentage);

};

// вызов метода для определения суммы на счету после определения процентов

//с сипользованием лямбда-выражения

Console.WriteLine("Вызов метода с передачей лямбда-выражения Func в качестве параметра");

InfoPrint\_func("Меркулова", "Ирина", "Владимировна","0445678", 52000, 0.15, Calc);

}

}

}

Reflection.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Reflection;

namespace Delegates

{

class Reflection

{

static void Main(string[] args)

{

Console.WriteLine("Program is done by Mironova Alexandra Romanovna group IU5-33b");

Console.WriteLine("\n");

// создадим объект типа Quadrate

Console.WriteLine("Quadrate");

Quadrate MyQuadrate = new Quadrate(3);

Console.WriteLine("Method");

MyQuadrate.Area();

Console.WriteLine("Interface");

MyQuadrate.Print();

Console.WriteLine("\n");

// вывод информации о классе при помощи рефлексии конструкторах, свойствах, методах.

Type t = MyQuadrate.GetType();

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("\nForInspection реализует IComparable -> " + t.GetInterfaces().Contains(typeof(IComparable)));

// вывод свойств, которым назначен атрибут

Type t1 = typeof(Quadrate);

Console.WriteLine("\nСвойства, помеченные атрибутом:");

foreach (var x in t1.GetProperties())

{

object attrObj;

if (GetPropertyAttribute(x, typeof(NewAttribute), out attrObj))

{

NewAttribute attr = attrObj as NewAttribute; // Для приведения полученного значения типа object к требуемому типу NewAttribute

Console.WriteLine(x.Name + " - " + attr.Description);

}

}

Type t2 = typeof(Quadrate);

Console.WriteLine("\nВызов метода:");

//Создание объекта

//Quadrate Quade = new Quadrate(3);

//Можно создать объект через рефлексию

Quadrate Quade = (Quadrate)t.InvokeMember(null, BindingFlags.CreateInstance, null, null, new object[] {3});

//Параметры вызова метода

object[] parameters = new object[] { 3, 3 };

//Вызов метода

object Result = t2.InvokeMember("Area", BindingFlags.InvokeMethod, null, Quade, new object[] { });

//Console.WriteLine("Area(3,3)={0}", (int)Result);

// Метод InvokeMember принимает различные

}

////////////////////////////////////-

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

public class NewAttribute : Attribute

{

public NewAttribute() { }

public NewAttribute(string DescriptionParam) { Description = DescriptionParam; }

public string Description { get; set; }

}

// 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;

}

////////////////////////////////////-

interface IPrint

{

void Print();

}

abstract class Geometrical\_figure

{

public virtual void Area()

{

}

public virtual void ToString()

{

}

}

////////////////////////////////////-

class Square : Geometrical\_figure, IPrint

{

public Square(double length, double height)

{

this.length = length;

this.height = height;

}

private double length;

private double height;

override public void Area()

{

Console.WriteLine("Height: {0} Length: {1} Square area = {2}", height.ToString(), length.ToString(),

(length \* height).ToString());

}

public void Print()

{

this.Area();

}

[New("Описание для Length")]

public double Length

{

get

{

return length;

}

/\* set

{

this.length = value;

}\*/

}

[New("Описание для Heigth")]

public double Height

{

get

{

return height;

}

/\* set

{

this.height = value;

}\*/

}

}

////////////////////////////////////-

class Quadrate : Square, IPrint

{

public Quadrate(double side) : base(side, side)

{

this.side = side;

}

private double side;

override public void Area()

{

Console.WriteLine("Side: {0} Quadrate area = {1}", side.ToString(), (side \* side).ToString());

}

new public void Print()

{

this.Area();

}

}

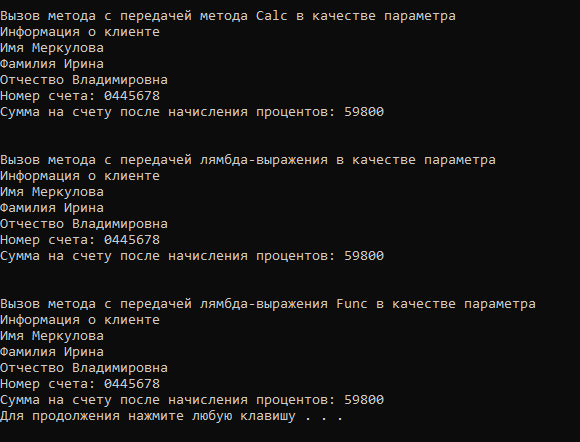
////////////////////////////////////-

}

}

**Экранные формы с примерами выполнения программы:**

Program.cs



Reflection.cs

