**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 №6.2

«Рефлексия»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Бабин Артём |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

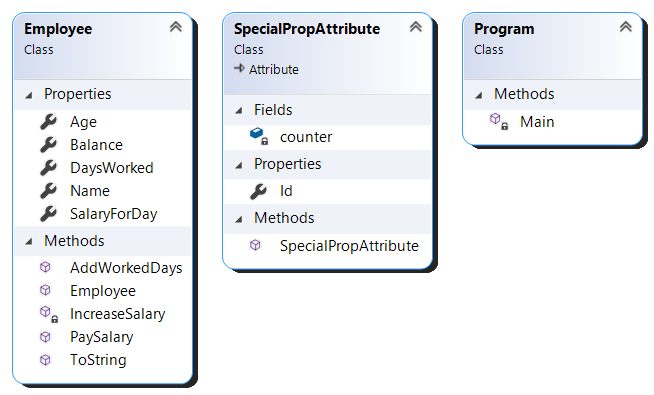
**Условие задания:**

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.Reflection;

namespace BKIT\_LAB6.\_2

{

class Program

{

static void Main(string[] args)

{

Employee em = new Employee("Babin Artem", 18, 50);

#region usage example

/\*Console.Write(em);

em.AddWorkedDays(15);

Console.Write(em);

em.AddWorkedDays(180);

Console.Write(em);

em.PaySalary();

Console.Write(em);\*/

#endregion

Type employeeType = typeof(Employee);

Console.WriteLine("Class Employee info\n\nConstructiors:");

ConstructorInfo[] constructors = employeeType.GetConstructors();

foreach (var constructor in constructors)

{

string info = (constructor.IsPublic) ? "public " : "private ";

info += employeeType.Name + "(";

ParameterInfo[] parameters = constructor.GetParameters();

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

{

info += parameters[i].ParameterType.Name + " " + parameters[i].Name;

if (i != parameters.Length - 1)

info += ", ";

}

info += ");";

Console.WriteLine("\t" + info);

}

Console.WriteLine("Properties:");

PropertyInfo[] props = employeeType.GetProperties();

foreach (var prop in props)

{

string info = "public " + prop.PropertyType.Name + " " + prop.Name + " { ";

if (prop.GetMethod.IsPrivate)

info += "private ";

info += "get; ";

if (prop.SetMethod.IsPrivate)

info += "private ";

info += "set; }";

Console.WriteLine("\t" + info);

/\*var attributes = prop.GetCustomAttributes(typeof(SpecialPropAttribute), false);

if (attributes.Length != 0)

Console.WriteLine("\t\tThis property has a 'SpecialProp' attribute with ID = " + (attributes[0] as SpecialPropAttribute).Id);\*/

}

Console.WriteLine("Methods:");

MethodInfo[] methods = employeeType.GetMethods(BindingFlags.NonPublic | BindingFlags.Public | BindingFlags.Instance | BindingFlags.DeclaredOnly);

foreach (var method in methods)

{

string info = (method.IsPublic) ? "public " : "private ";

info += method.ReturnType.Name + " " + method.Name + "(";

ParameterInfo[] parameters = method.GetParameters();

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

{

info += parameters[i].ParameterType.Name + " " + parameters[i].Name;

if (i != parameters.Length - 1)

info += ", ";

}

info += ");";

Console.WriteLine("\t" + info);

}

Console.WriteLine("Properties with 'SpecialProp' attribute:");

props = employeeType.GetProperties();

foreach (var prop in props)

{

var attributes = prop.GetCustomAttributes(typeof(SpecialPropAttribute), false);

if (attributes.Length != 0)

{

string info = "public " + prop.PropertyType.Name + " " + prop.Name + " { ";

if (prop.GetMethod.IsPrivate)

info += "private ";

info += "get; ";

if (prop.SetMethod.IsPrivate)

info += "private ";

info += "set; }";

Console.WriteLine("\t" + info + " ID = " + (attributes[0] as SpecialPropAttribute).Id);

}

}

Console.WriteLine("\nCalling Employee.ToString() method using reflection:");

Console.WriteLine(employeeType.GetMethod("ToString").Invoke(Activator.CreateInstance(employeeType, "Ivanov Ivan", 65, 20), null));

Console.ReadKey();

}

}

}

**Employee.cs:**

using System;

namespace BKIT\_LAB6.\_2

{

class Employee

{

public string Name { get; private set; }

public int Age { get; private set; }

public int DaysWorked { get; private set; }

[SpecialProp]

public double Balance { get; private set; }

[SpecialProp]

public double SalaryForDay { get; private set; }

public Employee(string name, int age, int baseSalaryForDay)

{

Name = name;

Age = age;

SalaryForDay = baseSalaryForDay;

}

public void AddWorkedDays(int days)

{

for (int i = 0; i != days; ++i)

{

Balance += SalaryForDay;

if (++DaysWorked % 90 == 0)

IncreaseSalary();

}

}

private void IncreaseSalary()

{

SalaryForDay \*= 1.2;

}

public void PaySalary()

{

Console.WriteLine($"{Balance}$ paid to {Name}.");

Balance = 0;

}

public override string ToString()

{

return $"EMPLOYEE INFO:\n\t{Name} is {Age} years old.\n\t" +

$"Worked for the company {DaysWorked} days.\n\t" +

$"Salary rate is {SalaryForDay} for day ({SalaryForDay \* 30} for month).\n\t" +

$"Current balance is {Balance}$\n";

}

}

}

**SpecialPropAttribute.cs:**

using System;

namespace BKIT\_LAB6.\_2

{

class SpecialPropAttribute: Attribute

{

public int Id { get; private set; }

private static int counter = 0;

public SpecialPropAttribute()

{

Id = counter++;

}

}

}

**Примеры выполнения программы:**

|  |  |
| --- | --- |
| *Пример* | *Вывод программы* |
| 1 |  |