Lab: Reflection and Attributes

Problems for the ["C# OOP" course @ SoftUni"](#).

You can check your solutions here: https://judge.softuni.org/Contests/1520/Reflection-and-Attributes-Lab

Part I: Reflection

1. Stealer

NOTE: You need a public **StartUp** class with the namespace **Stealer**.

Add the **Hacker** class from the box below to your project.

Hacker.cs

```csharp
public class Hacker
{
    public string username = "securityGod82";
    private string password = "mySuperSecretPassw0rd";

    public string Password
    {
        get => this.password;
        set => this.password = value;
    }

    private int Id { get; set; }

    public double BankAccountBalance { get; private set; }

    public void DownloadAllBankAccountsInTheWorld()
    {
    }
}
```

There is the one nasty hacker, but not so wise though. He is trying to steal a big amount of money and transfer it to his account. The police are after him but they need a professional… Correct - this is you!

You have the information that this hacker is keeping some of his info in private fields. Create a new class named Spy and add inside a method called - **StealFieldInfo,** which receives:

- **string** – the name of the class to investigate
- **an array of string** - names of the fields to investigate

After finding the fields, you must print on the console:

"**Class under investigation: {nameOfTheClass}**"

On the next lines, print info about each field in the following format:

"**{filedName} = {fieldValue}**"

Use **StringBuilder** to concatenate the answer. **Don't change anything in Hacker class!**

In your **Main()** method, you should be able to check your program with the current piece of code.

```csharp
0 references
static void Main(string[] args)
{
    Spy spy = new Spy();
    string result = spy.StealFieldInfo(investigatedClass: "Stealer.Hacker", params requestedFields: "username", "password");
    Console.WriteLine(result);
}
```

Example

Output
Class under investigation: Stealer.Hacker
username = securityGod82
password = mySuperSecretPassw0rd

Solution

```csharp
public string StealFieldInfo(string investigatedClass, params string[] requestedFields)
{
    Type classType = Type.GetType(investigatedClass);
    FieldInfo[] classFields = classType.GetFields(
        BindingFlags.Instance | BindingFlags.Static | BindingFlags.NonPublic | BindingFlags.Public);
    StringBuilder stringBuilder = new StringBuilder();

    Object classInstance = Activator.CreateInstance(classType, new object[] { });

    stringBuilder.AppendLine($"Class under investigation: {investigatedClass}");

    foreach (FieldInfo field in classFields.Where(f => requestedFields.Contains(f.Name)))
    {
        stringBuilder.AppendLine($"{field.Name} = {field.GetValue(classInstance)}");
    }

    return stringBuilder.ToString().Trim();
}
```

2. High-Quality Mistakes

NOTE: You need a public **StartUp** class with the namespace **Stealer**.

You are already an expert on **High-Quality Code**, so you know what kind of **access modifiers** must be set to the members of a class. You should have noticed that our hacker is not familiar with these concepts.

Create a method inside your Spy class called - **AnalyzeAccessModifiers(string className)**. Check all of the **fields and methods access modifiers**. Print on the console all of the **mistakes** in the format:

- Fields

SoftUni

- {fieldName} must be private!
- Getters
  - {methodName} have to be public!
- Setters
  - {methodName} have to be private!

Use **StringBuilder** to concatenate the answer. **Don't change anything in Hacker class!**

In your **Main()** method you should be able to check your program with the current piece of code.

```csharp
0 references
static void Main(string[] args)
{
    Spy spy = new Spy();
    string result = spy.AnalyzeAccessModifiers(className: "Hacker");
    Console.WriteLine(result);
}
```

Example

NOTE: **The order of your output** may **differ** based on your **solution logic**.

Output
username must be private!
get_Id have to be public!
set_Password have to be private!

Solution

```csharp
public string AnalyzeAcessModifiers(string investigatedClass)
{
    Type classType = Type.GetType(investigatedClass);
    FieldInfo[] classFields = classType.GetFields(BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public);
    MethodInfo[] classPublicMethods = classType.GetMethods(BindingFlags.Instance | BindingFlags.Public);
    MethodInfo[] classNonPublicMethods = classType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic);

    StringBuilder stringBuilder = new StringBuilder();

    foreach (FieldInfo field in classFields)
    {
        stringBuilder.AppendLine($"{field.Name} must be private!");
    }
    foreach (MethodInfo method in classNonPublicMethods.Where(m => m.Name.StartsWith("get")))
    {
        stringBuilder.AppendLine($"{method.Name} have to be public!");
    }
    foreach (MethodInfo method in classPublicMethods.Where(m => m.Name.StartsWith("set")))
    {
        stringBuilder.AppendLine($"{method.Name} have to be private!");
    }

    return stringBuilder.ToString().Trim();
}
```

3. Mission Private Impossible

NOTE: You need a public `StartUp` class with the namespace `Stealer`.

It's time to see what this hacker you are dealing with aims to do. Create a method inside your Spy class called - `RevealPrivateMethods(string className)`. Print all private methods in the following format:

All Private Methods of Class: `{className}`
Base Class: `{baseClassName}`
On the next lines, print found method's names each on a new line. Use **StringBuilder** to concatenate the answer. **Don't change anything in Hacker class!** In your **Main()** method, you should be able to check your program with the current piece of code.

```
0 references
static void Main(string[] args)
{
    Spy spy = new Spy();
    string result = spy.RevealPrivateMethods(investigatedClass: "Stealer.Hacker");
    Console.WriteLine(result);
}
```

Example

NOTE: **The order of your output** may **differ** based on your **solution logic**.

Output
All Private Methods of Class: Stealer.Hacker
Base Class: Object
get_Id
set_Id
set_BankAccountBalance
MemberwiseClone
Finalize

Follow us:

Solution

```csharp
public string RevealPrivateMethods(string investigatedClass)
{
    Type classType = Type.GetType(investigatedClass);
    MethodInfo[] classMethods = classType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic);
    StringBuilder stringBuilder = new StringBuilder();

    stringBuilder.AppendLine($"All Private Methods of Class: {investigatedClass}");
    stringBuilder.AppendLine($"Base Class: {classType.BaseType.Name}");

    foreach (MethodInfo method in classMethods)
    {
        stringBuilder.AppendLine(method.Name);
    }

    return stringBuilder.ToString().Trim();
}
```

4. Collector

NOTE: You need a public **StartUp** class with the namespace **Stealer**.

Use reflection to get all **Hacker** methods. Then prepare an algorithm that will recognize which methods are getters and setters.

Print to console each getter on a new line in the format:
> "{name} will return {Return Type}"

Then print all of the setters in the format:
> "{name} will set field of {Parameter Type}"

Use **StringBuilder** to concatenate the answer. **Don't change anything in Hacker class!**

In your **Main()** method you should be able to check your program with the current piece of code.

```csharp
0 references
static void Main(string[] args)
{
    Spy spy = new Spy();
    string result = spy.CollectGettersAndSetters(investigatedClass: "Stealer.Hacker");
    Console.WriteLine(result);
}
```

Example

Output
get_Password will return System.String
get_Id will return System.Int32
get_BankAccountBalance will return System.Double
set_Password will set field of System.String

```
set_Id will set field of System.Int32
set_BankAccountBalance will set field of System.Double
```

Solution

```csharp
public string CollectGettersAndSetters(string investigatedClass)
{
    Type classType = Type.GetType(investigatedClass);

    MethodInfo[] classMethods =
        classType.GetMethods(BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public);
    StringBuilder stringBuilder = new StringBuilder();

    foreach (MethodInfo method in classMethods.Where(m => m.Name.StartsWith("get")))
    {
        stringBuilder.AppendLine($"{method.Name} will return {method.ReturnType}");
    }

    foreach (MethodInfo method in classMethods.Where(m => m.Name.StartsWith("set")))
    {
        stringBuilder.AppendLine($"{method.Name} will set field of {method.GetParameters().First().ParameterType}");
    }

    return stringBuilder.ToString().Trim();
}
```

Part II: Attributes

5. Create Attribute

NOTE: You need a public **StartUp** class with the namespace **AuthorProblem**.

Create attribute **Author** with a **string** element called **name**, that**:**

- Can be used over classes and methods
- Allow multiple attributes of the same type

Examples

StartUp.cs
| ```csharp
[Author("Victor")]
class StartUp
{
    [Author("George")]
    static void Main(string[] args)
    {
    }
}
``` |

```csharp
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class AuthorAttribute : Attribute
{
    public AuthorAttribute(string name)
    {
        this.Name = name;
    }

    public string Name { get; set; }
}
```

6. Code Tracker

NOTE: You need a public **StartUp** class with the namespace **AuthorProblem**.

Create a class **Tracker** with a method:

- **void PrintMethodsByAuthor()**

The method above should **print on the console** information about **each method** that is **written by someone**. You should **print the information** about **the method** and **its creator** in the following format: "**{methodName} is written by {authorName}**". You should be able to call your method and see the output of it as in the example below.

Examples

StartUp.cs

<pre>[Author("Victor")]
class StartUp
{
    [Author("George")]
    static void Main(string[] args)
    {
        var tracker = new Tracker();
        tracker.PrintMethodsByAuthor();
    }
}</pre>

```csharp
public void PrintMethodsByAuthor()
{
    var type = typeof(StartUp);
    var methods = type.GetMethods(BindingFlags.Instance | BindingFlags.Public |
        BindingFlags.Static);
    foreach (var method in methods)
    {
        if (method.CustomAttributes .Any(n => n.AttributeType == typeof(AuthorAttribute)))
        {
            var attributes = method.GetCustomAttributes(false);
            foreach (AuthorAttribute attr in attributes)
            {
                Console.WriteLine("{0} is written by {1}",
                method.Name, attr.Name);
            }
        }
    }
}
```