Lab: Reflection and Attributes

Problems for exercises and homework for the "C# OOP" course @ SoftUni".

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

Part I: Reflection

1. Stealer

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

```
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 one really nasty hacker, but not so wise though. He is trying to steal a big amount of money and transfer it to his own account. The police is after him but they need a proffessional... 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:

- stirng name of the class to investigate
- array of string names of the filds 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.

















```
public static void Main()
{
   Spy spy = new Spy();
    string result = spy.StealFieldInfo("Hacker", "username", "password");
    Console.WriteLine(result);
```

```
Output
Class under investigation: Hacker
username = securityGod82
password = mySuperSecretPassw0rd
```

Solution

```
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

You are already an expert of 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 - AnalyzeAcessModifiers(string className). Check all of the fields and methods access modifiers. Print on the console all of the mistakes in format:

- **Fields**
 - o {fieldName} must be private!
- Getters
 - o {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.

















```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.AnalyzeAcessModifiers("Hacker");
    Console.WriteLine(result);
}
```

```
Output

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

Solution

```
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

It's time to see what this hacker you are dealing with aims to do. Create a method inside your Spy class called - **RevealPrivateMethods(stirng 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.

```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.RevealPrivateMethods("Hacker");
    Console.WriteLine(result);
}
```















```
Output

All Private Methods of Class: Hacker
Base Class: Object
get_Id
set_Id
set_BankAccountBalance
Finalize
MemberwiseClone
```

Solution

```
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

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.

```
public static void Main()
{
    Spy spy = new Spy();
    string result = spy.CollectGettersAndSetters("Hacker");
    Console.WriteLine(result);
}
```











```
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

```
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

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

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

Examples

```
[Author("Ventsi")]
class StartUp
{
    [Author("Gosho")]
    static void Main(string[] args)
    {
    }
}
```

















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

6. Coding Tracker

Create a class Tracker with a method:

• static void PrintMethodsByAuthor()

Examples

```
StartUp.cs

[Author("Ventsi")]
class StartUp
{
    [Author("Gosho")]
    static void Main(string[] args)
    {
       var tracker = new Tracker();
       tracker.PrintMethodsByAuthor();
    }
}
```











