Упражнения: Отражения на типовете

Задача 1. Крадец

Добавете към проекта си класа **Hacker** от кутията по-долу.

Hacker.cs

```
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()
    {
    }
}
```

Има един много гаден хакер, но не е много умен. Опитва се да открадне голяма сума пари и да я прехвърли в собствената си сметка. Полицията го преследва, но им трябва професионалист… Правилно – това сте вие!

Разполагате с информацията, че този хакер пази част от информацията си в private полета. Създайте нов клас, наречен Spy, и добавете в него метод, наречен StealFieldInfo, който да получава:

- низ – име на класа, който да разследва
- масив от низове – имена на полетата, които да разследва

След като намерите полетата, отпечатайте на конзолата:

- “Class under investigation: {имеНаКласа}”

На следващите редове отпечатайте информация за всяко поле в настоящия формат:

- “{имеНаПоле} = {стойностНаПоле}”

Използвайте **StringBuilder** да свържете отговора. Не променяйте нищо в "**Hacker**" класа!

В **Main** метод трябва да можете да проверите програмата си с това парче код:

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

Пример

Изход
Class under investigation: Hacker username = securityGod82 password = mySuperSecretPassw0rd

Решение

```
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();
}
```

Задача 2. Висококачествени грешки

Вече сте експерт в Качествен Код, така че знаете какъв вид модификатори на достъпа трябва да бъде зададен на членовете на класа. Трябва да сте забелязали, че нашият хакер не е запознат с тези концепции.

Създайте метод в своя **Spy** клас, наречен **AnalyzeAcessModifiers(string className)**. Проверете всички модификатори на достъп на полетата и методите. Отпечатайте на конзолата всички грешки във формат:

Полета

- {имеНаПоле} must be private!

Getters

- {имеНаМетод} have to be public!

Setters

- {имеНаМетод} have to be private!

Използвайте **StringBuilder** да свържете отговора. Не променяйте нищо в "**Hacker**" класа!

В Main метода си би трябвало да можете да проверите програмата си със следния блок от код:

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

Пример

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

Решение

```
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();
}
```

Задача 3. Мисия „Частно“ невъзможна

Време е да видим какво цели този хакер, с който се разправяме. Създайте метод в своя Spy клас, наречен RevealPrivateMethods(stirng className). Отпечатайте всички частни методи в следния формат:

- All Private Methods of Class: {имеНаКлас}
- Base Class: {базовКлас}

На следващите редове отпечатайте имената на намерените методи, всеки на нов ред.

Използвайте StringBuilder да свържете отговора. Не променяйте нищо в "Hacker" класа!

В метода Main трябва да можете да проверите програмата си със следния код:

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

Пример

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

Решение

```
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();
}
```

Задача 4. Колектор

Използвайте отражение, за да уловите всички "Hacker" методи. След това подгответе алгоритъм, който да разпознае кои методи са getters и setters.

Отпечатайте на конзолата всеки getter на нов ред във формат:

- {име} will return {Връщан Тип}

След това отпечатайте всички setters във формат:

- {име} will set field of {Тип на параметър}

Използвайте StringBuilder да свържете отговора. Не променяйте нищо в "Hacker" класа!

В Main метода трябва да можете да проверите програмата си със следните няколко реда:

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

Пример

Изход
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

Решение

```
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();
}
```

Задача 5. Поля за жътва

Даден ви е клас RichSoilLand с много полета (вижте предоставената конструкция). Като добър фермер, какъвто вие сте, трябва да ожънете тези полета. Да ги ожънете означава, че трябва да отпечатате всяко поле в определен формат (както е в изхода).

Вход

Ще получите максимум 100 реда с една от следните команди:

- private – отпечатва всички private полета
- protected – отпечатва всички protected полета
- public – отпечатва всички public полета
- all – отпечатва ВСИЧКИ декларирани полета
- HARVEST – край на входните данни

Изход

За всяка команда трябва да отпечатате полетата, които имат съответния модификатор за достъп, описан във входната секция. Форматът, в който полетата трябва да се отпечатат, е:

"<access modifier> <field type> <field name>"

Примери

Вход	Изход
protected HARVEST	protected String testString protected Double aDouble protected Byte testByte protected StringBuilder aBuffer protected BigInteger testBigNumber protected Single testFloat protected Object testPredicate protected Object fatherMotherObject protected String moarString protected Exception inheritableException protected Stream moarStreamz
private public private HARVEST	private Int32 testInt private Int64 testLong private Calendar aCalendar private Char testChar private BigInteger testBigInt private Thread aThread private Object aPredicate private Object hiddenObject private String anotherString

	private Exception internalException private Stream secretStream public Double testDouble public String aString public StringBuilder aBuilder public Int16 testShort public Byte aByte public Single aFloat public Thread testThread public Object anObject public Int32 anotherIntBitesTheDust public Exception justException public Stream aStream private Int32 testInt private Int64 testLong private Calendar aCalendar private Char testChar private BigInteger testBigInt private Thread aThread private Object aPredicate private Object hiddenObject private String anotherString private Exception internalException private Stream secretStream
all HARVEST	private Int32 testInt public Double testDouble protected String testString private Int64 testLong protected Double aDouble public String aString private Calendar aCalendar public StringBuilder aBuilder

	private Char testChar public Int16 testShort protected Byte testByte public Byte aByte protected StringBuilder aBuffer private BigInteger testBigInt protected BigInteger testBigNumber protected Single testFloat public Single aFloat private Thread aThread public Thread testThread private Object aPredicate protected Object testPredicate public Object anObject private Object hiddenObject protected Object fatherMotherObject private String anotherString protected String moarString public Int32 anotherIntBitesTheDust private Exception internalException protected Exception inheritableException public Exception justException public Stream aStream protected Stream moarStreamz private Stream secretStream

Задача 6. Black Box Integer

Помагате на свой приятел, който е все още в OOP Basics курса – името му е Пешослав (да не се бърка с реални хора или треньори). Той е малко „бавничък“ и е направил клас, в който всички членове са private. Вашите задачи са да представите с конкретни примери обект от неговия клас (винаги с начална стойност 0) и после да извикате всички методи, които той има. Ограничението ви е да не променяте „ръчно“ нищо в самия клас (смятайте го за черна кутия). Можете да разглеждате класа му, но не го пипайте! Самият клас се казва BlackBoxInt и е обвивка за базовия тип int.

Методите, които има този клас, са:

- Add(int)
- Subtract(int)
- Multiply(int)
- Divide(int)
- LeftShift(int)
- RightShift(int)

Вход

Входът ще се състои от редове във вида:

- <име на метод>_<стойност>

Например: Add_115

Входът винаги ще е валиден и в описания формат, така че няма нужда да го проверявате изрично. Спирате да получавате вход когато срещнете командата "END".

Изход

Всяка команда (освен END) трябва да отпечатва настоящата стойност на innerValue от BlackBoxInt обекта, който представяте. Не мамете, като предефинирате ToString() в класа – трябва да вземете стойността от private полето.

Примери

Вход	Изход
Add_999999	999999
Subtract_19	999980
Divide_4	249995
Multiply_2	499990
RightShift_1	249995
LeftShift_3	1999960
END	

Задача 7. BarracksWars – Нова фабрика

Даден ви е малък конзолен проект, наречен Barracks (неговият код е включен в предоставената програмна конструкция).

Основните функционалности на проекта са добавяне на нови единици към склада и отпечатване на доклад със статистики за единиците, които в момента са в склада. Първо нека прегледаме оригиналната задача преди проектът да е бил създаден:

Вход

Входът се състои от команди, всяка на отделен ред. Командите, които изпълняват функционалностите, са:

- add <Archer/Swordsman/Pikeman/{…}> - добавя единица към склада.
- report – отпечатва статистика по лексикологичен ред за единиците в склада.

- fight – край на входните данни.

Изход

Всяка команда освен fight трябва да отпечатва изхода си на конзолата.

add трябва да отпечатва: "<Archer/Swordsman/Pikeman/{…}> added!"

report трябва да отпечатва цялата информация в склада във формата: "<UnitType> -> <UnitQuantity>", сортиран с UnitType

Ограничения

Входът ще представлява не повече от 1000 реда

Командата report никога няма да бъде дадена преди коя да е валидна команда да е дадена

Вашата задача

1) Трябва да проучите кода на проекта и да разберете как работи. В него обаче има части, които не са имплементирани (оставени са с TODO). Трябва да имплементирате функционалността на метода CreateUnit в класа UnitFactory, така че да създаде единица на базата на нейния тип, получен като параметър. Имплементирайте я по такъв начин, че когато добавите нова единица, тя да може да бъде създадена без да е необходимо да се променя нещо в UnitFactory класа (ще ви кажа на ушенце: използвайте отражение). Може да използвате подхода, наречен Simple Factory.

2) Добавете два нови класа за единици (ще има тестове, които ги изискват) - Horseman с 50 здраве И 10 атака и Gunner с 20 здраве и 20 атака.

Ако правилно изпълните всичко в тази задача, трябва да добавяте код само в папките Factories и Units.

Примери

Вход	Изход
add Swordsman add Archer add Pikeman report add Pikeman add Pikeman report fight	Swordsman added! Archer added! Pikeman added! Archer -> 1 Pikeman -> 1 Swordsman -> 1 Pikeman added! Pikeman added! Archer -> 1 Pikeman -> 3 Swordsman -> 1
add Pikeman add Pikeman	Pikeman added! Pikeman added!

add Gunner	Gunner added!
add Horseman	Horseman added!
add Archer	Archer added!
add Gunner	Gunner added!
add Gunner	Gunner added!
add Horseman	Horseman added!
report	Archer -> 1
fight	Gunner -> 3
	Horseman -> 2
	Pikeman -> 2

Задача 8. BarracksWars – Командите отвръщат на удара

Както може би сте забелязали, командите в проекта от задача 3 са имплементирани чрез a switch case и извиквания на методи в класа Engine. Въпреки че този подход работи, той има недостатъци когато добавяте нова команда, защото за нея трябва да бъде добавен нов case. В някои проекти може да нямате достъп до класа Engine и това не би работило. Представете си този проект да е outsource-нат - outsourcing фирмата няма да има достъп до двигателя. Направете така, че когато искат да добавят нова команда, нищо в Engine да не трябва да се променя.

За да постигнете това, употребете шаблон в проектирането, наречен Command Pattern. Правили сме това и преди в BashSoft Lab и можете да погледнете и там за съвети. Използвайте предоставеният интерфейс Executable като рамка за класовете на командите. Поставете новите командни класове в предоставения commands пакет в core. Също може да направите интерпретатор на команди, за да откачите тази функционалност от Engine. Ето как трябва да изглежда базовата (абстрактна) команда:

```
public abstract class Command : IExecutable
{
    private string[] data;
    private IRepository repository;
    private IUnitFactory unitFactory;

    protected Command(string[] data, IRepository repository, IUnitFactory unitFactory)
    {
        this.Data = data;
        this.Repository = repository;
        this.UnitFactory = unitFactory;
    }

    protected string[] Data
    {
        get { return this.data; }
        private set { this.data = value; }
    }

    protected IRepository Repository
    {
        get{ return this.repository; }
        private set{ this.repository = value; }
    }

    protected IUnitFactory UnitFactory
    {
        get { return this.unitFactory; }
        private set { this.unitFactory = value; }
    }

    public abstract string Execute();
}
```

Забележете, че всички команди, които разширяват тази, ще имат както склад, така и UnitFactory, въпреки че не всички се нуждаят от тях. Оставете това така за тази задача, защото за да работи отражението трябва всички конструктори да приемат едни и същи параметри. Ще видим как да заобиколим този проблем в следващата задача.

След като сте имплементирали шаблона, добавете нова команда. Ще има следният синтаксис:

retire <UnitType> - Всичко, което трябва да направи, е да премахне единица от дадения тип от склада.

Ако в момента няма такива единици в склада, отпечатайте: "No such units in repository."

Ако в момента има такъв единици в склада, отпечатайте: "<UnitType> retired!"

За да имплементирате командата, ще трябва да имплементирате и съответния метод в UnitRepository.

Ако правилно изпълните всичко в тази задача, трябва да пишете/променяте код само в Core и Data пакетите.

Примери

Вход	Изход

retire Archer	No such units in repository.
add Pikeman	Pikeman added!
add Pikeman	Pikeman added!
add Gunner	Gunner added!
add Horseman	Horseman added!
add Archer	Archer added!
add Gunner	Gunner added!
add Gunner	Gunner added!
add Horseman	Horseman added!
report	Archer -> 1
retire Gunner	Gunner -> 3
retire Archer	Horseman -> 2
report	Pikeman -> 2
retire Swordsman	Gunner retired!
retire Archer	Archer retired!
fight	Archer -> 0
	Gunner -> 2
	Horseman -> 2
	Pikeman -> 2
	No such units in repository.
	No such units in repository.

Задача 9. * BarracksWars - Завръщане на Зависимостите

В последната част от тази епична трилогия от задачи ще разрешим проблема, при който всички команди получават всички utility класове като параметри в своите конструктори. Можем да постигнем това, използвайки подход, наречен dependency injection container. Този подход се използва в много библиотеки.

Ще променим малко този подход. Премахнете всички полета от абстрактните команди освен data. Вместо това сложете каквито полета са нужни на всяка команда в конкретния клас. Създайте параметър, наречен Inject, и направете така, че да може да се използва само на полета. Подайте този параметър за полетата, които трябва да променим чрез отражение. След като сте подготвили всичко това, напишете необходимия код за отражение в Командния Интерпретатор (който трябва да сте преработили от Engine в предната задача).

Можете да използвате същия пример като в предната задача, за да проверите дали сте я изпълнили правилно.

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).