Code review

This document aims to describe the work done during week four, which consisted in attempting 10 code review challenges.

Performing a Code Review

After performing 10 code reviews on some code snippets we had to choose the one that we thought was done better and comment further on it. This is the one I chose, it simulates a simple environment with people and departments:

```
class HumanResources
 {
    static void Main(string[] args)
     {
         Person person = new Person();
         person = person.GetManager();
     }
 }


 class Person
 {
     public Department Department { get; set; }
     public Person GetManager()
     {
         return Department.GetManager();
     }
 }


 class Department
 {
     private readonly Person _manager;
     public Department(Person manager)
     {
         _manager = manager;
     }
     public Person GetManager()
     {
         return _manager;
     }
 }
```

After analysing the code I came to a few conclusions:

- First of all it would always be nice to see the classes' access level, which, along with some few other things, does not really adhere with the **C# standard coding conventions**. For example the "Person" type in the first line of the Main method could be replaced with "var", since the type is obvious.
- The "GetManager()" method inside the Person class can be removed, that is because the Department class' "GetManager()" method can be called instead. That is a repetition but also extra not-needed work, therefore it does not satisfy the **KISS** and **DRY** principles.
- Finally the first class has no reason to be called "HumanResources" since it does not do anything other than creating a Person object. This kind of violates the **YAGNI** principle, since it is not needed. Also the line "person = person.GetManager();" will not work because the "Department" property of the Person class was never initialised.

Responding to a Code Review

Here follows the code written by me:

```
public class Program
 {
    public static void Main(string[] args)
     {
         var person = new Person();
         var department = new Department(person);
         person.Department = department;
         person = person.Department.GetManager();
     }
 }


 public class Person
 {
     public Department Department { get; set; }
 }


 public class Department
 {
     private readonly Person _manager;
     public Department(Person manager)
     {
         _manager = manager;
     }
     public Person GetManager()
     {
         return _manager;
     }
 }
```

This code addresses all the principle's violations and code smells that I spotted during my code review. Here follows what I did to improve this code:

- I provided the classes with an accessibility level and replaced the Person and Department types in the Main class with "var" because they are easily derivable.
- I renamed the "HumanResources" class to "Program" because it was not doing anything important or human resources' related.
- I removed the "GetManager()" method from the Person class because the Department property already has a "GetManager()" method.
- Finally the main class works a bit differently, in fact before it was not possible to get the Departments' manager because this was not initialised. Now a Person is created, a Department is created with it and the Person's department is initialised, so it can now be accessed.

Further discussion

- One of the principles that I initially thought the code violated was the **Open/Closed Principle (OCP)** but while writing the portfolio I realised that, as simple as it is, this code can be easily extended, for example by changing "Person" into an abstract class and creating an "Employee" and a "Manager" class that can inherit from it.
- According to the results I misinterpreted a code smell: in fact I considered the "GetManager()" issue to be a "Duplicated Code" code smell. The reason was that a method with the same name already existed and could be easily accessed from the Person class itself. Instead, the right choice was **"Middle man"** because the HumanResources class was indeed using the person's "GetManager()" method when it could have used the person's department's "GetManager()" method, without the need of a middle man. Despite my choice I still did the right code change, in fact I feel like both are right answers, after all it is doing something via a middle man but it is also repeating code.