CHAPTER 1

Private en Public

FHICT docenten, met name Alexander van D, Bartosz P, Björn H, Coen C, Frank de N, Frank P, Frenk R, Gertjan S, Inge van E, Jan O, Jean-Paul L, Joeri van B, Marcel V, Mark M, Martijn L, Nico K, Peter L, René B, Ruud H, Wilrik de L.

1.1 Private

- *Information hiding*: `Fields` binnen een `class` maken we altijd `private`.
- Voordeel: beter onderhoudbare software, programmeurs kunnen vanuit andere `classes` niet zomaar in de *internals* van jouw `class`, maar gebruiken de daarvoor bedoelde `methods` en/of `properties`.

1.2 Methods

`Methods` die van buitenaf aanroepbaar moeten zijn zijn dus typisch `public`. Als het `methodes` betreft die alleen bedoeld zijn voor gebruik binnen de `class` kun je ze beter `private` maken.

1.3 Properties

Een property definieert vaak een manier om de waarde van een veld op te vragen of zelfs te veranderen. Het opvragen kan vaak `public` zijn, maar het is bij de meeste velden niet de bedoeling dat de waarde van buitenaf veranderd kan worden.

```
class Persoon
{
   // Maak Velden private
   private string naam;
   private int leeftijd;

   public Persoon(string naam, int leeftijd)
```

```
    {
        this.naam = naam;
        this.leeftijd = leeftijd;
    }

}
```

Het is nu niet mogelijk van buitenaf de waarde van een `veld` als `leeftijd` te veranderen:

```
Persoon persoon = new Persoon("Pietje Puk");
int leeftijd = persoon.leeftijd;  // dit werkt dus NIET!
```

Uiteraard kan er in `class` *Persoon* een `methode` gemaakt worden die de waarde van `veld` *leeftijd* teruggeeft, een zogenaamde *get-method*:

```
public int GetLeeftijd()
{
    return this.leeftijd;
}
```

zodat de leeftijd opgevraagd kan worden:

```
int leeftijd = persoon.GetLeeftijd();
```

Merk op dat de *leeftijd* van buitenaf dus niet veranderd kan worden, maar alleen opgevraagd!

1.4 Waarom?

Wat is hier nu het voordeel van? Nou, binnen een jaar zal de ontwikkelaar van deze `class` de eerste klachten krijgen dat niet de leeftijden niet goed berekend worden, aangezien dat de *leeftijd* geen vaste waarde is: op het moment dat de *persoon* jarig is moet de waarde opgehoogd worden. In dit geval kun je zien dat het verstandiger is de *geboortedatum* van de persoon op te slaan (die verandert namelijk niet) en dan wordt bij het opvragen van de *leeftijd* de goede waarde berekend. De programmeur kan nu zonder problemen de `class` veranderen:

```
class Persoon
{
    // velden
    private string naam;
    private DateTime geboortedatum;

    public int GetLeeftijd()
    {
        int leeftijd;
        ... // voeg hier code toe om de leeftijd te berekenen mbv de
     geboortedatum.
        return leeftijd;
```

```
    }
```

Doordat het veld *leeftijd* `private` was kan de programmeur dit aanpassen zonder dat er elders problemen ontstaan in code die hier gebruik van maakt, externe code roept namelijk de methode *GetLeeftijd()* aan en die zal na de wijziging zonder problemen werken.

1.5 Encapsulation

Stel dat er een bug zit in (de waarde van velden van) een bepaalde `class`, dan is het ook prettig te weten dat (in geval van `private` velden) de bug ergens in de `class` moet zitten. Dit wordt `Encapsulaton` genoemd: een stukje gedrag van een programma wordt afgeschermd. Hierdoor wordt het makkelijker een deel van je programma te hergebruiken.

Encapsulatie betekent kortweg dat een groep fields, methodes en overige eigenschappen gezien worden als een enkel, afgebakende eenheid of object. Dit klinkt wat droog, dus een andere bewoor-ding die mogelijk duidelijker is door encapsulatie te zien als het vermogen van een klasse om fields en methodes die niet interessant zijn voor anderen, te verbergen.

De beste reden om bepaalde onderdelen van je klasse af te schermen is dat je code makkelijker in het gebruik wordt. Klassen gebruiken private fields om hun toestand bij te houden; hoeveel levens heb ik nog, hoe snel mag ik bewegen, et cetera. Deze informatie is voor andere code niet per sé interessant, maar wel essentieel voor de werking van de klasse. Als iedereen zomaar die fields aan zou kunnen passen, wordt de werking van je klasse een stuk onbetrouwbaarder en willekeuriger: wanneer je zelf de enige bent die het aantal levens aan kunt passen, weet je ook precies waar in je code het voor kan komen dat je levens op 0 worden gezet. Dit voordeel valt weg als iedereen het aantal levens aan kan passen.

1.6 External References

Over private[1]

Techopedia[2]

[1] https://softwareengineering.stackexchange.com/questions/143736/why-do-we-need-private-variables

[2] http://www.techopedia.com/definition/3787/encapsulation-c