Les F-01 Objectgeoriënteerd programmeren

In deze lesbrief wordt globaal beschreven wat **object georiënteerd programmeren** (**OOP: object oriented programming)** inhoudt. In deze lesbrief wordt niet uitgelegd *hoe* je object georiënteerd moet programmeren maar wordt beschreven welke begrippen een rol spelen en wat deze begrippen inhouden.

Wil je leren om object georiënteerd te leren programmeren, dan zijn programmeertalen zoals C++ en Java goede talen om mee te starten.

Doel van deze lesbrief is om kennis en inzicht te verschaffen in wat object georiënteerd programmeren inhoudt. Daartoe worden enkele basisbegrippen toegelicht.

1.1 Procedureel programmeren

Procedureel programmeren (ook wel **imperatief programmeren** genoemd) is op natuurlijke wijze in de computergeschiedenis ontstaan. Processors dienden volgens de Von Neumann cyclus (les B-03) door middel van instructies stap voor stap te worden geprogrammeerd. Programmeertalen zagen er aanvankelijk dan ook uit als lijsten van programmeerregels die achtereenvolgens moesten worden uitgevoerd.

In module C heb je geleerd hoe je in de programmeertaal Visual Basic bepaalde programmaonderdelen onder kan brengen in aparte (sub)routines gebruik door gebruik te maken van **functies** en **procedures**. Deze maken de programmeercode overzichtelijk en efficiënt (geen herhalingen).

Hieronder zie je hoe de functie dobbelsteen() meerdere malen wordt gebruikt in de procedure btnGooi, die het gooien van vijf dobbdelstenen bij het spel Yathzee simuleert.

```
Private Sub frmDobbelsteen_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Randomize()
End Sub
Function dobbelsteen() As Integer
    dobbelsteen = 1 + Int(6 * Rnd())
End Function

Private Sub btnGooi_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnGooi.Click
    txtDobbel1.Text = dobbelsteen()
    txtDobbel2.Text = dobbelsteen()
    txtDobbel3.Text = dobbelsteen()
    txtDobbel4.Text = dobbelsteen()
    txtDobbel5.Text = dobbelsteen()
End Sub
```

1.2 Object georiënteerd programmeren

Een stap verder gaat het **object georiënteerd programmeren**. Bij deze wijze van programmeren wordt een programma opgebouwd uit met elkaar samenwerkende **objecten**.

Een **klasse** is een beschrijving van een object. Het is een beschrijving van de vormgeving van de klasse (**attributen** = vormeigenschappen van de klasse) en van de acties die met de klasse kunnen worden verricht (**methoden** = dynamische eigenschappen van de klasse). Dat klinkt allemaal heel abstract maar het volgende voorbeeld maakt hopelijk al wat meer duidelijk. Een voorbeeld van een klasse is de klasse “boek”. De klasse “boek” heeft een aantal attributen zoals “titel”, “auteur”, “uitgever”, “eigenaar”, enz maar ook een aantal methoden zoals “openen”, “uitlenen”, “weggooien” …

Een object is een instantie van een klasse. Zo is het object “Koning van Katoren” een instantie van de klasse “boek”.

Een ander voorbeeld van een klasse is de klasse “mens”. Deze heeft een aantal attributen zoals bijvoorbeeld “lichaamslengte”, “kleur haar” en “naam”. En we kennen ook een aantal methodes zoals “praten”, “schrijven”, “lezen”, “lopen”, “ruiken”, enz.

Je kunt je voorstellen dat ook in games klassen een belangrijke rol spelen. Karakters in games hebben allemaal specifieke vormeigenschappen (attributen) en dynamische eigenschappen (methoden).

1.3 Relaties

Als je de objectleer op de mens toepast zal je zien dat wij eigenlijk bestaan uit andere objecten, o.a. “longen”, “neus”, “lever”, “hart” enz. Als je nog beter zou gaan kijken zal je zelfs zien dat ook die ‘objecten’ weer zijn opgebouwd uit andere objecten. Uiteindelijk zal je ergens uitkomen bij “atomen”, “protonen” en “neutronen”. Nu is dat natuurlijk wel erg veel programmeerwerk. De softwareontwikkelaar zal dan ook altijd bepalen tot hoever het noodzakelijk is om de objectleer toe te passen.
De objecten waaruit een mens bestaat, zoals “longen”, “neus”, “lever”, “hart” zijn op hun beurt ook weer instanties van klassen. Zij komen niet alleen in de klasse “mens” voor, maar bijvoorbeeld ook in de klasse “varken”. Als attributen instanties zijn van een (andere) klasse dan spreken we van een **relatie**. Een relatie verbindt de ene klasse met een andere.

1.4 Overerving

Je zou kunnen zeggen dat de mens als klasse eigenlijk niet bestaat. Vrouwen en mannen hebben tenslotte andere eigenschappen. We noemen de mens dan ook een **abstracte klasse**. Van een abstracte klasse kan je geen instantie maken: er is geen mens denkbaar die slechts de eigenschappen van de klasse “mens” heeft.
Vrouwen en mannen zijn klassen die voor een (groot) deel bestaan uit de abstracte klasse mens en daarnaast nog een aantal specifieke eigenschappen en methoden. Vrouwen en mannen zijn dus **uitbreidingen** van de abstracte klasse mens. Het feit dat vrouwen en mannen beide de eigenschappen van een mens meekrijgen noemen we **overerving**.

1.5 Toegankelijkheid

Je hebt zojuist inmiddels een en ander geleerd over klassen, eigenschappen en methoden. We hebben inmiddels een en ander over klassen geleerd. Ook methoden en eigenschappen hebben een bijzonder kenmerk. Namelijk hun **toegankelijkheid**. Dat wil zeggen: de mate waarin objecten toegang hebben tot de eigenschappen en methoden.

We onderscheiden drie niveaus van toegankelijkheid:

- **public**: mag door alle objecten worden gelezen of ingesteld (eigenschappen) of uitgevoerd (methodes)
- **protected**: mag door alle objecten gelezen of ingesteld (eigenschappen) of uitgevoerd (methodes) en kan worden gebruikt vanuit overgeërfde methodes.
- **private**: mag enkel door het object zelf worden gelezen of ingesteld (eigenschappen) of uitgevoerd (methodes)

1.6 Samenvatting

Bij **procedureel programmeren** wordt gewerkt met **functies** en **procedures**.

Bij **object georiënteerd programmeren** wordt gewerkt met **klassen**, **attributen** en **methoden**. Een **object** is een **instantie** (**exemplaar**) van een klasse.

Abstracte klassen zijn kunstmatige klassen waarvan geen instantie van bestaat maar waarvan door **overerving** de attributen en methoden kunnen worden overgenomen.

Tussen klassen kunnen **relaties** bestaan.

De toegankelijkheid van attributen en methoden kan zijn:

- **public**
- **protected**
- **private**