Object Georiënteerd Programmeren 1

Wouter Brinksma

Deze les

- Bespreken van opzet van het vak
- Bespreken verschillende soorten van programmeren
- Kijken naar Object Georiënteerd Programmeren algemeen
- Bespreken van opdracht voor deze week

Opzet van het vak

- Bij dit vak gaan we kijken naar object georiënteerd programmeren.
- Jullie krijgen een gecombineerde opdracht met het vak Graphics 1
- De gezamenlijke opdracht is om een simulatie te maken van een Amazon magazijn. Hier gaan we volgende week verder naar kijken.
- Dit is een inleidende week.

Opzet van het vak

- We maken gebruik van een reader
- Hierin kun je veel informatie vinden, en deze staat online op Blackboard
- Je krijgt verspreid over de komende weken nieuwe hoofdstukken van de reader digitaal uitgereikt. Deze kun je gebruiken bij de opdrachten

Welke programmeertaal gebruiken we?

- We gaan gebruikmaken van C#
- Alle voorbeelden werken met .NET Core. Dit betekent dat je ze op Windows, MacOS en Linux kunt draaien
- Deze eerste week kun je nog gewoon gebruikmaken van Visual Studio met C#. Volgende week wordt de eindopdracht duidelijk gemaakt, en kijken we of .NET Core handiger is.

Korte inleiding

- Eigen ervaring
- Verschil tussen *weten* en *begrijpen*
- Richard Feynman
- Is ook waar voor programmeren

De basis: Programmeertalen

- **Een programmeertaal is een versameling symbolen en opdrachten waarmee de programmeur commando's kan geven aan de comptuter**
- *Hogere programmeertalen* en *lagere programmeertalen*
- We gaan een aantal van die programmeertalen bekijken

Machinetaal

- Machinetaal is een set aan instructies die direct door de CPU kan worden uitgevoerd
- Numerieke machinecode is geschreven in bits (**1** & **0**)
- Erg foutgevoelig om hierin te programmeren

Machintaal voorbeeld

Instructie Register Waarde

10110000 01100001

Je kunt machinetaal “makkelijker” opschrijven

- Hexadecimale notatie
- B0 61
- Iemand die bekend is met machinetaal kan dit lezen
- Voorbeeld: Nintendo 64 ROMS en emulatoren

Assembly Lanuage

- Assembly language zit een niveau hoger dan machinetaal
- Het is een tekstuele variant van machinecode
- **MOV AL**, 61h
- Dit is beter te begrijpen dan 001001100…

De programmeertaal C

- Waarschijnlijk bij iedereen wel bekend
- C bestaat sinds 1972 en vormt de brug tussen de lagere talen zoals machinetaal en assembly language, en hogere talen
- Veel talen zijn geïnspireerd door C of draaien in een omgeving die is geschreven in C (Python, PHP, etc.)

Programmeertalen classificeren

- Programmeertalen kun je classificeren met verschillende **Programmeerparadigma's**
- Dat zijn manieren van denken of een concept waar een programmeertaal aan voldoet
- We gaan er 4 behandelen:
  - Imperatief Programmeren
  - Procedureel Programmeren
  - Functioneel Programmeren
  - Declaratief Programmeren

Imperatief Programmeren

- **Imperatief programmeren kan het best worden omschreven als een vorm van programmeren waarbij commando's worden gebruikt om de staat van het programma aan te passen**
- Staat van het programma = Alle informatie in het programma
- Lijkt op de state machine van WebGL
- Zelfs machinecode is een vorm van imperatief programmeren

Procedureel Programmeren

- Procedureel programmeren is nauw verbonden met imperatief programmeren
- Procedureel programmeren betekent dat je een programma uit “procedures” kunt opbouw. Dat zijn **functies**

```
#include <stdio.h>

int main()
{
  printf("Hello, World!");
  return 0;
}
```

Control

- Normaal gesproken worden commando’s een voor een uitgevoerd
- Soms willen we dat code in een andere volgorde wordt uitgevoerd
- In een imperatieve/procedurele taal zijn twee vormen van uitvoeringscontrole aanwezig:
  - Expressie-level control
  - Statement-level control

Expressie-level control

- Dit komt voor bij bijvoorbeeld wiskundige expressies:

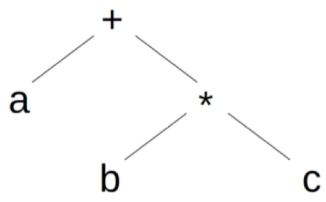
3 + 5 * 6

Er zijn verschillende rekenregels waarmee een programmeertaal rekening moet houden

Dit kun je vormgeven in een expressieboom

Expressieboom

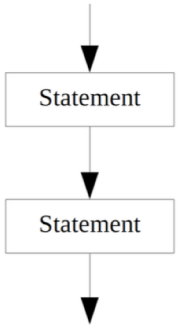
- Een expressieboom van **a + b * c** ziet er als volgt uit:



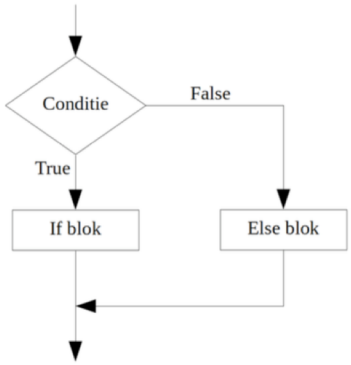
Statement-level control

- Normaal gesproken wordt code van boven naar beneden uitgevoerd
- Soms willen we wat anders, en dit doen we door gebruik te maken van **control statements**
- If, switch, for, while, do-while

Normaal gesproken



If statement



Enkele bijzonderheden

- De meeste control statements kennen jullie wel
- Wanneer gebruik je meestal een for-loop en wanneer een while-loop?
- Wat is een infinite loop?

Functioneel Programmeren

- Functionele programmeertalen zijn als het ware het tegenovergestelde van imperatieve talen. Ze proberen juist niet de staat van het programma aan te passen
- Stukken code die de staat van het programma aanpassen hebben zogenaamde *side effects*
- Deze kunnen gevaarlijk zijn

Functionele functie..

```
fibonacci_aux = \n first second->
  if n == 0 then "" else
  show first ++ "\n" ++ fibanocci_aux (n - 1) second
     (first + second)
fibonacci = \n-> fibonacci_aux n 0 1
main = putStr (fibonacci 10)
```

- Deze functie berekent de Fibonacci reeks. Dat is een reeks waarbij je een nieuw getal krijgt door de vorige twee bij elkaar op te tellen: 0,1,1,2,3,5,8,13,21,enz.

Functionele functie..

- Deze code werkt **recursief** met **opsparende parameters**

```
fibonacci_aux 8 1 2
fibonacci_aux 7 2 3
fibonacci_aux 6 3 5
fibonacci_aux 5 5 8
fibonacci_aux 4 8 13
```

- (**Recursief** = roept zichzelf aan)

Declaratief Programmeren

- In plaats van aan te geven *hoe* een programma iets moet doen, geven declaratieve talen aan *wat* het programma moet doen
- Informatie wordt als relaties opgeslagen in het geheugen
- Werkt erg goed in combinatie met kunstmatige intelligentie

Prolog

- Prolog is een logische/declaratieve taal
- Je kunt er feiten in opslaan:

```
cat(tom).
```

- Je kunt ook feiten opvragen:

```
?- cat(tom).
Yes

?- cat(X).
X = tom
```

Object Georiënteerd Programmeren

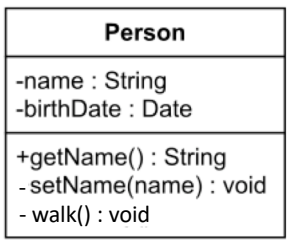
- Het programmeerparadigma waar we ons bij dit vak mee bezig gaan houden
- Deze vorm van programmeren wordt tegenwoordig veel gebruikt in de zakelijke wereld, industrie, overheid, etc.
- Het doel van OOP is om programmeren makkelijker te maken door het dichter bij de *echte wereld* te brengen.

Object Georiënteerd Programmeren

- Je kunt met OOP je software vormgeven volgens het probleem dat je aan het oplossen bent
- Je bouwt je software namelijk op uit objecten die in de echte wereld ook (kunnen) bestaan
- Een beheerssysteem voor een school heeft waarschijnlijk student- en docentobjecten. Studentobjecten kunnen bijvoorbeeld studiepunten hebben. Etc.

Objecten

- Objecten bestaan uit *data* en *gedrag*
- Dit wordt in vaktaal *attributen* en *methoden* genoemd



Information hiding

- Sommige onderdelen van objecten zijn wel zichtbaar voor andere objecten, andere onderdelen niet
- Dit concept heet information hiding, en beschermt je code tegen fouten, verminderd side effect problemen
- *Kunnen andere objecten dit onderdeel veranderen of aanspreken?*

Data abstraction

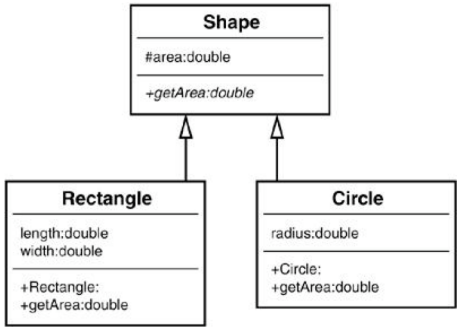
- Programmeurs kunnen zelf objecten maken, deze opbouwen uit verschillende onderdelen en kunnen aangeven of die wel en niet zichtbaar zijn voor andere objecten

Inheritance

- Object georienteerde talen hebben ook voorzieningen voor *inheritance*
- In het Nederlands heet dit overerving, en betekent dat een class elementen overneemt van een andere class
- Bijvoorbeeld: Een werknemer is ook een persoon
  Werknemer erft dus van persoon

Polyformisme

- Polyformisme betekent dat je voor dezelfde methode verschillende implementaties in verschillende objecten kunt hebben:



4 Kernconcepten van OOP

1. **Abstraction**
   Het kunnen omzetten van het probleem (bijvoorbeeld uit de echte wereld) naar de software wereld.
   Kernwoorden: Data abstraction

2. **Seperation**
   Het kunnen kiezen welke onderdelen van een object wel en niet zichtbaar zijn voor de buitenwereld.
   Kernwoorden: Information hiding

3. **Composition**
   Het kunnen opbouwen van objecten uit andere objecten
   Kernwoorden: Modulariteit

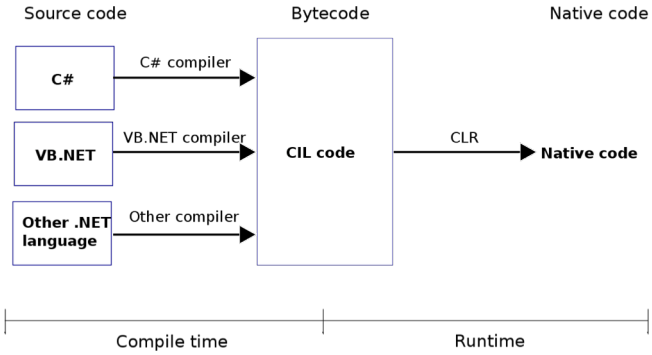
4. **Generalization**
   Het kunnen aangeven van relaties tussen verschillende soorten objecten
   Kernwoorden: Overerving, Polymorfisme

Tot slot: Compileren en C#

- In de reader kun je extra uitleg vinden over compileren
- Voor nu beperken we ons tot de definitie: compileren is code omzetten in uitvoerbare code voor de CPU
- Er bestaan ook *interpreters*. Deze lezen ook net als compilers code in, maar voeren die direct uit in plaats van ze om te zetten

C# is in dit opzicht bijzonder

- C# compileert naar een tussentaal, en ”compileert” daarna nog een keer met een proces dat *Just In Time Compilation* (JIT-Compilation) heet



Samenvatting

- Bespreken van opzet van het vak
- Bespreken verschillende soorten van programmeren
- Kijken naar Object Georiënteerd Programmeren algemeen

Opdracht voor deze week

- Intro-opdracht
- Vooral om te kijken hoe ver iedereen is qua programmeren
- Probeer de opdracht zo netjes mogelijk object georienteerd te programmeren
- Opdracht is te vinden op Blackboard, daar kun je ook de code inleveren

Vragen?

- Volgende week beginnen we met de grotere eindopdracht voor dit vak