Grondslagen v/d beleidsinformatica Hoofdstuk 1: Software en klassen

**1.1 OVERZICHT**

**Software** (*programmatuur)* doet wat gebruiker wenst te doen met het computersysteem

**Objecten:** kleine, consistente modulaire bouwblokken
--> werkgeheugen v/h computersysteem (ook op secundaire media mogelijk)
--> binair: gecodeerd in 0- en 1-signalen
Men beschrijft objecten in **klassen**

**Secundaire media:** schijfgeheugens, CD’s, DVD’s

Men bouwt software op basis van het “*verdeel, organiseer en delegeer*”-principe

**1.2 KLASSEN: DE DEFINITIES VAN OBJECTEN**

Programma is een reeks van instructies
--> samenvoegen tot één geheel (**procedure**)
Men gaat grote procedures opsplitsen in kleinere (onafhankelijke) modules

Een computersysteem ***manipuleert*** gegevens
--> deze “manipulaties” moeten we kunnen definiëren

*Voorbeeld:*Programma om geld af te halen
- Moet rekeningnummer bevatten (**check-digit**: twee cijfers)
- Saldo v/d betrokken rekening (**reëel getal:** bevat komma’s)
- Af te halen bedrag
- Toegestaan krediet

*Zie manipulatieregels pagina 13*

Men heeft bijgevolg **gegevens** en **manipulatieregels**

Objecten kan men vergelijken met een recept in een kookboek
- bevat ingrediënten (**gegevens**)
- bevat stappen voor de bereiding (**manipulatieregels**)

Objecten moeten **consistent** zijn: er mogen geen gegevens, noch manipulaties ontbreken
**!** Ook “extra” nutteloze zaken moet men vermijden
**!** Het moet ook duidelijk zijn wat men tracht te bereiken (3-4 eiwitten is vaag)

De inhoud v/h object moet het toelaten om meerdere kopieën te maken v/h object

**Klassen:** bevat alle objecten die tot de klasse behoren
--> werkt op dezelfde manier als objecten, enkel hoger in rang
**!** Klasse is een ***stuk tekst***

Het ***stuk tekst*** kan men **omzetten** naar **binaire computerobjecten** zodat de software er komt
--> **Compiler:** objectgerichte softwarevertaler
--> **Interpreter:** vertolker v/d softwaretaal

Klasse bevatten gegevens en instructies
De instructies zijn de eigenschappen v/d objecten v/d klassen (**diensten** voor gebruiker object)

Objecten moeten kunnen *manipuleren, onthouden* en *weten* welke gegevens te gebruiken

*Kortom:*Klassen beschrijven alle eigenschappen (**features**) die gemeenschappelijk zijn bij objecten
Deze features kan men opdelen in
 - **attributen:** strategische eigenschappen
 - **procedures:** dynamische eigenschappen
 - **routines:** manipuleren attributen
 - **functies:** gebruiken attributen, wijzigen ze niet

**Attributen** kunnen zelf ook objecten zijn
*Voorbeeld:* bij recept voor taart gebruikt men misschien “chocoladepoeder”, dit wordt ook gemaakt

**! Essentie van objectgerichte software:** **alles is een object, waarvoor klassenbeschrijving bestaat**

**1.3 EEN EENVOUDIG VOORBEELD: DE TELLER**

Bij een teller is er één object, de **stand v/d teller**
--> deze wordt gemanipuleerd door de **features**

Na het **definiëren** van de **klasse**
***class TELLER***Moet men het toelaten dat er **procedures** kunnen **gemaakt worden
*creation*** *(of create)* ***maken***We **definiëren** ook het **object
*feature
 stand: INTEGER***

Vervolgens sommen we de verschillende opties tot manipulatie op

**INTEGER**: deze optie laat het toe zaken te **onthouden**

**1.4 KLASSEN EN DIENSTEN**

**OBJECTPRINCIPE 1
Object wordt ontworpen rond attribuutwaarden die het wil onthouden/memoriseren**

Bij de teller is het belangrijk dat het getal onthouden wordt

Men moet ook omschrijven wat een object kan uitvoeren, manipuleren
--> manipulatie van **attribuutwaarden** (ingrediënten)
Deze “manipulatie” kan meerdere objecten omvatten

*Wat kan ik voor u doen***--> features** met de manipulaties
*Wat kan ik voor u onthouden****--> stand: INTEGER*** voor te onthouden (Integer is een geheel getal)

**OBJECTPRINCIPE 2
Procedure (routine) is zo klein en eenvoudig als mogelijk**
**!** Procedures van één instructie bestaan

Men kan de teller uitbreiden naar bijvoorbeeld een bankrekening
--> nu maakt men gebruik van een kommagetal bij het saldo (voordien stand v/d teller)
Daarnaast kan het bedrag ook vermeerder en verminderd worden met en concreet ander getal

We maken gebruik van volgende feature
***feature
 saldo: REAL***

Deze keer zijn het geen TELLER-objecten, maar REKENING-objecten

**1.5 OBJECTEN GEBRUIKEN: INKAPSELING**

**OBJECTPRINCIPE 3
Ieder object behoort tot een klasse**

Men kan objecten aanmaken met de procedures omschreven in de clausule creation
Deze procedures kan men oproepen via de **dot-operator**

**Dubbele-punt-operator:** geeft weer tot welke klasse het object behoort
***IkkeTeller: TELLER***

**!** Men geeft de intentie om het object te gebruiken (IkkeTeller)
--> het object is nog NIET aangemaakt

We gaan het object vervolgens maken met de **creation** clausule
--> we gebruiken een dot-operator
***create IkkeTeller.maken***

Ook verschillende “manipulaties” geven we weer met de dot-operator
***IkkeTeller.verhogen
IkkeTeller.verlagen
IkkeTeller.herzetten***

Ook het opvragen v/h resultaat is met een dot-operator
***IkkeTeller.stand***

Een object is volledig verplicht voor de attributen dat het moet onthouden
--> andere objecten kunnen deze lezen, maar niet meteen wijzigen

**OBJECTPRINCIPE 4
Attribuutwaarden kan men enkel wijzigen door routines gedefinieerd in de klasse**

Bij een onnatuurlijke wijziging v/e object
--> oorzaak zoeken in de procedures v/d klasse

Ingekapselde softwarecomponenten zijn modulair
--> hoge samenhang + geen sterke koppelingen vereist tussen componenten

**!** Objecten kunnen onderling geen gegevens van elkaar wijzigen
--> enkel **routines** kunnen dit wel

*Voorbeeld:*Object B bevat een attribuut x dat REAL-waarden kan aannemen
--> B.x = y is bijgevolg GEEN mogelijkheid want B.x kan REAL zijn, wat niet kan bij een y

Men kan de attribuutwaarde x wel aanpassen via een **procedure in de klasse** v/h object B
--> een object A kan enkel de waarde x aanpassen door het te vragen aan object B

**OBJECTPRINCIPE 5
Attributen worden beschreven met procedures die de attribuutwaarden kunnen aanpassen
--> zijn de enige procedures die deze waarden direct kunnen aanpassen**

Bijgevolg kunnen we zeggen dat een object een **duale constructie** is
 - bevat **gegevens** - bevat **routines**

**1.6 SYSTEMEN MET OBJECTEN**

Men bouwt een systeem als een verzameling (**assembly**)van objecten
--> werken samen of vervullen grote taken samen

Men werkt met een “verdeel en heers”-principe (**separation of concern**)

Elk systeem heeft nood aan een coördinatorobject (**root object** / **wortel object**)

**! Objecten kunnen pas werken als ze eerst gecreëerd zijn**--> men start met de creatie via een oerobject (**root object**)
Bij **Eiffel** moet je enkel de **root klasse** definiëren
--> dit is de coördinator v/h systeem en beschrijft het root object
Na het definiëren moet men het effectief maken via **creation** (of **create**)

*Voorbeeld:*Zie volgende pagina

*Voorbeeld:****Class Mijn\_systeem
creation
 maken
feature
 A: objecttype 1
 B: objecttype 2
feature maken is
 do
 create A.make
 create B.make
 A.taak1
 B.taak2
 end – maken
end – Class Mijn\_systeem***

***Zeer belangrijk***- Een **klasse** is steeds met een **hoofdletter**- De tekst na **–** is een “toelichting” en compleet irrelevant voor het programma
- **End-**attributen voorzien we het best v/e toelichting (overzicht)
- Elk systeem eindigt met een end-attribuut

Bij bovenstaan programma creëert men een object v/h type “Mijn\_systeem”
--> we gebruiken de routine “maken”
De routine zelf maakt twee objecten aan en geeft hen een taak

Men moet deze taken nog definiëren en aanmaken
**ZIE PAGINA 25 CURSUSTEKST**

Na het uitvoeren v/d taken v/e object verliest het zijn “nut” binnen het systeem
--> we gaan het geheugen “vrij maken” voor nieuwe objecten
**Garbage collector**

We kunnen objecten ook **persistent** maken, wanneer we het later gaan hergebruiken
--> “*opslaan op een secundair geheugen binnen het systeem”*

**ZIE PAGINA 26 SCHEMATISCHE VOORSTELLING BIJHOREND BIJ PAGINA 25**