2012-2013

# Klassen en Objecten

## In JAVA

Begrippen object en klassen
Implementatie in Java
Objectreferenties
Het begrip static
Enumeratie- of opsommingstype



Gilles Callebaut
KAHOSL

## 1.2 Begrippen object en klasse

Java is een **objectgeoriënteerde** programmeertaal.

### 1.2.1 Object

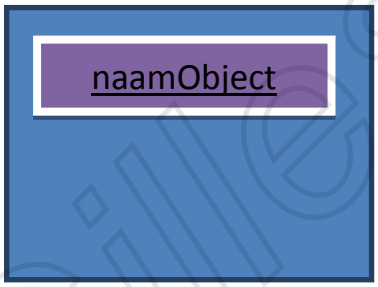
Een **object** is een op zich staand element dat een aantal eigenschappen of kenmerken heeft (*attributen*) en dat ontworpen is om een aantal specifieke taken (*operaties*) uit te voeren.

### 1.2.2 Klassen

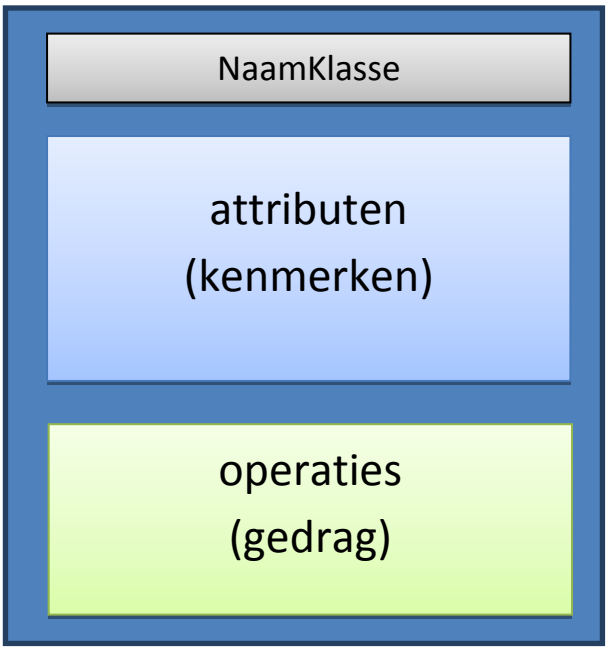
Een **klasse (*class*)** is een beschrijving van de *eigenschappen (attributen)* en *gedragingen (operaties)* van gelijkaardige objecten. Een klasse is dus een blauwdruk (blueprint) voor één of meerdere objecten.
De eigenschappen of kenmerken van een klasse noemt men **attributen**.
Het **gedrag** bepaalt wat een object van die klasse kan doen of doet als een ander object iets vraagt.

### 1.2.3 Voorstellen in UML

UML (UnifiedModeling Language) is een standaard om klassen en objecten voor te stellen.







## 1.3 Implementatie in Java

### 1.3.1 Definiëren van een klasse in Java

Een klasse is een soort nieuw datatype dat vastlegt hoe de gegevens worden opgeslagen en wat men ermee kan doen. Een klasse is een sjabloon voor elk object dat tot die klasse behoort.
Gedefinieerd: **class NaamKlasse**.
Tussen de accolades volgen de attributen en het gedrag.
**Methoden** moeten een actie uitdrukken en bevatten dus **werkwoorden**.
Op het ogenblik dat de klasse geschreven is, zijn de eigenschappen en de functionaliteiten van de objecten die tot die klasse zullen behoren gekend. Er zijn echter nog **geen objecten** aanwezig. Deze moeten nog expliciet gemaakt worden.

### 1.3.2 Creëren van een object in Java

Zelfde als een integer-variabele. De betekenis echter is totaal verschillend.
Er wordt niet direct geheugen voorzien. Er wordt ruimte voorzien voor een referentie.
Via de new-instructie wordt dus een object gecreëerd.

```
Rechthoek r = new Rechthoek();
```

Het creëren van een object gebeurt meestal in de *klantklasse* (*client class*).

### 1.3.3 Benaderen van de attributen en het gedrag

Benaderen vanuit lidmethoden

Lidmethoden zijn methoden die het gedrag implementeren, waarin de attributen rechtstreeks gemanipuleerd worden.

Benaderen vanuit de buitenwereld

De attributen en het gedrag van een object hangen steeds aan dat object vast en moeten dus steeds via dat object benaderd worden. Dit gebeurt aan de hand van de **.-operator**.

```
r1.lengte=…
```

### 1.3.4 Afschermen van informatie (public, private)

Nood aan data-hiding

**Data-abstractie**: het afschermen of verbergen van onnodige details.
Niet hoe het geïmplementeerd wordt is van belang.

**Data-hiding**: informatie of gegevens verbergen, maar nu met het del oneigenlijke gebruik tegen te gaan.

Implementatie in Java

De *toegangs-specificatie* schrijft men voor het attribuut of de methode die men wil afschermen.
Men moet een aantal methoden voorzien die de interactie tussen het object en de buitenwereld verzorgen. => toegangs-specificatie: **public** (interface-methoden)
Goede gewoonte: alle attributen en alle hulpmethoden al private te specifiëren.

```
private int attribuut1, attribuut2;
```

Nu kunnen we wel via de buitenwereld niet aan een attribuut vanuit een lidmethode (*membermethode*).
We sturen een boodschap naar het object met de vraag om het attribuut te veranderen.

- *Set-methode*

```
public voidsetMethode () {
    attribuut =         ;
}
```

- *Get-methode*

```
public intgetMethode () {
    return attribuut;
}
```

Elke communicatie gebeurt door het oproepen van *membermethoden*.

**Opmerkingen:**

- *Protected*en *package*. Default toegang is *package*. Een aantal klassen die bij elkaar horen kun je groeperen in een package.
- Ook op het niveau van de klasse kan men een toegang specifiëren.
  Schrijft men niet dan kunnen enkel de klassen binnen hetzelfde package die klasse gebruiken.

### 1.3.5 Constructor

In objectgeoriënteerde programmeertalen is een constructor een bepaald soort methode van een klasse die uitgevoerd wordt wanneer een object van de klasse wordt aangemaakt.

Een constructor is vergelijkbaar met een methode maar het levert geen waarde op en het kan niet overgeërfd worden. Constructors hebben vaak dezelfde naam als de klasse waartoe ze behoren. Het doel van een constructor is het initialiseren van de velden van een klasse en het vastleggen van een invariant voor de klasse (indien mogelijk, anders faalt het aanmaken van het object). Een correcte constructor levert een object in een 'geldige' toestand op.

```
public naamKlasse (int paramter1, int paramter2){
    attribuut1 = paramter1;
    attribuut2 = paramter2;
}
```

In de meeste programmeertalen is overloading van de constructor mogelijk: een klasse kan meerdere constructors hebben met dezelfde naam maar met verschillende parameters.

```
public naamKlasse (int paramter1, int paramter2){
    attribuut1 = paramter1;
    attribuut2 = paramter2;
}
public naamKlasse (){
    attribuut1 = 0;
    attribuut2 = 0;
}
```

*Defaultconstructor*: eenconstructor zonder paramters.

**Copy-constructor**: de attributen kunnen geïnitialiseerd worden op de waarden van een reeds bestaand object.

```
public naamKlasse (int paramter1, int paramter2){
    attribuut1 = paramter1;
    attribuut2 = paramter2;
}
//copy-constructor
public naamKlasse (naamObjectDatJeWilKopiëren oh){
    attribuut1 = oh.attribuut1;
    attribuut2 = oh.attribuut2;
}
```

## 1.4 Objectreferenties

### 1.4.1 this

*This*: dit is een referentie naar het object waarmee men tijdens de uitvoering van een programma mee bezig is.
De namen van de methode-paramters zijn (meestal) dezelfde als de attribuutnamen. Het gebruik van de naam binnen de methode verwijst dan steeds naar de methode-parameter en niet naar het attribuut. Men kan aan het attribuut door er **this.**voor te zetten.

```
public naamKlasse (int paramter1, int paramter2){
this.attribuut1 = paramter1;
this. attribuut2 = paramter2;
}
```

### 1.4.2 Toekennen en kopiëren

Bij Object o = new Object()
De variabelenaam o is **niet** het object als dusdanig, maar is een **verwijzing** of **referentie** naar.
Het toekennen van een objectreferentie aan een ander heeft dus gevolgen dat beiden naar hetzelfde object verwijzen. Wijzigingen zijn dus ook zichtbaar voor alle verwijzingen.

```
Object o1, o2;
01 = new Object(x,y);
O2 = O1;
```

Als men echter twee objecten wilt, met dezelfde inhoud.

- De *copy-constructor*

```
Public Object (Object o){
    attribuut1 = o.attribuut1;
    attribuut2 = o.attribuut2;
}
```

- Door een kopieer-methode te definiëren

```
Public Object kopieer (){
    Object temp = new Object (x,y);
    Return temp;
}
```

Met in de main:

```
o3= 01.kopieer;
// of
o3 = new Object (o1);
```

### 1.4.3 Object als methode-argument

Wanneer we een object doorgeven, geven we eigenlijk **objectreferenties** door via *call-by-value*.
*call-by-value* : Er wordt een lokale kopie gemaakt van de eigenlijke argumenten en de argumenten (in het oproepende blok) blijven onveranderd.
De objecten zelf kunnen wel gewijzigd worden, *call-by-reference* genaamd.

```
// declareren van een object
Public Object (int par1, int par2){
      attribuut1 = par1;
      attribuut2 = par2;
}
// call-by-reference
Publicvoidwijzigobject (int par1, int par2){
attribuut1 = par1;
attribuut2 = par2;
}
```

## 1.5 Het begrip static

### 1.5.1 Statische attributen

**Instantievariabelen**: de attributen leven en sterven samen met het object.

**Klassevariabelen**: De definitie van het attribuut wordt dan voorafgegaan door static (in een klasse). Waarbij deze niet gekoppeld is aan het object maar aan de klasse. D.w.z. dat de waarde ervan voor alle objecten van die klasse gelijk is, onafhankelijk van het aantal objecten.

```
publicstaticint/voidnaamKlasseVariabele (int par/    ){
return ;
…
}
```

Publieke statische variabelen worden normaal benaderd via *KlasseNaam.attribuut*.

### 1.5.2 Statische methoden

Statische methoden uit andere klassen: *KlasseNaam.methode*.

## 1.6 Herhalingsoefening 1

```
public class Punt {
private double x, y;
private static intaantalPunten = 0;
//defaultconstructor
public Punt() {
x = 0;
y = 0;
aantalPunten++;
}
// declareren Punt
public Punt (double xc, doubleyc){
aantalPunten++;
x = xc;
y= yc;
}
// copy-constructor
public Punt (Punt p){
aantalPunten++;
        x = p.x;
        y= p.y;
    }
// methode voor het wijzigen van het Punt
publicvoidwijzigPunt (double xc, doubleyc){
        x = xc;
        y= yc;
    }
//  get-methoden
publicdoublegetX (){
return x;
    }
public doublegetY (){
returny;
    }
//  methoden
public doubleberekenAfstandTotOorsprong(){
returnMath.sqrt(x*x +y*y);
    }
publicvoidschrijf(){
System.out.print(“ (“ + x + “ , “ + y + “) “);
    }
//  get-methode
publicstatic intgetAantalPunten(){
returnaantalPunten;
}
}
```

```
public classGebruikPunt{
public staticvoidmain( String[] args){
Punt p1 = new Punt(1,2);
Punt p2 = new Punt(p1);
    p1.schrijf();
p2.wijzigPunt (10,20);
S.O.P ( p2.get() );
    S.O.P (p1.berkenenAfstandTotOorsprong() );
    S.O.P ( Punt.getAantalPunten ());
}
}
```

## 1.7 Enumeratie- of opsommingstype

### 1.7.1 Inleiding

Een **enumeratie-type** of **opsommingstype** is een type dat bestaat uit een vaste verzameling van constante waarden.
Deze velden van het **enum-type** worden in Java steeds met HOOFDLETTERS geschreven omdat het constanten zijn.

### 1.7.2 Declaratie en gebruik

Declaratie gebeurt via het sleutelwoord *enum*.

```
public enum Naam{
CONSTANTE1, CONSTANTE2, CONSTANTE3
}
```

```
publicenum Planeet {
//diameter (km) en aantrekkingskracht (m/s²)
MERCURIUS (4878, 3.78)
…
AARDE (12756, 9.75)
…
public final double diameter;
public final doubleaantrekkingskracht;

public Planeet (double diameter, double aantrekkingskracht){
this.diameter = diameter;
this.aantrekkingskracht = aantrekkingskracht;
}
public double getDiameter(){
Return diameter;
}
public double getAantrekkingskracht(){
Returnaantrekkingskracht;
}
public double getGewichtObject(doublemassaObject){
Returnaantrekkingskracht*massaObject;
}
public double getMassaObject (doublegewichtObject){
ReturngewichtObject /aantrekkingskracht;
}
}
```

```
importjava.util.Scanner;
public class EnumTestPlaneten {
public staticvoidmain (String[] args) {
S.O.Pln("Geef uw gewicht op aarde");
Scanner sc = new Scanner(System.in);
double gewicht = sc.nextDouble();
double massa = Planeet.AARDE.getMassaObject(gewicht);
for (Planeet p: Planeet.values())
S.O.Pln ("jouw gewicht op "+p+" is "+p.getGewichtObject(massa));
}
}
```

# Karakters en Strings

## In JAVA

Het datatype char (karakters)
- Werken met karakters
- I/O van karakters

Strings
- Inleiding
- De klasse String
- De klasse StringBuffer

Commmand-line argumenten

Gilles Callebaut
KAHOSL

## 2.1 Het datatype char (karakters)

Java werkt met de *Unicode*, maar om de overeenkomst met de ASCII-standaard te bewaren komen de eerste 256 karakters van de Unicode overeen met deze van de ASCII-standaard.
Enkele belangrijke waarden zijn:

- '0' → '9': 48 → 57
- 'A' → 'Z': 65 → 90
- 'a' → 'z': 97 → 122

Merk op dat het verschil in ASCII-waarde tussen de correspondeerde hoof- en kleineletters steeds 32 bedraagt.

### 2.1.1 Gebruik van karakters binnen Java

In Java wordt voor karakters het primitieve datatype **char** gebruikt.

Literals, variabelen en constanten

Voorstellen van speciale karakters: Om niet met de getalwaarde te moeten werken heeft men hiervoor *escape sequenties* voorzien. Deze beginnen steeds met een \.

| ESCAPE SEQUENTIE | KARAKTER WAARDE |
|---|---|
| \b | backspace |
| \t | Horizontale tab |
| \n | newline |
| \f | form feed |
| \r | carriage return |
| \" | Dubbel aanhalingsteken |
| \' | Enkel aanhalingsteken |
| \\ | backslash |

De tekens die niet op het toetsenbord voorkomen moet men gebruiken via hun Unicode-waarde: \u gevolgd door hexadecimale notatie van de Unicodewaarde.

### 2.1.2 Werken met karakters

Relationele operatoren

Omdat karaktervariabelen binnen de Unicode in alfabetische volgorde gedefinieerd zijn, kunnen karaktervariabelen door het gebruik van relationele operatoren in alfabetische volgorde gesorteerd worden.

'a' == 'a'

'a' < 'b'

'A' < 'a'

De klasse Character

```
BooleanisCijfer =       Character.isDigit(ch);
BooleanisKleineLetter   =       Character.isLowerCase(ch);
BooleanisHoofdLetter    =       Character.isUpperCase(ch);
charKleineLetter        =       Character.toLowerCase(ch);
charHoofdLetter         =       Character.toUpperCase(ch);
```

Conversie char-int

```
Public classKarakterIntegerCasting{
Public staticvoidmain (String[] args){
// declareren karakter
Char c1 = '5';
// omzetten karakter naar getal waarde 5
Int getal = c1 – '0';

// declareren karakter
Char c2 = 'b';
// karakter b vervangen door de letter 3 plaatsen verder in het alfabet
        C2 = (char) (c2+3);
    }
}
```

### 2.1.3 I/O van karakters

De klasse Scanner voorziet geen methode om 1 karakter van de input te halen.

```
BufferReaderbr = newBufferReader (newInputStreamReader (System.in));
charch = br.read();
```

Inlezen van het enter-teken:

- Windows carriage return \r
  linefeed \n
- Mac carriage return \r

In java Virtuele Machine:

```
String eol = System.getProperty("line.separator");
```

## 2.2Strings

### 2.2.1Inleiding

Een **String**object is een object waarvan de waarde na creatie nooit meer kan veranderen. Wil men een tekst manipuleren (karakters wijzigen, toevoegen,…) dan heeft men een **StringBuffer**object nodig.

### 2.2.2 De klasse String

Een **string literals** wordt steeds gekenmerkt door een aantal karakters omsloten door dubbele aanhalingstekens.

Creatie van objecten

```
String str = “Ik maak nu een string.”;
```

Java laat *str* verwijzen naar het automatisch gemaakte object.
De klasse String bevat een hele reeks constructoren:

- String ()
  - Creëren van een lege string
- String(char[])
  - Creëren van een string met dezelfde karaktersequentie als in de karakterrij.
- String(String)
  - Creëren van een string met dezelfde karaktersequentie als in de actuele parameterstring
- String(StringBuffer)
  - Creëren van een string met dezelfde karaktersequentie als in de actuele parameterstringbuffer

**Methoden die inwerken op een String**

| Return-type | methodeNaam(parameter) |
|---|---|
| char | charAt(**int** index)<br>// Karakter op de plaats van de index |
| int | compareTo (String andereString)<br>//vergelijkt twee strings lexicografisch (alfabetisch). Het resultaat is negatief als het object alfabetisch voor het parameterobject komt en positief als het omgekeerde geldt. Als beiden identiek, is het resultaat nul. |
| int | compareToIgnoreCase(String andereString)<br>//vergelijk beide strings maar hou geen rekening met hoofdlettergevoeligheid |
| String | IndexOf(**verscheidene mogelijkheden**)<br>• **Int**ch<br>• **Int**char, **int**fromindex<br>//fromindex wil zeggen vanaf welke positie hij mag beginnen zoeken<br>• **String** str<br>• **String** str, **int**fromindex<br>// geeft de positie weer van het eerst voorkomende karakter in een string (hoe reking mee dat positie 0 ook bestaat)<br>lastIndexOf , analoog maar geeft het laatst voorkomende positie. |
| Int | length |
| String | Replace (**char**oldChar, **char**newChar) |
| String | Substring(**int**beginIndex) |
| String | Substring(**int**beginIndex, **int** endIndex)<br>//extrageerd de karakters in een string tussen de twee paramters, en geeft een nieuwe substring terug |
| String | toLowerCase |
| String | toUpperCase |

Dit zijn member-methoden en geen, statische methoden.

De statische methode **valueOf**

| Return-type | methodeNaam(parameter) |
|---|---|
| String | valueOf(**verscheidene mogelijkheden)**<br>• **boolean** b<br>• **char** c<br>• **double** d<br>• **float**f<br>• **int** i<br>• **long** l<br>//weergeven van de stringvoorstelling van de parameter |

Conversiemethoden

| Return-type | methodeNaam(parameter) |
|---|---|
| int | Integer.parseInt(**String** s) |
| long | Long.parseLong(**String** s) |
| double | Double.parseDouble(**String** s) |

//conversie van een String naar een getalwaarde.

| | | |
|---|---|---|
| Dit is niet altijd geldig, | NumberFormatException | andere karakters bevat |
| | NullPointerException | geen object bij de stringreferentie |

Input/output

**Niet-grafisch**

*System.out.print()* heeft de paramter van het type String. Dei worden automatisch geconverteeerd en samengesmolten tot één stringobject door + operatoren.

| | | |
|---|---|---|
| Voor de **input** in de klasse Scanner: | next() | inlezen 1 woord |
| | nextLine() | inlezen 1 lijn |

met hasNext of hasNextLine kunnen we weten of er nog een woord of lijn in de input is.

### 2.2.3 De klasse StringBuffer

Niet alleen de karakters van de inhoud, maar ook de lengte kunnen gewijzigd worden.

**Creatie van StringBufferobjecten**

we beschikken over drie constructoren:

- StringBuffer()
  - bevat geen karakters en een initiële capaciteit van 16 karakters
- StringBuffer(**int** lengte)
  - bevat geen karakters en een initiële capaciteit van *lengte* karakters
- StringBuffer(**string**str)
  - bevat dezelfde karaktersequentie als de parameterstring en een initiële capaciteit van 16 + de lengte van de string

2 indicaties voor de grootte:

- capaciteit
  - aantal karakters kan bevatten vooraleer nieuw geheugen gealloceerd moet worden
- lengte
  - effectieve lengte (aantal karakters als inhoud)

Het StringBufferobject:

```
StringBufferstrbuf = newStringBuffer("Hallo");
```

**Methoden**

Opvragen van:

| capaciteit | lengte |
|---|---|
| strbuf.capacity() | strbuf.length() |

Converteren van een StringBuffernaart een String **ToString()**
iets toevoegen in een StringBuffer **append()**
iets toevoegen vanaf een positie in een StringBuffer **insert()**

| Return-type | methodeNaam(parameter) |
|---|---|
| char | charAt(**int** index) |
| void | seCharAt(**int** index, **char** c) |
| StringBuffer | delete(**int** start, **int** end) |
| StringBuffer | Replace(**int** start, **int** end, **String**str) |
| String | substring(**int**beginIndex, **int** endIndex) |
| StringBuffer | reverse |

## 2.3 Command-line argumenten

Wanneer me neen Java-programma start, kan men een aantal argumenten meegeven.
Voor deze command-line argumenten wordt er dan in het programm een array van Strings gemaakt die net groot genoeg is om alle argumenten te bavatten.

Via de main kun je aan deze argumenten:

```
public classProgrammaMetCommandLineArguments {
public staticvoidmain (String[] args){
        S.O.P (“aantal argumenten: “ + args.length);
for (int i: args){
            S.O.P (args[i])
}
```

(die for-each lus kan fout zijn als je de juiste weet weet het mij te zeggen)

## 2.4Voorbeelden

### 2.4.1Tekst achterstevoren weergeven

```
String tekst = “teskt”;
Int lengte = tekst.length();
StringBufferstrBuf = newStringBuffer(lengte);
For (int i=lengte-1; i<=0;i--) strBuf.append(tekst.charAt(i));
```

Via de reverse-methode:

```
strBuf.reverse();
```

### 2.4.2 Encryptie: monoalfabetische substitutie

Bij het encrypteren wordt elke letter vervangen door een andere letter. De lijst met de koppelingen tussen de originele en de vervangingsletters wordt bijgehouden in de substitutietabel.

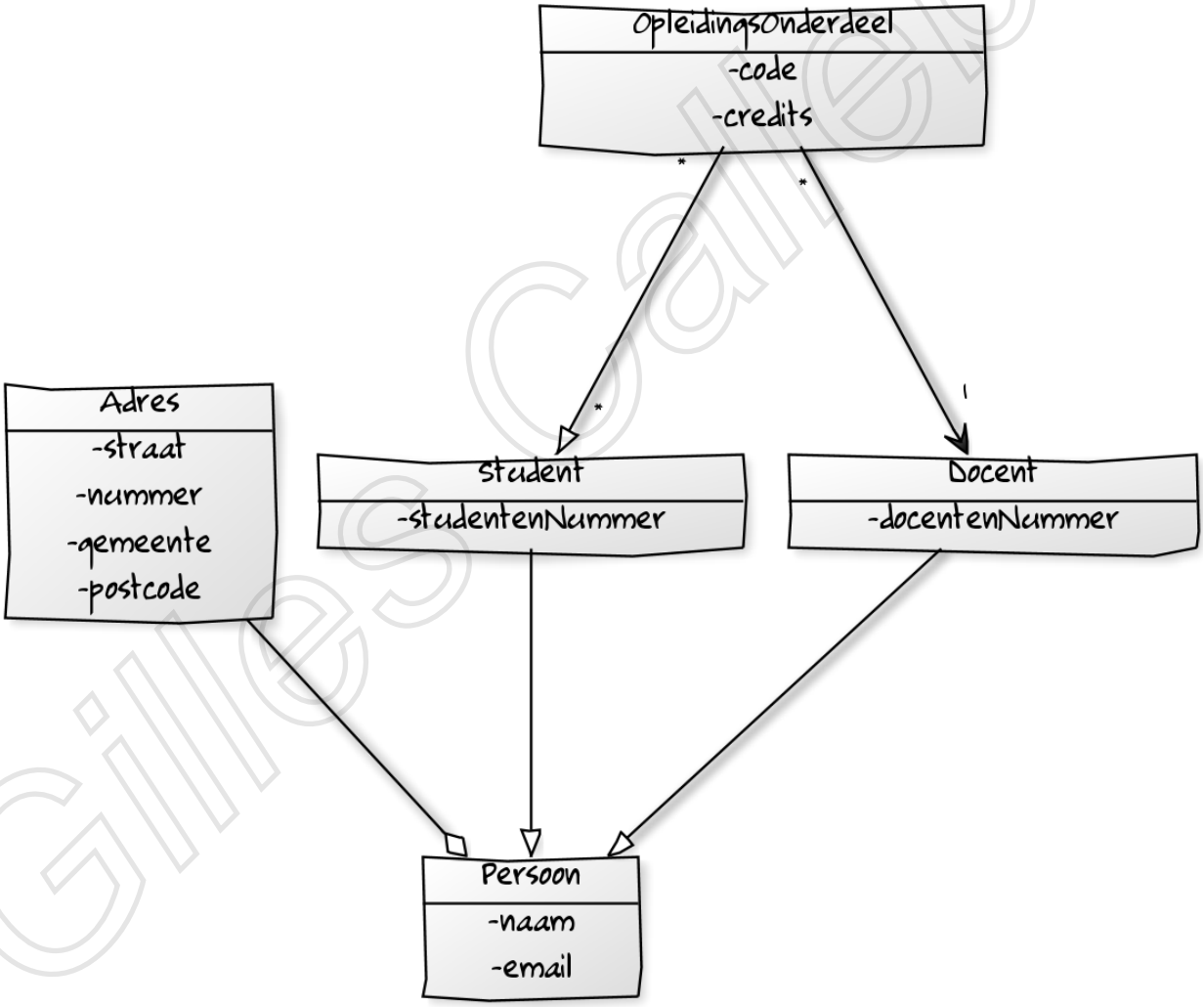
```
importjava.util.Random;
publicclassEncryptie {

	privateint [] substitutieTabel;
	privatebooleangetalNietInRij(int positie, int getal){
		boolean uniek =true;
		int i=0;
		while((i<positie) && uniek)
		{
			uniek = substitutieTabel[i] != getal;
			i++;
		}
		return uniek;
	}
	privatevoidgenereerTabel()
	{
		Random random = new Random();
		int getal;
		int i=0;
		while (i<substitutieTabel.length)
		{
			getal =random.nextInt(substitutieTabel.length);
			if (getalNietInRij(i,getal))
			{
				substitutieTabel[i++] = getal;
			}
		}
	}
	public Encryptie()
	{
		substitutieTabel= newint[26];
				genereerTabel();
	}
	publicString codeer (String origineel)
	{
		StringBuffersb = newStringBuffer(origineel.length());
		char c;
		for(int i=0;i<origineel.length();i++)
		{
			c =origineel.charAt(i);
			if (Character.isLowerCase(c))
			{
				c = (char) (substitutieTabel[c-'a'] +'a');
			}
			elseif(Character.isUpperCase(c))
			{
				c = (char) (substitutieTabel[c - 'A'] + 'A');
			}
			sb.append(c);
		}
		returnsb.toString();
	}

}
```

```
importjava.util.Scanner;
publicclassMain{
	publicstaticvoidmain(String[] args){
		Scanner sc= new Scanner(System.in);
		String origineel,gecodeerd;
		Encryptieencryptie = newEncryptie();
		while (sc.hasNextLine())
		{
			origineel = sc.nextLine();
			gecodeerd = encryptie.codeer(origineel);
			System.out.println(gecodeerd);
				}
	}
}
```

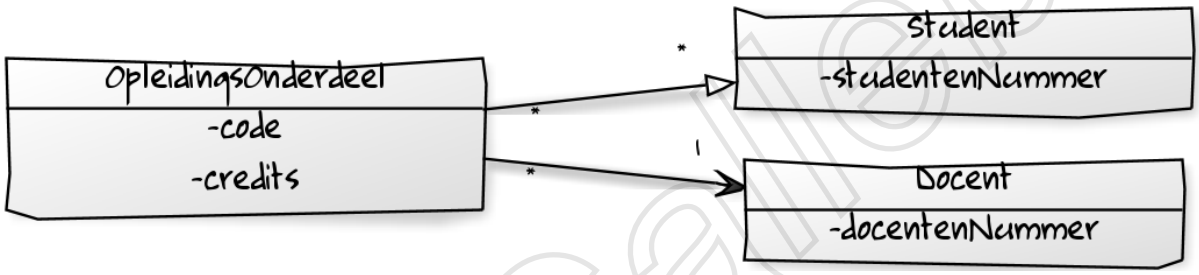
# H3: Relaties tussen klassen



## Associatie

Een associatie kan bidirectioneel zijn, in twee richtingen hebben ze een associatie.
unidirectioneel zijn, in één richting hebben ze een associatie.

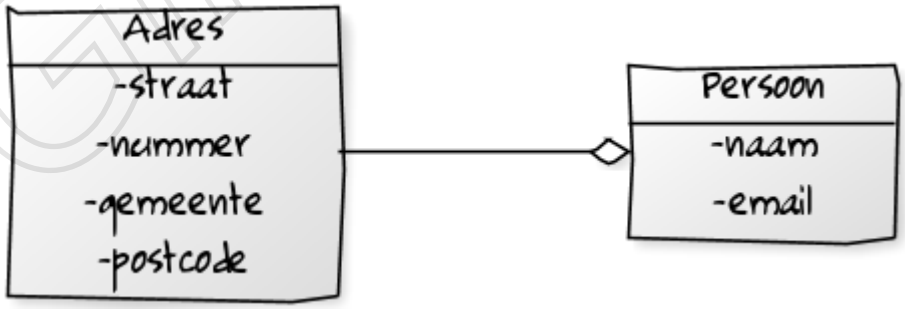
Een associatie in Java wordt vertaalt naar een attribuut in de klasse.
Bij bidirectionele staan er in beide klassen attributen.




## Aggregatie

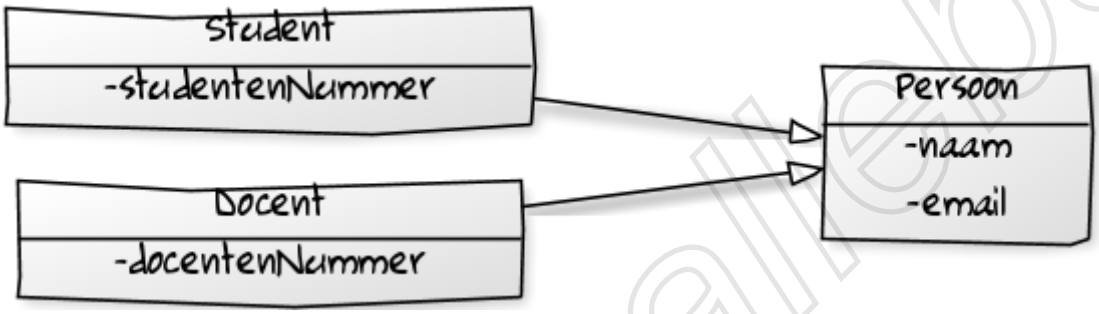
Dit noemt men een *heeft-een-relatie*.
De ene klasse is een (onder)deel van de andere.

Er bestaat een sterkere versie van een aggregatie, nl compositie, die leeft en sterft met het geheel.



# Specialisatie/generalisatie/overerving

Het is mogelijk om een klasse te schrijven die een aantal eigenschappen van een andere klasse(de basis- of superklasse) overneemt of erft.
In Java duidt men overerving aan via het sleutelwoord **extends**.




# Abstracte klassen en interfaces

## Abstracte klassen

Men kan geen object creëren van abstracte klassen. Abstracte klassen worden enkel gebruikt om van af te leiden.

## Interfaces

Interfaces specifiëren enkel welke methoden de klassen die de interfaces implementeren zeker zullen bevatten.
De klasse implementeert een interface en is dus verplicht om die methodes te implementeren.

# Klasse object

## toString-methode

```
public String toString()
      {return""+ietsWatGeenStringIsEnJeWelWiltUitPrinten;}
```

## Equals

dit is te complex en valt buiten de scope van de cursus

## Packages

Klassen uit andere pakketten (packages) moeten worden geïmporteerd worden via een **import**-instructie.