



samen sterk voor werk



Deze cursus is eigendom van de VDAB©

## Inhoudsopgave

<b>1</b>	<b>INLEIDING.....</b>	<b>3</b>
1.1	Doelstelling.....	3
1.2	Vereiste voorkennis.....	3
1.3	Nodige software .....	3
1.4	Algemeen .....	3
1.4.1	Waterval methode.....	3
1.4.2	Iteratieve methode .....	4
<b>2</b>	<b>USE-CASES .....</b>	<b>5</b>
2.1	Actor .....	5
2.2	Het diagram.....	5
2.3	StarUML.....	6
2.4	Use-case in tekstvorm .....	7
2.5	Tekstverwerker.....	7
2.6	Verbanden tussen use-cases .....	8
2.7	User stories.....	9
<b>3</b>	<b>CLASS DIAGRAM .....</b>	<b>10</b>
3.1	StarUML.....	11
3.2	Attributen .....	11
3.3	StarUML.....	11
3.4	Attributen met class bereik .....	12
3.5	Methods en constructors .....	12
3.6	StarUML.....	12
3.7	Associaties tussen classes .....	13
3.8	Aggregatie .....	15
3.9	Compositie .....	15
3.10	StarUML.....	15

3.11	Associatieclass .....	16
3.12	StarUML.....	16
3.13	Inheritance .....	17
3.14	StarUML.....	18
3.15	Abstract method .....	18
3.16	Interface .....	19
3.17	StarUML.....	19
3.18	Enum .....	19
3.19	StarUML.....	20
3.20	Rollen.....	20
<b>4</b>	<b>SINGLE RESPONSIBILITY .....</b>	<b>25</b>
4.1	StarUML.....	25
<b>5</b>	<b>SEQUENCE DIAGRAM .....</b>	<b>26</b>
5.1	Conditionele message .....	26
5.2	StarUML.....	26
<b>6</b>	<b>STATE CHART DIAGRAM.....</b>	<b>28</b>
6.1	StarUML.....	28
<b>7</b>	<b>STAPPENPLAN .....</b>	<b>30</b>
<b>8</b>	<b>COLOFON.....</b>	<b>31</b>

# 1 INLEIDING

## 1.1 Doelstelling

Je leert werken met UML: een taal waarmee je de analyse van je applicatie beschrijft.

## 1.2 Vereiste voorkennis

- Een object georiënteerde programmeertaal

## 1.3 Nodige software

- StarUML versie 1 (te downloaden op <http://staruml.sourceforge.net/v1>)  
We gebruiken niet StarUML 2, omdat die ook in een leeromgeving te betalen is.

## 1.4 Algemeen

UML is een afkorting van Unified Modeling Language.

UML definieert een aantal diagrammen met gestandaardiseerde symbolen.

Je beschrijft met deze diagrammen de analyse van een applicatie. Als iedereen dezelfde diagrammen en dezelfde symbolen gebruikt, is het gemakkelijk elkaars analyses te lezen.

UML definieert niet met welke methode (werkwijze) je een applicatie maakt (volgorde van analyse, implementatie, testen, ...). Twee methoden zijn

- Waterval methode (vroeger vooral gebruikt)
- Iteratieve methode (nu vooral gebruikt)

### 1.4.1 Waterval methode

Bij de waterval methode doe je eerst de volledige analyse van het project, daarna de volledige implementatie in code, daarna het testen van het volledige project.

Een project van 1 jaar kan dus bestaan uit 3 maanden analyse, gevolgd door 6 maanden implementatie, gevolgd door 3 maanden testen.

Het kan gebeuren dat je tijdens de implementatie tot nieuwe inzichten komt waarbij je de analyse aanpast, maar dit zou minimaal moeten zijn. Het kan ook gebeuren dat je tijdens het testen code aanpast, maar ook dit zou minimaal moeten zijn.

Het grote nadeel van de waterval methode is dat je tijdens het project moeilijk kan zeggen of het project volgens plan verloopt: pas bij het coderen merk je fouten in de analyse, pas bij het testen merk je fouten in de code. Pas nadat het project helemaal "af" is lever je het aan de eindgebruiker. Deze kan pas dan laten weten dat er door communicatiestoornissen nog fouten waren in de analyse, dus ook in de implementatie.

Grafische voorstelling van de waterval methode:



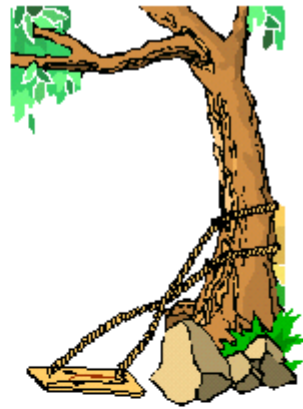
De volgende afbeelding toont het probleem op een grappige manier:



Wat de adviseur voorstelde



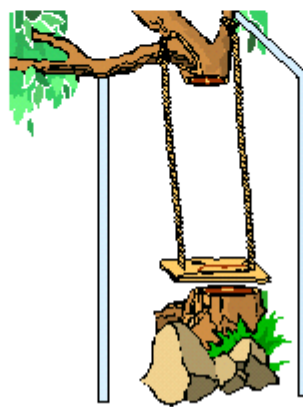
Wat de ontwerper ervan maakte



Hoe de analist het beschreef



Wat de programmeur ervan maakte



Hoe de implementator het aan de praet kreeg



Wat de klant eigenlijk wou

#### 1.4.2 Iteratieve methode

Bij de iteratieve methode verdeel je het project in deelprojecten.

Je maakt in ieder deelproject enkele programma onderdelen. De onderdelen die dringendst en/of belangrijkst zijn voor de eindgebruiker komen eerst aan bod. In een voorbeeld van een tijdsregistratiesysteem kunnen volgende onderdelen voorkomen, in volgorde van dringendheid

- Registreren van begin- en einduur van iedere werkdag
- Maandwedde berekenen
- Statistieken over gepresteerde uren
- Statistieken over overuren

Je doet in het eerste deelproject (ook sprint genoemd) de analyse, implementatie en test van het eerste onderdeel. Je kan daarna dit onderdeel al aan de eindgebruiker opleveren, zodat je snel feedback krijgt. Je doet in de tweede sprint de analyse, implementatie en test van het tweede onderdeel. Je kan daarna dit onderdeel al aan de eindgebruiker opleveren, zodat je snel feedback krijgt. Je doet in de laatste sprint de analyse, implementatie en test van de laatste onderdelen. Je zorgt er voor dat elke sprint even lang duurt (gemiddeld drie weken) Het aantal onderdelen dat je in één sprint kan uitvoeren kan daarom verschillen van sprint tot sprint. Deze methode wordt ook wel de Scrum methode genoemd.

Grafische voorstelling van de iteratieve methode:



## 2 USE-CASES

Je gaat bij het maken van use-cases uit van de gebruiker van je applicatie.

Use-cases zijn behoeften (requirements) van de gebruiker in je applicatie.

Een use-case wordt uiteindelijk een onderdeel van je applicatie.

Een tijdsregistratiesysteem kan volgende use-cases bevatten:

- Nieuwe werknemer registreren
- Uren registreren
- Maandwedde berekenen
- Statistieken gepresteerde uren
- Statistieken overuren
- Statistieken voor overheid

Use-cases beschrijven wat de gebruiker van je applicatie wenst, niet hoe je dit technisch uitwerkt.

Een use-case bevat geen technische vereisten (snelheid, geheugengebruik, ...) en geen informatica technische woorden. De eindgebruiker kan dus de use-cases lezen.

Dit is zeer interessant om feedback te krijgen van de eindgebruiker.

### 2.1 Actor

Een actor is een persoon of een extern programma die een behoefte heeft in je applicatie.

De naam van een actor is geen specifieke persoon (bvb. Mieke), maar een rol die een persoon speelt (bvb. Werknemer)

Om de actoren te vinden kan je volgende vragen stellen:

- Welke gebruikersgroepen hebben de hulp van je applicatie nodig bij het uitvoeren van hun taken ?
- Welke gebruikersgroepen zijn nodig om de hoofdtaken in je applicatie uit te voeren ?
- Welke gebruikersgroepen zijn nodig om de secundaire taken (onderhoud, administratie) in je applicatie uit te voeren ?
- Met welke andere applicaties communiceert je applicatie ?

### 2.2 Het diagram



- De rechthoek stelt de applicatie voor.
- De poppetjes stellen actors voor.  
Je plaatst de actors buiten de rechthoek van de applicatie.
- De ovalen stellen use-cases voor.  
Er wordt aangeraden dat een use-case bestaat uit 1, 2 of 3 woorden.  
Je plaatst de use-cases binnen de rechthoek van de applicatie.
- De lijnen van de actors naar de use-cases stellen voor  
welke actors welke use-cases zullen uitvoeren.

Volgende vragen helpen je bij het vinden van use-case:

- Wat zijn de belangrijke taken die een actor in de applicatie wenst uit te voeren ?
- Zal de actor data in de applicatie maken, opzoeken, wijzigen of verwijderen ?
- Moet de actor aan de applicatie een gebeurtenis laten weten ?
- Moet de actor op de hoogte gebracht worden van een gebeurtenis in de applicatie ?

## 2.3 StarUML

1. Je start StarUML.
2. Je kiest Default Approach en je kiest OK.
3. Je ziet in de Diagram Explorer (rechts) een plus teken voor UseCase Diagrams.  
Je klapt dit open en je dubbelklikt het diagram Main dat zich hier bevindt.
4. Je kiest links in de Toolbox een System Boundary  
en je klikt in de gespikkelde oppervlakte van het diagram (midden in het venster)
5. Je kiest Actor in de Toolbox en je klikt in het diagram,  
buiten de rechthoek van de applicatie. Je wijzigt de naam naar Werknemer.
6. Je doet hetzelfde voor een Actor Beheerder.
7. Je doet hetzelfde voor een Actor Overheid.
8. Je kiest UseCase in de Toolbox en je klikt in het diagram, binnen de rechthoek  
van de applicatie. Je wijzigt de naam naar Nieuwe werknemer registreren.
9. Je doet hetzelfde voor de use-case Uren registreren.
10. Je doet hetzelfde voor de use-case Statistieken gepresteerde uren.
11. Je doet hetzelfde voor de use-case Statistieken overuren.
12. Je doet hetzelfde voor de use-case Statistieken overheid.
13. Je kiest Association in de Toolbox en je sleept van de actor Beheerder  
naar de use-case Nieuwe werknemer registreren.
14. Je doet hetzelfde tussen Werknemer en Uren registreren.
15. Je doet hetzelfde tussen Werknemer en Statistieken gepresteerde uren.
16. Je doet hetzelfde tussen Werknemer en Statistieken overuren.
17. Je doet hetzelfde tussen Overheid en Statistieken overheid.
18. Je slaat het project op.



Tip: je kan een onderdeel verwijderen uit een diagram door het te selecteren en op de Delete toets te drukken. Dit onderdeel is nog niet verwijderd uit het project. Als je een onderdeel wil verwijderen uit het diagram én uit het project geef je een rechtermuisklik op dit onderdeel en je kiest Select In Model Explorer. Je geeft daarna een rechtermuisklik op dit onderdeel in de Model Explorer en je kiest Delete From Model.

## 2.4 Use-case in tekstvorm

Je maakt per use-case een tekst die de use-case in detail beschrijft.

Iedere tekst heeft eenzelfde opbouw:

- Naam: naam van de use-case die je ook in het use-case diagram vermeldt.
- Samenvatting: in één of twee zinnen.
- Actor(en) die de use-case zullen uitvoeren.
- Voorwaarden: condities waaraan moet voldaan worden om de use-case te kunnen starten.
- Beschrijving: een genummerde lijst met de individuele stappen in de use-case.
- Alternatief: stappen die uitzonderlijk uitgevoerd worden in de use-case.
- Resultaat: het resultaat van het uitvoeren van de use-case als deze succesvol is doorlopen.

Voorbeeld:

<b>Naam</b>
Nieuwe werknemer registreren
<b>Samenvatting</b>
Een nieuwe werknemer registreren.
<b>Actor</b>
Beheerder
<b>Voorwaarden</b>
De beheerder is ingelogd.
<b>Beschrijving</b>
<ol style="list-style-type: none"><li>1. De beheerder tikt de gegevens van de nieuwe werknemer.</li><li>2. De applicatie valideert deze gegevens.</li><li>3. De applicatie slaat de nieuwe werknemer op.</li></ol>
<b>Alternatief</b>
2a. Als reeds een werknemer met het ingetikte rijksregisternummer voorkomt, toont de applicatie een foutmelding.
<b>Resultaat</b>
De werknemer is opgeslagen.



Het nummer bij alternatief (2a) verwijst naar een bijbehorende nummer in beschrijving (2).

## 2.5 Tekstverwerker

Je start je tekstverwerker en je tikt (of kopieert) de tekst van de use-case

Nieuwe werknemer registreren. Je slaat dit document op.

Je koppelt dit document aan de bijbehorende use-case in StartUML:

1. Je selecteert de use-case Nieuwe werknemer registreren in het use-case diagram.
2. Je kiest het tabblad Attachments (rechts onder).
3. Je kiest de knop .
4. Je kiest de knop .
5. Je selecteert het document dat je maakte met je tekstverwerker.
6. Je kiest Open, OK

Telkens je het document dubbelklikt in het venster Attachments, wordt dit document geopend in je tekstverwerker.



Ander voorbeeld: geld overschrijven van een spaarrekening naar een zichtrekening.

**Naam**

Overschrijven van spaarrekening naar zichtrekening

**Samenvatting**

Een bedrag overschrijven van een spaarrekening naar een zichtrekening

**Actor**

Klant

**Voorwaarden**

De klant is ingelogd.

**Beschrijving**

1. De klant kiest de spaarrekening.
2. De klant kiest de zichtrekening.
3. De klant tikt het bedrag.
4. De klant bevestigt.

**Alternatief**

4a. Als het saldo op de spaarrekening kleiner is dan het over te schrijven bedrag, toont de applicatie een foutmelding.

**Resultaat**

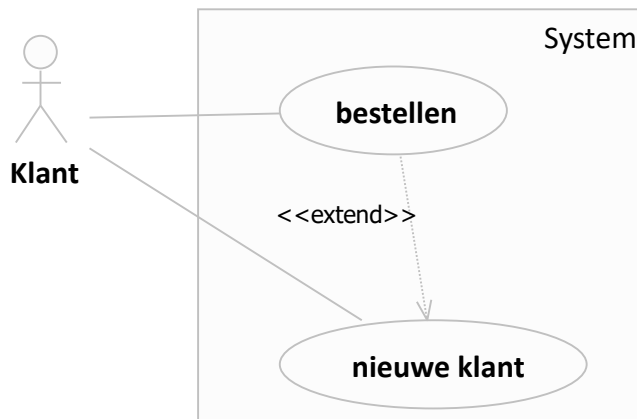
Het saldo op de spaarrekening is verminderd met het bedrag.  
Het saldo op de zichtrekening is vermeerderd met het bedrag.

## 2.6 Verbanden tussen use-cases

Als tijdens de use-case A *altijd* een use-case B wordt opgeroepen, is er een include verband tussen deze use-cases. Voorbeeld in een bank: bij de use-case geld afhalen wordt altijd de use-case rekening zoeken uitgevoerd:



Als tijdens de use-case A *optioneel* een use-case B wordt opgeroepen, is er een extend verband tussen deze use-cases. Voorbeeld in een webshop: bij het bestellen wordt de klant optioneel als nieuwe klant toegevoegd:



## 2.7 User stories

Een user story is een korte beschrijving (één zin ) van de use-case, geschreven door de eindgebruiker. De user stories dienen om in te schatten hoeveel use-cases (user stories) je zal maken in het huidige deelproject (sprint).

De zin die een user story beschrijft heeft volgende vorm:

Als *actor* wil ik *doel*

Voorbeelden:

- Als beheerder wil ik nieuwe werknemers registreren.
- Als werknemer wil ik statistieken over mijn overuren.



Use-cases: zie takenbundel

### 3 CLASS DIAGRAM

Het class diagram beschrijft welke classes en interfaces je nodig hebt om de te automatiseren werkelijkheid in je applicatie uit te werken.

Het class diagram beschrijft ook de relaties tussen deze classes en interfaces.

Voorbeelden van class categorieën en classes:

Categorie	Voorbeeldclasses
Tastbare objecten	Kassa, autobus, schaakstuk
Specificaties	Productspecificatie, vluchtplan
Plaatsen	Winkel, parkeerplaats
Handelingen	Reservatie, betaling
Rollen	Klant, leverancier
Container	Winkelrek, parking
Externe applicatie	Kredietkaartautorisatiesysteem

Je onderzoekt de werkelijkheid om deze classes te vinden.

Ieder zelfstandig naamwoord uit de werkelijkheid kan een class of een attribuut van een class zijn.

Een zelfstandig naamwoord is een attribuut van een class als dit woord

- tekst voorstelt (bvb. voornaam, familienaam, productnaam, ...)
- een getal voorstelt (bvb. aantal kinderen, btw percentage, ...)
- een ja/nee waarde voorstelt (bvb. gehuwd, ...)
- een datum of tijd voorstelt (bvb. geboortedatum, factuurdatum, ...)

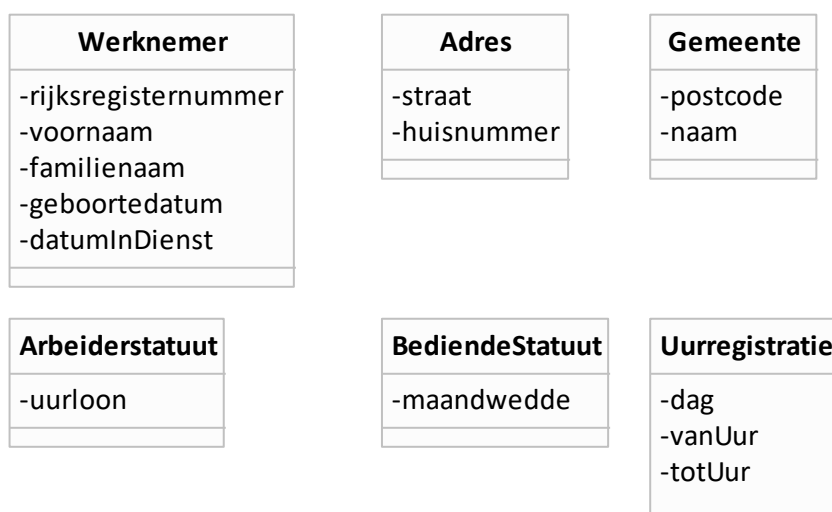
De andere zelfstandige naamwoorden worden classes.

Je vindt deze zelfstandige naamwoorden in beschrijvingen van de werkelijkheid (documenten, bestellingen, facturen, contracten, use-cases in tekstvorm, ...)

Je stelt een class voor als een rechthoek met drie compartimenten:

- in het bovenste compartiment de naam van de class
- daar onder een compartiment met de attributen van de class
- daar onder een compartiment met de methods van de class


Voorbeelden in de tijdsregistratie applicatie:



Je ziet in dit diagram nog geen verbanden tussen classes. Deze komen later.

Het compartiment met de attributen en/of het compartiment met de methods wordt soms weggelaten om een eenvoudiger overzicht te verkrijgen.

### 3.1 StarUML

1. Je ziet in de Diagram Explorer (rechts) een plus teken voor Class Diagrams.  
Je klapt dit open en je dubbelklikt het tweede diagram Main dat zich hier bevindt.
2. Je kiest links in de Toolbox een Class  
en je klikt in de gespikkelde oppervlakte van het diagram (midden in het venster)
3. Je wijzigt de naam van de class naar Werknemer.
4. Je dubbelklikt in het bovenste compartiment van de class en je kiest de knop 
5. Je tikt -rijksregisternummer en je drukt Enter.

Je maakt op deze manier alle classes zoals hierboven weergegeven.

### 3.2 Attributen

Het minteken voor elk attribuut duidt aan dat dit een *private* variabele wordt in de class.

Voor Java ontwikkelaars: getXXX en setXXX methods rond de private variabele worden verondersteld en vermeld je niet. Zo blijft het diagram overzichtelijk.

Voor C# ontwikkelaars: de property (met get en set) rond de private variabele wordt verondersteld en vermeld je niet. Zo blijft het diagram overzichtelijk.

Een attribuut kan meerdere keren voorkomen in een class.

Een werknemer kan bijvoorbeeld meerdere voornamen hebben:

Werknemer
-rijksregisternummer
-voornamen[1..*]
-familiennaam
-geboortedatum
-datumInDienst

[1..\*] betekent één of meerdere keren.

[\*] betekent nul of meerdere keren.

[3] betekent 3 keren.

[1..4] betekent tussen 1 en 4 keren

Je kan van elk attribuut ook het type vermelden:

Werknemer
-rijksregisternummer: long
-voornamen: String[1..*]
-familiennaam: String
-geboortedatum: Date
-datumInDienst: Date

Opmerking: Als je dit uitwerkt in Java code wordt Date vervangen door LocalDate.

### 3.3 StarUML

1. Je kiest de Model Explorer (rechts).
2. Je selecteert in de class Werknemer het attribuut rijksregisternummer
3. Je selecteert Properties (onder de Model Explorer)  
Als je dit venster niet ziet, gebruik je ◀ ▶ rechts onder in StarUML.
4. Je tikt long bij Type.
5. Je selecteert het attribuut voornaam
6. Je selecteert Properties
7. Je tikt voornamen bij Name
8. Je kiest 1..\* bij Multiplicity

### 3.4 Attributen met class bereik

Bij gewone attributen heeft ieder object zijn eigen versie van een attribuut. Elke werknemer heeft bijvoorbeeld zijn eigen familienaam. Hij deelt deze familienaam niet met de andere werknemers.

Daarnaast heb je ook attributen met class bereik. Zo'n attribuut wordt *gedeeld* door alle objecten van de class. In een programmeertaal geef je dit aan met het keyword `static`.

Je onderlijnt attributen met class bereik.

Voorbeeld: alle arbeiderstatuten delen hetzelfde arbeidsreglement:

Arbeiderstatuut
-uurloon
<u>-arbeidsreglement</u>

Je kan in StarUML in het Properties venster een gewoon attribuut wijzigen naar een attribuut met class bereik: je plaatst OwnerScope op CLASSIFIER.

### 3.5 Methods en constructors

Je tikt methods en constructors in het onderste compartiment van een class.

Je kan constructors extra aanduiden met `<<constructor>>` voor de constructor.

Een woord tussen `<<` en `>>`, dat extra uitleg geeft, wordt een stereotype genoemd.

Werknemer
-rijksregisternummer: long -voornamen: String[1..*] -familienaam: String -geboortedatum: Date -datumInDienst: Date
<code>&lt;&lt;constructor&gt;&gt;+Werknemer(rijksregisternummer: long)</code>


Het plusteken voor de constructor duidt aan dat dit een *public* constructor wordt in de class.

Je kan tussen ronde haakjes parameters en hun type vermelden.

Een method kan ook class bereik hebben. Dit betekent dat je de method niet enkel op een object kan uitvoeren, maar ook op de class. Je onderstreept een method met class bereik:

Spaarrekening
-nummer: long -saldo: decimal <u>-intrest: decimal</u>
<u>+verhoogIntrest(waarde: decimal)</u> <u>+verlaagIntrest(waarde: decimal)</u>

### 3.6 StarUML

1. Je dubbelklikt in het bovenste compartiment van de class `Werknemer` en je kiest de knop .
2. Je tikt `<<constructor>> Werknemer(rijksregisternummer: long)` en je drukt Enter.

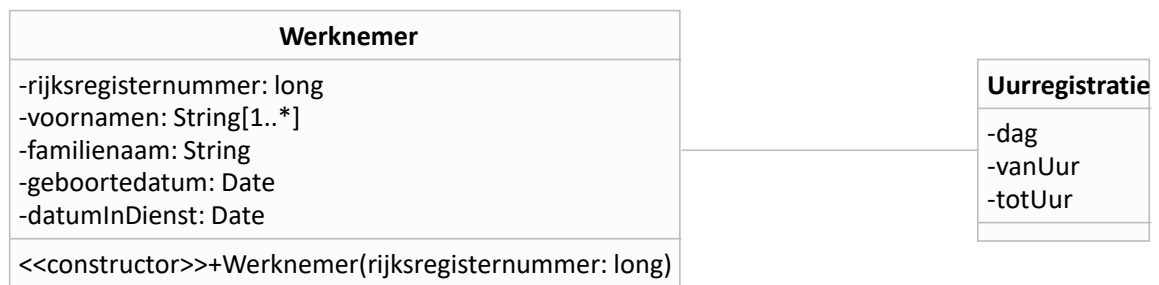
### 3.7 Associaties tussen classes

Een class heeft een associatie met een andere class als een object uit de ene class gedurende het grootste deel van zijn bestaan in het systeem verbonden is met een object uit de andere class.

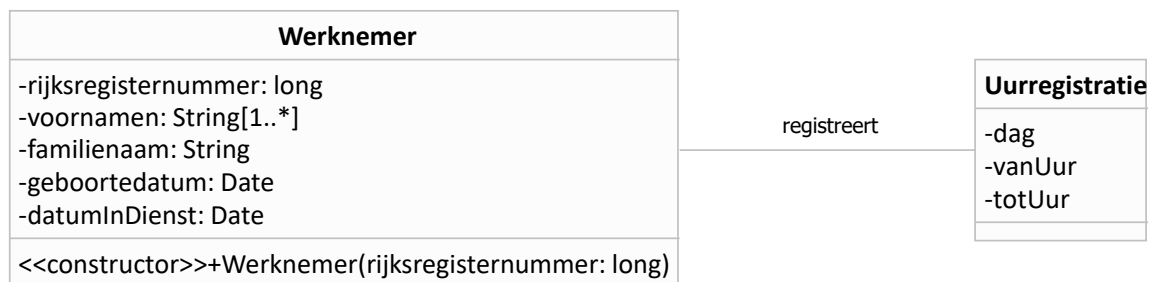
Als twee zelfstandige naamwoorden uit de werkelijkheid één van volgende verbanden hebben, is er een associatie tussen de classes die bij deze zelfstandige naamwoorden horen:

Verband	Voorbeeld
Fysiek	Een vliegtuig heeft twee vleugels
Logisch	Een bedrijf heeft verkoopafdelingen
Is een beschrijving van	Een laadbon is een beschrijving van een lading
Is geregistreerd in	Een wijziging is geregistreerd in een logboek
Is lid van	Een inwoner is lid van een gemeente
Is eigendom van	Een auto is eigendom van een klant

Je drukt een associatie uit met een lijn tussen de twee classes:



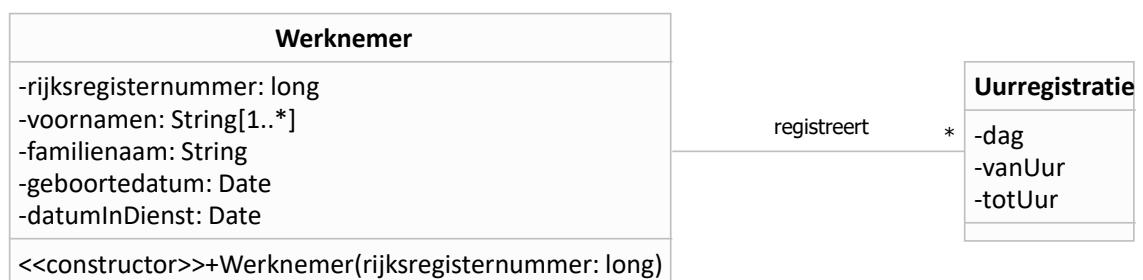
Je kan de associatie een naam geven:



Je leest de associatie van links naar rechts. Je leest dus "Werknemer registreert Uurregistratie", niet "Uurregistratie registreert Werknemer".

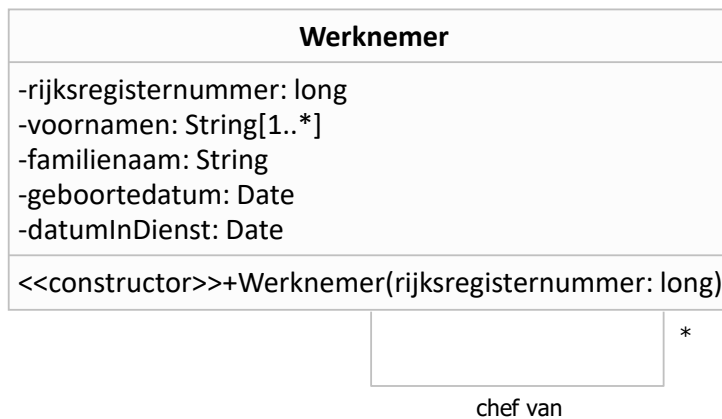
Een associatie heeft twee associatie-einden (één aan de ene class en één aan de andere class). Elk associatie-einde heeft een multiplicititeit. Deze geeft aan hoeveel objecten kunnen deelnemen aan de associatie. Standaard is de multiplicititeit één. Dit zou inhouden dat één werknemer één uurregistratie heeft. Een werknemer heeft echter meerdere uurregistraties.

Je duidt dit aan door de multiplicititeit aan de kant van Uurregistratie op \* te plaatsen:



- 1..\* betekent één of meerdere keren.
- \* betekent nul of meerdere keren.
- 3 betekent 3 keren.

Een class kan ook een associatie met zichzelf hebben (een reflexieve associatie). De chef van een werknemer is bijvoorbeeld terug een werknemer. Een chef kan meerdere ondergeschikten hebben:



Het is in dit voorbeeld onduidelijk welk associatie-einde de chef voorstelt en welk associatie-einde de ondergeschikten voorstelt. Je verduidelijkt door aan elk associatie-einde een woord te tikken:



Associaties zijn standaard bidirectioneel. Je kan de associatie van links naar rechts volgen (een chef kent zijn ondergeschikten) en omgekeerd (een ondergeschikte kent zijn chef).

Associaties kunnen ook gericht zijn. In het volgende voorbeeld duidt de pijl aan dat de chef wel zijn ondergeschikten kent, maar dat een ondergeschikte zijn chef niet kent:



In code houdt dit in dat de class `Werknemer` wel een `Set<Werknemers>` ondergeschikten variabele bevat, maar geen `Werknemer chef` variabele.

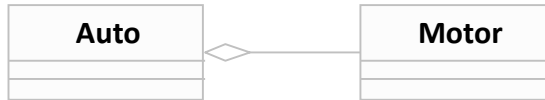
### 3.8 Aggregatie

Als de associatie kan uitgedrukt worden als *bevat een*, is dit een speciale associatie: een aggregatie.

Voorbeeld: een auto bevat een motor. Je drukt een aggregatie uit met een lijn.

Deze lijn heeft een open parallellogram aan de kant van de class die de andere bevat.

Voorbeeld: een auto bevat een motor.



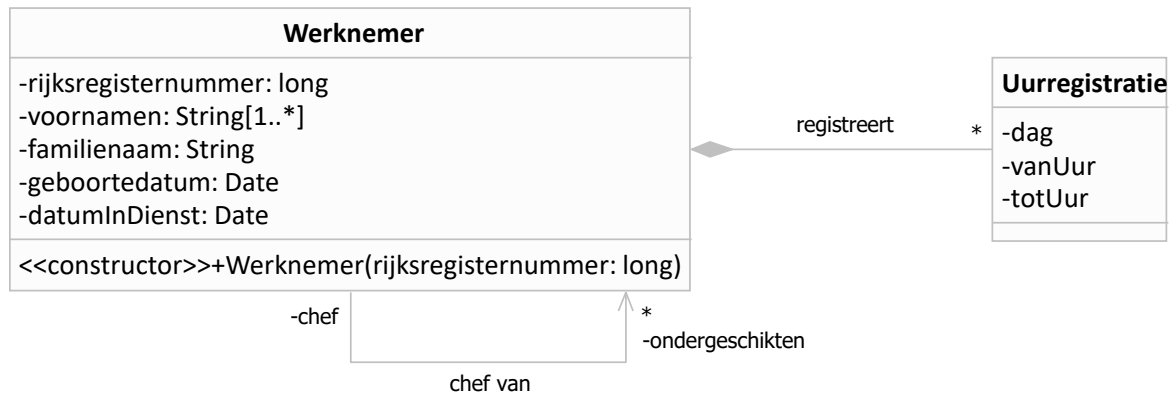
### 3.9 Compositie

Als bij een aggregatie het kleine ding in zijn levensduur maar tot één groot bevattend ding kan behoren is dit een speciale vorm van aggregatie: compositie. Dit geldt niet voor een motor ten opzichte van een auto: je kan de motor uit de auto nemen en in een andere auto plaatsen.

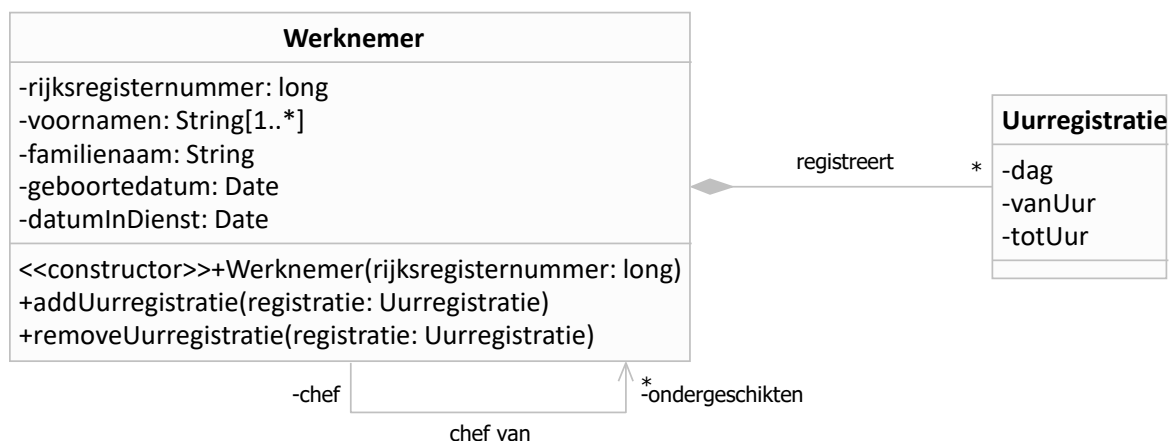
Dit geldt wel voor een uurregistratie ten opzichte van een werknemer: een uurregistratie behoort zijn volledige levensduur tot één werknemer. Ook geldt dat een uurregistratie niet langer kan bestaan dan de werknemer waartoe hij behoort.

Je drukt een compositie uit met een lijn.

Deze lijn heeft een gesloten parallellogram aan de kant van de class die de andere bevat:



Je komt nu ook tot het inzicht dat de class Werknemer twee extra methods heeft:



### 3.10 StarUML

1. Je kiest in de Toolbox een Composition en je sleept van Uurregistratie naar Werknemer.
2. Je dubbelklikt het compositie-einde aan de kant van Uurregistratie en je wijzigt in het uitvalluik rechts de multipliciteit naar \*.
3. Je dubbelklikt de compositie in het midden en je wijzigt de naam naar registreert.
4. Je kiest in de Toolbox een Association en je sleept van Werknemer naar Werknemer.

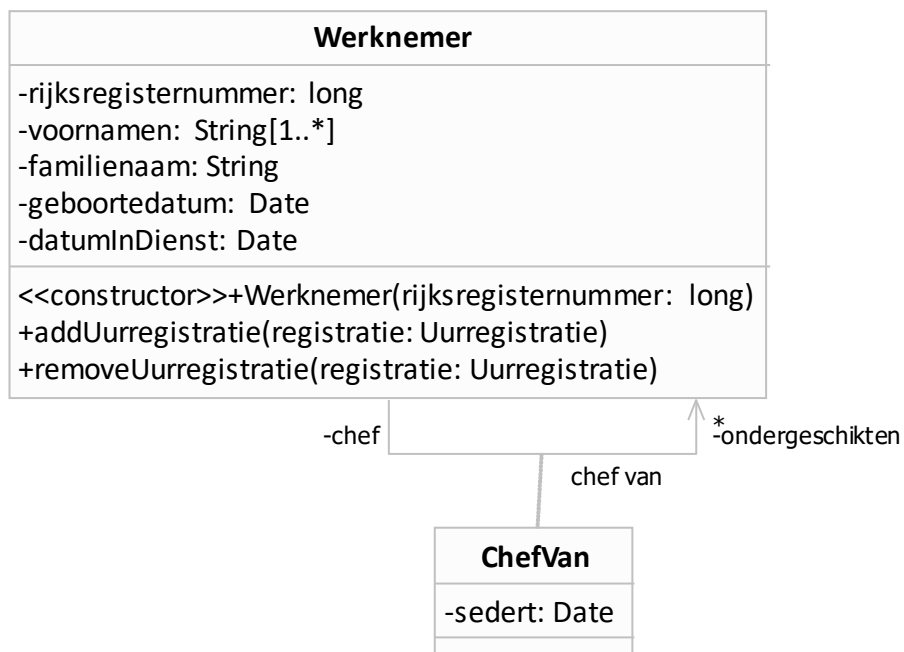


5. Je dubbelklikt één van de associatie einden, je wijzigt in het uitvalluik rechts de multipliciteit naar \* en je wijzigt links de rol naar ondergeschikten.
6. Je dubbelklikt de associatie in het midden en je wijzigt de naam naar chef van.
7. Je dubbelklikt het andere associatie einde en je wijzigt links de rol naar chef.
8. Je voegt de methods addUurregistratie en removeUurregistratie toe.  
Opmerking: als je C# programmeert, noem je de methods AddUurregistratie en RemoveUurregistratie.
9. Je kiest in de Toolbox een Association en je sleept van Adres naar Gemeente.
10. Je wijzigt de multipliciteit aan de kant van Adres naar 1..\*
11. Je kiest in de Toolbox een Association en je sleept van Werknemer naar Adres.
12. Je wijzigt de multipliciteit aan de kant van Werknemer naar 1..\*

### 3.11 Associatieclass

Een associatie class is verbonden met een associatie. Je maakt een associatie class als de associatie zelf attributen of methods heeft. De levensduur van een object van een associatie class is beperkt tot de levensduur van de associatie waar dit object bij hoort.

Voorbeeld: de associatie "chef van " tussen twee werknemers heeft een attribuut sedert (Date) :



Er loopt een onderbroken lijn van de associatie naar de associatieclass.

### 3.12 StarUML

1. Je voegt de class ChefVan toe
2. Je voegt het attribuut sedert toe.
3. Je kiest in de Toolbox een AssociationClass en je sleept van de associatie naar de class ChefVan.



Class diagram 1: zie takenbundel

### 3.13 Inheritance

Inheritance drukt een *is een* relatie uit tussen twee classes.

De derived class *is een* base class.

Voorbeelden:

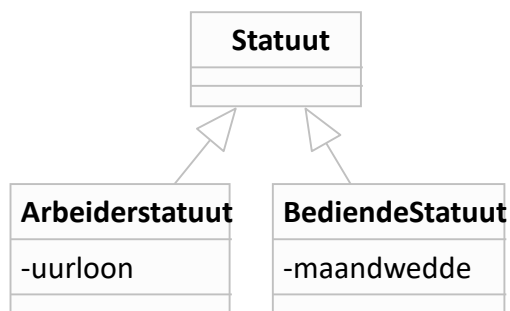
- Een spaarrekening is een rekening
- Een vierkant is een figuur
- Een auto is een voertuig

Je vindt inheritance door de classes die je al hebt per twee te overlopen en in volgende zin het woord op ... te vinden: de eerste derived class *is een* ... en de tweede derived class *is ook een* ...

Als je dit doet in het voorbeeld van de tijdsregistratie kom je tot:

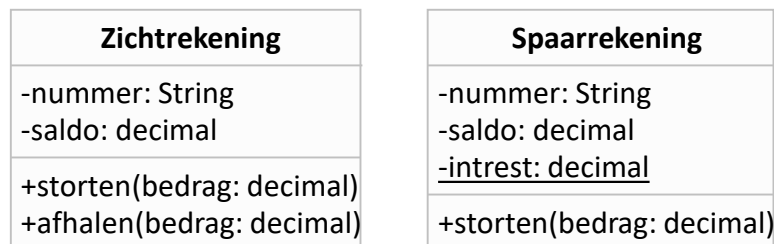
een arbeider statuut *is een* statuut en een bediende statuut *is ook een* statuut.

Je maakt van het gevonden woord een class, waar de andere twee classes van erven:

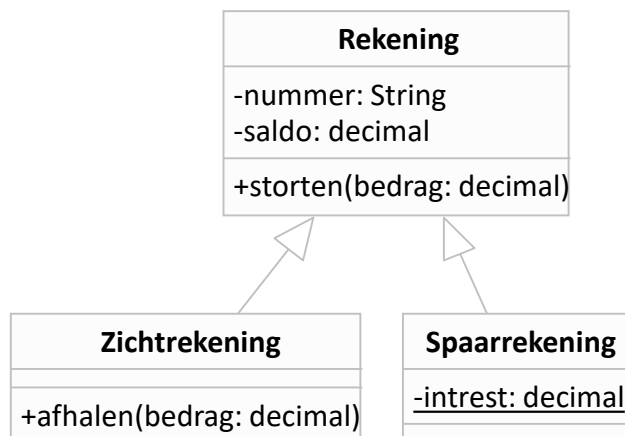


Je duidt inheritance aan met een pijl die wijst van de derived class naar de base class.

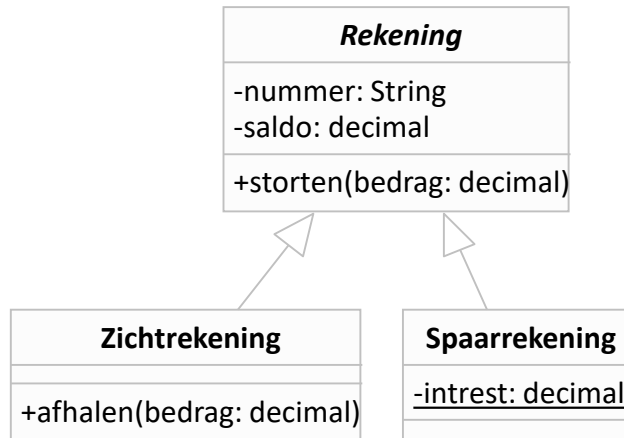
Als de derived classes gemeenschappelijke attributen of methods hebben, verplaats je deze naar de base class. Een voorbeeld in de bank: vóór het toepassen van inheritance:



Na het toepassen van inheritance:



Als de base class enkel dient om er van te erven, maar je van die class geen concrete instantie kan maken in de werkelijkheid, maak je van die class een abstract class. Dit is meestal het geval. Ook hier: je kan een concrete zichtrekening of spaarrekening maken bij een bank, je kan echter geen concrete "rekening" maken bij een bank:



Je plaatst de naam van een abstract class schuin.

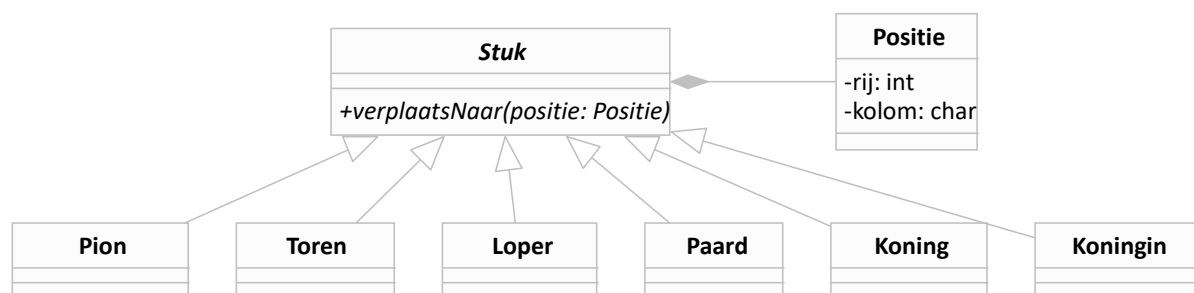
### 3.14 StarUML

1. Je voegt een class Statuut toe.
2. Je maakt er een abstract class van door in het Properties venster `isAbstract` aan te vinken.
3. Je kiest een Generalization in de Toolbox en je sleept van `ArbeiderStatuut` naar `Statuut`.
4. Je kiest een Generalization in de Toolbox en je sleept van `BediendeStatuut` naar `Statuut`.
5. Je komt tot het inzicht dat elke werknemer een statuut heeft. Je kiest een Composition in de Toolbox en je sleept van `Statuut` naar `Werknemer`.

### 3.15 Abstract method

Een abstract class kan ook abstract methods bevatten.

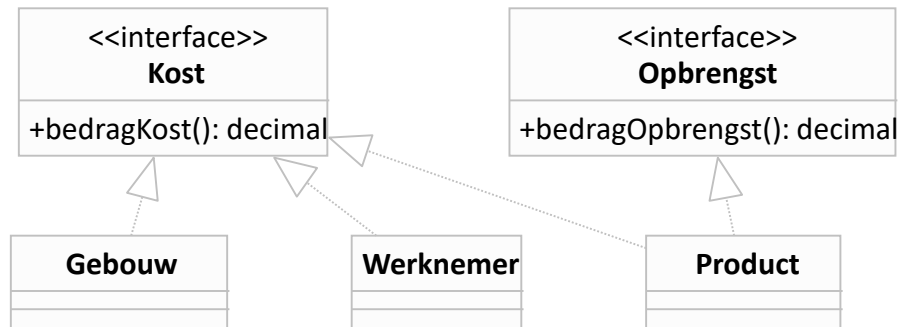
Dit zijn methods die de abstract class *kan* uitvoeren in de werkelijkheid, maar waarvan je niet concreet kan zeggen *hoe* de method uitgevoerd wordt. Dit is het geval bij een schaakspel: ieder stuk *kan* bewegen, maar je kan in de class `Stuk` niet zeggen hoe een stuk concreet beweegt. Je kan dit maar zeggen in afgeleide classes:



Abstract methods krijgen een schuine tekst.

### 3.16 Interface

Een interface lijkt op een class, maar bevat enkel abstract methods, geen concrete methods of attributen. Het voordeel van een interface in Java en C# is dat een class meerdere interfaces kan implementeren, waar een class maar van één class kan erven.



Je tekent een interface als een rechthoek met twee compartimenten

- Het bovenste compartiment bevat het stereotype `<<interface>>` en de naam van de interface.
- Het onderste compartiment bevat de methods

Je trekt een onderbroken pijl van een class (die de interface implementeert) naar de interface.

Je kan dit voorbeeld lezen als

- Een gebouw is een kost
- Een werknemer is een kost
- Een product is een kost én een opbrengst

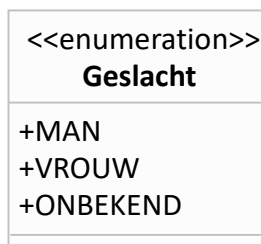
### 3.17 StarUML

Je vervangt de class Statuut door een interface Statuut


1. Je klikt met de rechtermuisknop op de class Statuut en je kiest **Select** in **Model Explorer**.
2. Je klikt met de rechtermuisknop op de class in de **Model Explorer** en je kiest **Delete** from **Model**.
3. Je kiest een **Interface** in de **Toolbox** en je klikt in het class diagram.
4. Je wijzigt de naam van de interface naar Statuut.
5. Je klikt met de rechtermuisknop op de interface en je kiest **Format, Stereotype Display, Textual**.
6. Als je methods zou toevoegen aan de interface, klik je met de rechtermuisknop op de interface en je kiest **Format, Suppress Operations**.
7. Je kiest een **Realization** in de **Toolbox** en je sleept van **Arbeiderstatuut** naar Statuut.
8. Je kiest een **Realization** in de **Toolbox** en je sleept van **Bediendestatuu**t naar Statuut.
9. Je kiest een **Composition** in de **Toolbox** en je sleept van Statuut naar **Werknemer**.

### 3.18 Enum

Je stelt een enum op volgende manier voor:

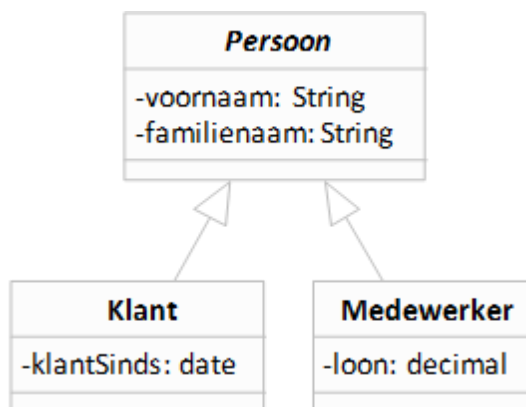


### 3.19 StarUML

1. Je kiest een Enumeration in de Toolbox en je klikt in het class diagram.
2. Je wijzigt de naam naar Geslacht
3. Je dubbelklikt in het bovenste compartiment en je kiest de knop 
4. Je voegt de literal MAN toe.  
Je voegt op dezelfde manier de literals VROUW en ONBEKEND toe.  
Opmerking: Dit voorbeeld volgt de Java naamgeving conventie.  
Als je C# programmeert, voeg je de attributen Man, Vrouw en Onbekend toe.
5. Je kiest een DirectedAssociation in de Toolbox en je sleept van Werknemer naar Geslacht.
6. Je kiest een Enumeration in de Toolbox en je klikt in het class diagram.
7. Je wijzigt de naam naar Status
8. Je voegt de literals INGEBRACHT, GOEDGEKEURD en AFGEKEURD toe.
9. Je kiest een DirectedAssociation in de Toolbox en je sleept van Uurregistratie naar Status.
10. Je voegt aan de class Uurregistratie volgende methods toe:
  - a. heeftOveruren():boolean
  - b. keurGoed()
  - c. keurAf()

### 3.20 Rollen

Je definieert in je class diagram soms rollen die mensen spelen. Voorbeeld: in een garage heb je de rol klant en de rol medewerker. Het is verkeerd dit uit te drukken met inheritance:

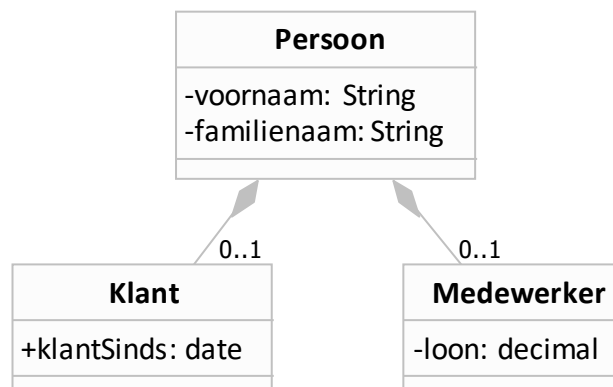


De fout is als volgt: een persoon kan in de werkelijkheid tegelijk medewerker én klant zijn.

In dit verkeerde model moet je voor die persoon een klant object én een medewerker object maken. Beide objecten bevatten de voornaam en de familienaam van de persoon.

Je hebt dus dubbele data.

Het is correct dit uit te drukken met composition:



- Je maakt per persoon een Persoon object.
- Als deze persoon enkel klant is, maak je ook een Klant object.
- Als deze persoon enkel medewerker is, maak je ook een Medewerker object.
- Als deze persoon klant én medewerker is, maak je ook een Klant object en een Medewerker object

Dit voorbeeld uitgewerkt in Java:

```
import java.math.BigDecimal;
import java.time.LocalDate;
class Klant {
    private LocalDate klantSinds;
    Klant(LocalDate klantSinds) {
        this.klantSinds = klantSinds;
    }
    @Override
    public String toString() {
        return "klant sinds:" + klantSinds;
    }
}
class Medewerker {
    private BigDecimal loon;
    Medewerker(BigDecimal loon) {
        this.loon = loon;
    }
    @Override
    public String toString() {
        return "loon:" + loon;
    }
}
class Persoon {
    private String voornaam;
    private String familienaam;
    private Klant klant;
    private Medewerker medewerker;
    Persoon(String voornaam, String familienaam) {
        this.voornaam = voornaam;
        this.familienaam = familienaam;
    }
    void setKlant(Klant klant) {
        this.klant = klant;
    }
    void setMedewerker(Medewerker medewerker) {
        this.medewerker = medewerker;
    }
    @Override
    public String toString() {
        StringBuilder builder = new StringBuilder();
        builder.append(voornaam).append(' ').append(familienaam);
        if (klant != null) {
            builder.append(',').append(klant);
        }
        if (medewerker != null) {
            builder.append(',').append(medewerker);
        }
        return builder.toString();
    }
}
```

```

public class Main {
    public static void main(String[] args) {
        Persoon zonderKlantZonderMedewerker =
            new Persoon("geen klant", "geen medewerker");
        System.out.println(zonderKlantZonderMedewerker);
        // output: geen klant geen medewerker
        Persoon enkelKlant = new Persoon("enkel", "klant");
        enkelKlant.setKlant(new Klant(LocalDate.of(2000, 1, 1)));
        System.out.println(enkelKlant);
        // output: enkel klant,klant sinds:2000-01-01
        Persoon enkelMedewerker = new Persoon("enkel", "medewerker");
        enkelMedewerker.setMedewerker(new Medewerker(BigDecimal.valueOf(3_000)));
        System.out.println(enkelMedewerker);
        // output: enkel medewerker,loon:3000
        Persoon klantEnMedewerker = new Persoon("klant", "en medewerker");
        klantEnMedewerker.setKlant(new Klant(LocalDate.of(1994, 1, 1)));
        klantEnMedewerker.setMedewerker(new Medewerker(BigDecimal.valueOf(4_000)));
        System.out.println(klantEnMedewerker);
        // output: klant en medewerker,klant sinds:1994-01-01,loon:4000
    }
}

```

Dit voorbeeld uitgewerkt in C#:

```

using System;
using System.Text;
class Klant
{
    private DateTime klantSinds;
    public Klant(DateTime klantSinds)
    {
        this.klantSinds = klantSinds;
    }
    override public String ToString()
    {
        return $"klant sinds:{klantSinds}";
    }
}
class Medewerker
{
    private decimal loon;
    public Medewerker(decimal loon)
    {
        this.loon = loon;
    }
    override public String ToString()
    {
        return $"loon:{loon}";
    }
}
class Persoon
{
    private string voornaam;
    private string familienaam;
    private Klant klant;
    private Medewerker medewerker;
    public Persoon(string voornaam, string familienaam)
    {
        this.voornaam = voornaam;
        this.familienaam = familienaam;
    }
}

```

```

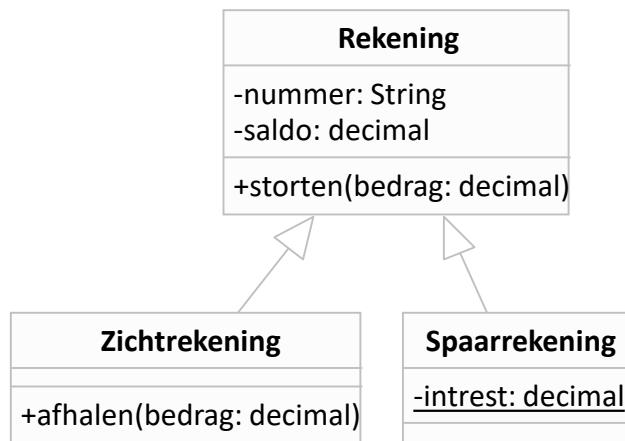
public Klant Klant
{
    set
    {
        this.klant = value;
    }
}
public Medewerker Medewerker
{
    set
    {
        this.medewerker = value;
    }
}
override public string ToString()
{
    var builder = new StringBuilder();
    builder.Append(voornaam).Append(' ').Append(familienaam);
    if (klant != null)
    {
        builder.Append(',').Append(klant);
    }
    if (medewerker != null)
    {
        builder.Append(',').Append(medewerker);
    }
    return builder.ToString();
}
}
class RollenVoorbeeld
{
    static void Main(String[] args)
    {
        var zonderKlantZonderMedewerker = new Persoon("geen klant", "geen medewerker");
        Console.WriteLine(zonderKlantZonderMedewerker);
        // output: geen klant geen medewerker
        var enkelKlant = new Persoon("enkel", "klant");
        enkelKlant.Klant = new Klant(new DateTime(2000, 1, 1));
        Console.WriteLine(enkelKlant);
        // output: enkel klant,klant sinds:01/01/2000 00:00:00
        var enkelMedewerker = new Persoon("enkel", "medewerker");
        enkelMedewerker.Medewerker = new Medewerker(3000M);
        Console.WriteLine(enkelMedewerker);
        // output: enkel medewerker,loon:3000
        var klantEnMedewerker = new Persoon("klant", "en medewerker");
        klantEnMedewerker.Klant = new Klant(new DateTime(1994, 1, 1));
        klantEnMedewerker.Medewerker = new Medewerker(4000M);
        Console.WriteLine(klantEnMedewerker);
        // output: klant en medewerker,01/01/1994 00:00:00,loon:4000
    }
}

```

Je mag de verbanden tussen rekening, zichtrekening en spaarrekening wél met inheritance voorstellen, omdat

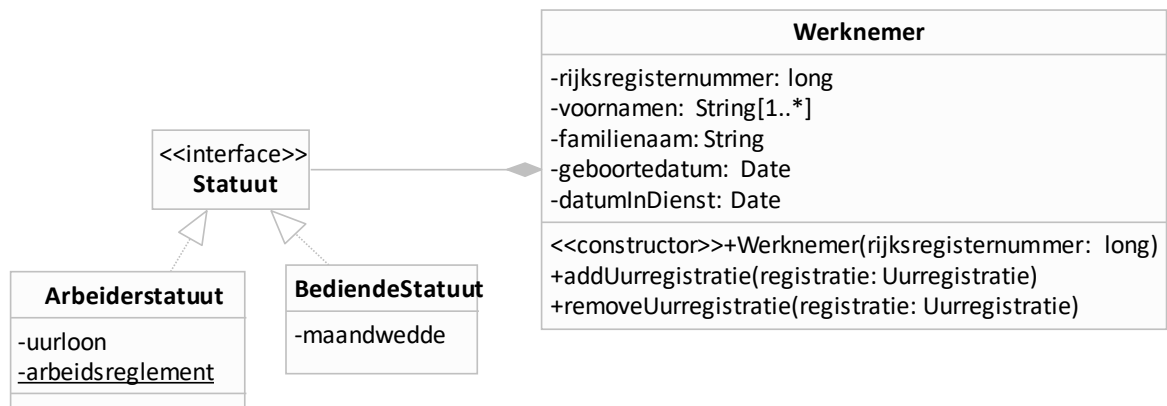
- een rekening niet tegelijk zichtrekening en spaarrekening kan zijn
- je een zichtrekening niet kan wijzigen naar een spaarrekening
- je een spaarrekening niet kan wijzigen naar een zichtrekening





De verbanden tussen Werknemer, Arbeider en Bediende zijn ook niet met inheritance voorgesteld (Arbeider en Bediende zouden erven van Werknemer), omdat een werknemer in zijn levensduur kan veranderen van arbeider naar bediende. Het bijbehorende object daarentegen kan in zijn levensduur niet veranderen van Arbeider class naar Bediende class.

Je ziet de correcte voorstelling hier onder. Een werknemer verandert niet van arbeider naar bediende, maar wijzigt enkel zijn statuut eigenschap van ArbeiderStatuut naar BediendeStatuut.



Class diagram 2: zie takenbundel



Class diagram 3: zie takenbundel

## 4 SINGLE RESPONSIBILITY

Het single responsibility principe zegt dat een class slechts één soort functionaliteit bevat. Een class die de werkelijkheid voorstelt (bvb. de class Werknemer) mag geen andere code bevatten dan de werkelijkheid voor te stellen. Dit houdt in:

- dat je een aparte class maakt die de GUI voorstelt om een werknemer bij te werken
- dat je een aparte class maakt om werknemer van én naar de database te brengen
- dat je een aparte class maakt om werknemers in XML formaat over het internet te sturen

Op deze manier blijft ieder van deze classes klein en overzichtelijk.

Als je al deze classes in één diagram plaatst, is dit diagram ook niet meer overzichtelijk.

Het is beter een apart diagram te maken voor classes die de werkelijkheid voorstellen, een ander diagram voor classes die met de database samenwerken, ...

Het is daarbij handig als elk diagram ook een duidelijke naam heeft.

### 4.1 StarUML

Je hernoemt het class diagram dat je al hebt naar Model

(model is een veel gebruikte verzamelnaam voor classes die de werkelijkheid voorstellen)

1. Je kiest de Diagram Explorer (rechts)
2. Je kiest het tweede diagram in de categorie Class Diagrams
3. Je wijzigt in het Properties venster de Name property naar Model

Je voegt een class diagram toe voor repository classes

(repository is een veel gebruikte naam voor een class die met de database samenwerkt).

1. Je kiest de Model Explorer (rechts)
2. Je klikt met de rechtermuisknop op <<designModel>> Design Model
3. Je kiest Add Diagram, Class Diagram
4. Je wijzigt de naam van het diagram naar Repositories

Je voegt volgende class toe:

WerknemerRepository
<pre>+findByRijksregisternummer(nummer: long): Werknemer +findAll(): Werknemer[*] +create(werknemer: Werknemer) +update(werknemer: Werknemer) +delete(werknemer: Werknemer)</pre>

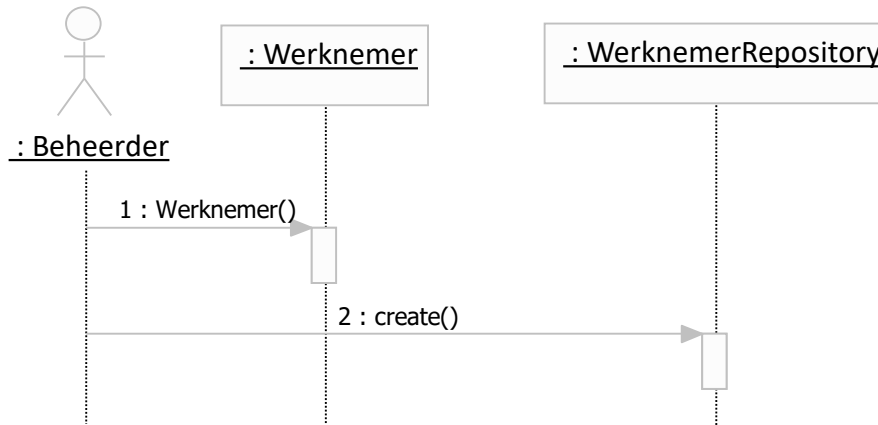
- De method `findByRijksregisternummer` zoekt een werknemer in de database aan de hand van zijn rijksregisternummer.
- De method `findAll` zoekt alle werknemers in de database.
- De method `create` voegt een werknemer toe aan de database.
- De method `update` wijzigt een werknemer in de database.
- De method `delete` verwijdert een werknemer uit de database.



Repository : zie takenbundel

## 5 SEQUENCE DIAGRAM

Je beschrijft in een sequence diagram welke classes samenwerken om één use-case uit te werken. Je beschrijft ook de method oproepen die daarbij gebeuren.



Je ziet boven in het diagram welke actor de use-case start en welke classes betrokken zijn bij het uitvoeren van de use-case. Je leest de rest van het diagram van boven naar onder:

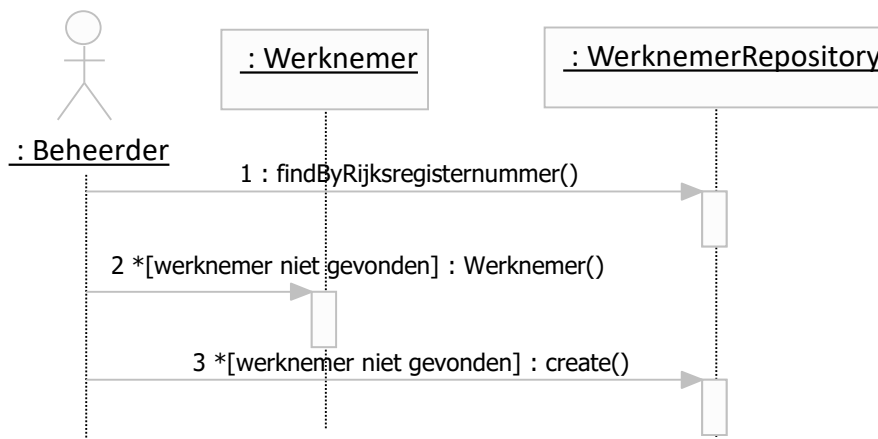
1. Je maakt een nieuw werknemer object door de constructor van de class Werknemer op te roepen.
2. Je slaat dit werknemer object op in de database door de create method van de class WerknemerRepository op te roepen.

### 5.1 Conditionele message

Een message is een method oproep. Een conditionele message is een method oproep die je enkel doet als aan een voorwaarde voldaan is.

Voorbeeld: je zoekt eerst of de werknemer al voorkomt in de database.




Pas als dit niet het geval is, maak je een werknemer object en sla je dit object op in de database:





Je schrijft de voorwaarde tussen vierkante haakjes voor de method oproep.

### 5.2 StarUML

1. Je kiest de Model Explorer (rechts)
2. Je klikt met de rechtermuisknop op <<designModel>> Design Model en je kiest Add Diagram, Sequence Diagram
3. Je wijzigt de naam naar Nieuwe werknemer registreren
4. Je sleept de Actor Beheerder van de Model Explorer naar het nieuwe diagram.
5. Je sleept de class Werknemer van de Model Explorer naar het diagram.
6. Je sleept de class WerknemerRepository van de Model Explorer naar het diagram.
7. Je kiest een Stimulus in de Toolbox en je sleept van de lifeline (verticale streep) onder Beheerder naar de lifeline onder WerknemerRepository.

8. Je kiest de knop  in het mini venster. Je ziet een lijst met de constructor(s) en method(s) in de class WerknemerRepository. Je kiest de method findByRijksregisternummer, OK.
9. Je kiest een Stimulus in de Toolbox en je sleept van de lifeline onder Beheerder naar de lifeline onder Werknemer.
10. Je kiest de knop  in het mini venster. Je ziet een lijst met de constructor(s) en method(s) in de class Werknemer. Je kiest de constructor en je kiest OK.
11. Je tikt in het Properties venster Werknemer niet gevonden in de property Branch.
12. Je kiest een Stimulus in de Toolbox en je sleept van de lifeline onder Beheerder naar de lifeline onder WerknemerRepository.
13. Je kiest de knop  in het mini venster. Je ziet een lijst met de constructor(s) en method(s) in de class WerknemerRepository. Je kiest de method create en je kiest OK.
14. Je tikt in het Properties venster Werknemer niet gevonden in de property Branch.

Je koppelt dit diagram aan de bijbehorende use-case:

1. Je selecteert de use-case Nieuwe werknemer registreren in het use-case diagram.
2. Je kiest het tabblad Attachments (rechts onder).
3. Je kiest de knop .
4. Je kiest de knop .
5. Je opent het + teken voor Design Model
6. Je opent het + teken voor CollaborationInstanceSet1
7. Je opent het + teken voor InteractionInstanceSet1
8. Je kiest Nieuwe werknemer registreren
9. Je kiest OK, OK

Telkens je Nieuwe werknemer registreren dubbelklikt in het venster Attachments, wordt dit diagram geopend in StarUML.



Overschrijven : zie takenbundel

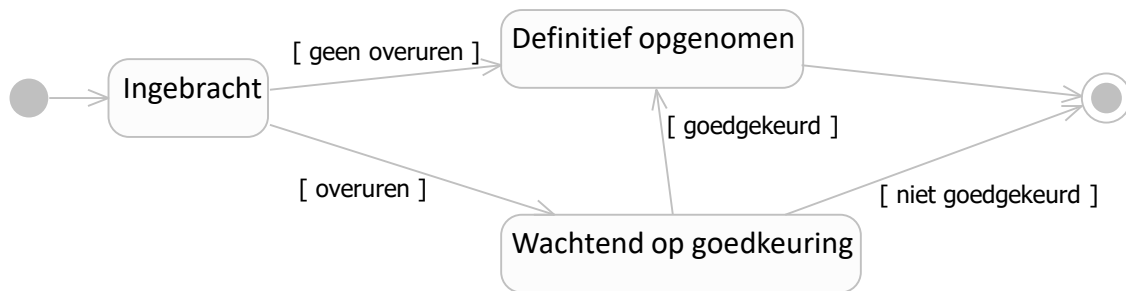
## 6 STATE CHART DIAGRAM

Als een object in zijn levensduur meerdere toestanden heeft, kan je dit uitdrukken in een state chart diagram.

Een uurregistratie in het tijdregistratiesysteem heeft in zijn levensduur twee toestanden

- Ingebracht
- Wachtend op goedkeuring (indien overuren gepresteerd).
- Definitief opgenomen

Het bijbehorende state chart diagram ziet er als volgt uit:



Je leest een statechart diagram van de bol (●) naar de bol in de bol (⦿)

De bol betekent het begin van de toestanden, de bol in de bol betekent het einde.

Iedere toestand wordt vermeld in een rechthoek met afgeronde hoeken.

De overgang van een toestand naar een andere wordt getekend met een pijl.

Als deze toestandsovergang enkel onder een bepaalde voorwaarde gebeurt, wordt deze voorwaarde tussen vierkante haakjes bij de pijl vermeld.

### 6.1 StarUML

1. Je kiest de Model Explorer.
2. Je klikt met de rechtermuisknop op <<design Model>> Design Model en je kiest Add Diagram, Statechart Diagram.
3. Je wijzigt de naam van het diagram naar States Uurregistratie.
4. Je kiest in de Toolbox een InitialState en je klikt in het diagram.
5. Je kiest in de Toolbox een State en je klikt in het diagram.  
Je wijzigt de naam naar Ingebracht.
6. Je kiest in de Toolbox een Transition en je sleept van de bol naar de state.
7. Je kiest in de Toolbox een State en je klikt in het diagram.  
Je wijzigt de naam naar Definitief opgenomen.
8. Je kiest in de Toolbox een FinalState en je klikt helemaal rechts in het diagram.
9. Je kiest in de Toolbox een Transition en je sleept van Ingebracht naar Definitief opgenomen.
10. Je klikt op de Transition en je tikt in het Properties venster bij de GuardCondition property geen overuren.
11. Je kiest in de Toolbox een State en je klikt in het diagram.  
Je wijzigt de naam naar Wachtend op goedkeuring.
12. Je kiest in de Toolbox een Transition en je sleept van Ingebracht naar Wachtend op goedkeuring.
13. Je klikt op de Transition en je tikt in het Properties venster bij de GuardCondition property overuren.
14. Je kiest in de Toolbox een Transition en je sleept van Wachtend op goedkeuring naar Definitief opgenomen.
15. Je klikt op de Transition en je tikt in het Properties venster bij de GuardCondition property goedgekeurd.



16. Je kiest in de Toolbox een Transition  
en je sleept van Wachtend op goedkeuring naar de eindbollen.
17. Je klikt op de Transition  
en je tikt in het Properties venster bij de GuardCondition property niet goedgekeurd.
18. Je kiest in de Toolbox een Transition  
en je sleept van Definitief opgenomen naar de eindbollen.

De overgangen kunnen zelf ook een naam (Name property) hebben.

Je ziet dit in volgend voorbeeld, waarbij de naam van iedere overgang bij de pijl staat.



Je koppelt dit diagram aan de bijbehorende class:

1. Je selecteert de class Uurregistratie in het class diagram.
2. Je kiest het tabblad Attachments (rechts onder).
3. Je kiest de knop .
4. Je kiest de knop .
5. Je opent het + teken voor Design Model
6. Je opent het + teken voor StateMachine1
7. Je kiest States Uurregistratie
8. Je kiest OK, OK



Schaakspel: zie takenbundel

## 7 STAPPENPLAN

1. Maak het use-case diagram voor het huidige deelproject (of breid het use-case diagram van een vorig deelproject uit).
2. Maak de use-cases in tekstvorm voor het huidige deelproject.
3. Maak het class diagram van de classes uit de werkelijkheid voor het huidige deelproject (of breidt het class diagram van een vorig deelproject uit).
  - a. Maak aan de hand van zelfstandige naamwoorden uit de werkelijkheid classes en attributen.
  - b. Maak aan de hand van werkwoorden uit de werkelijkheid methods.
  - c. Maak de associaties tussen de classes.
  - d. Verfijn (waar toepasselijk) associaties naar aggregaties.
  - e. Verfijn (waar toepasselijk) aggregaties naar composities.
  - f. Maak associatieclasses voor associaties die zélf attributen of methods hebben.
  - g. Maak base classes (gebaseerd op "is een") en maak inheritance van derived classes naar base classes.
  - h. Maak (waar toepasselijk) base classes abstract.
  - i. Maak (waar toepasselijk) methods abstract
  - j. Wijzig base classes die enkel abstract methods bevatten tot interfaces.
4. Maak sequence diagrammen voor de use-cases die veel stappen bevatten.
5. Maak state chart diagrams voor de classes die toestanden hebben in hun levensduur.



Herhalingstaak: zie takenbundel

## 8 COLOFON

<b>Domeinexpertisemanager:</b>	Jean Smits
<b>Moduleverantwoordelijke:</b>	Hans Desmet
<b>Medewerkers:</b>	Hans Desmet
<b>Versie:</b>	28/8/2018
<b>Nummer dotatielijst:</b>	