Inhoudstafel

1 Inleiding

1.1 Waarvoor dient UML

UML is een modelleertaal die we kunnen gaan gebruiken om situaties uit te schrijven of te tekenen alvorens we beginnen te programmeren. UML zorgt er voor dat er eerst over het probleem moet worden nagedacht voordat we beginnen met het schrijven van de code. UML zorgt er ook voor dat anderen een bepaald proces kunnen verstaan zonder dat kennis van enige programmeertaal nodig is.

UML dient ook voor het documenteren van het programma. Nieuwe programmeurs moeten zich op die manier niet eerst door alle code worstelen om het hele programma te begrijpen. Ze kunnen de UML documenten doornemen om zo te begrijpen hoe de applicatie is opgebouwd.

1.2 Wat is UML

Het zal u waarschijnlijk verbazen maar UML is niet enkel een tekening maken van een bepaald scenario binnen een programma. Onder UML wordt ook het maken van **Use-Cases** verstaan. Daarnaast hebben we ook nog **Klassendiagrammen**, **Interactiediagrammen** (zoals Sequentiediagrammen en Samenwerkingsdiagrammen), **Toestandsdiagrammen**, **Activiteitendiagrammen** en **Fysieke diagrammen**.

De belangrijkste van verschillende diagrammen en de use-cases zullen in deze tutorial aan bod komen.

2 Use-cases

2.1 Inleiding

Een use-case is een verhaal dat beschrijft hoe het systeem (of een deel ervan) gebruikt kan worden om aan de vereisten te voldoen. Een niets zeggen de uitleg over wat een use-case nu eigenlijk is. Kort samengevat is het een stappenplan voor een bepaalde functie van het systeem.

Een use-case geeft een overzicht van:

- taken die uitgevoerd moeten zijn voor dat men aan de huidige use-case begint (precondities)
- een omschrijving van een eenvoudige situatie waarbij het doel bereikt wordt (main-scenario)
- uitbreidingen op het main-scenario
- taken die op het einde van de use-case moeten volbracht zijn (postcondities)
- uitzonderlijke situaties kunnen worden opgenomen in een use-case (exceptions)

2.2 Voorbeeld[1]

Doorheen heel deze tutorial zullen we het eenvoudige voorbeeld gebruiken van een registratie van een nieuwe klant in een webapplicatie.

Titel

Registratie nieuwe klant

Primaire actor

Klant

Precondities

De klant moet zich op de juiste pagina bevinden om zich te kunnen registreren.

Main-succes-scenario:

1. De klant vult persoonlijke gegevens (naam, adres, telefoonnummer, e-mail) in.
2. Systeem valideert de gegevens en vraagt confirmatie aan de gebruiker.
3. Klant confirmeert de registratie
4. Systeem registreert de nieuwe klant en verstuurt e-mail naar de klant met de nodige gegevens.

Alternatieve wegen

- Stroompanne/systeemfout.
    - Einde use-case.

2a Klantgegevens komen al voor in het systeem.

- Systeem waarschuwt de klant en keert terug naar stap 1.

2b Klantgegevens zijn niet volledig of correct ingevuld.

- Systeem waarschuwt de klant en keert terug naar stap 1.

3a Klant wil gegevens niet confirmeren wegens fout.

- Terug naar stap 1.

4a Database om nieuwe klant in te bewaren is niet bereikbaar.

- Systeem waarschuwt de klant.
- Systeem waarschuwt de administrator van de website.

Postcondities

De nieuwe klant is toegevoegd aan het systeem.

Opmerking

Bij de alternatieve wegen verwijst 2a naar een mogelijke alternatieve weg die kan voorkomen tijdens stap 2 van het main-succes-scenario.

2.3 Eigenschappen van een goede use-case

Een goede use-case beschrijft een bepaald doel dat moet bereikt worden. Het is een proces dat slechts door één persoon (actor) wordt uitgevoerd. Een goede use-case omschrijft een proces dat tussen de 2 en de 20 minuten duurt. Een use-case bestaat het best uit 3 tot 10 stappen indien men meer dan 10 stappen binnen één use-case heeft loopt men het risico dat ofwel de use-case te gedetailleerd is, of dat de use-case beter opgesplitst kan worden in meerdere aparte use-cases. Als je let op het voorbeeld beschrijft de use case ook een stuk interactie. Het reageert op de input van de actor en vice versa.

1 Het opstellen van een use-case kan op verschillende manieren gebeuren. Sommigen verkiezen de RUP documenten die je van een bleuprint voorzien, anderen verkiezen hun eigen stijl. De basis van al de use-cases is echter wel dezelfde.

2.4 Wat is een actor

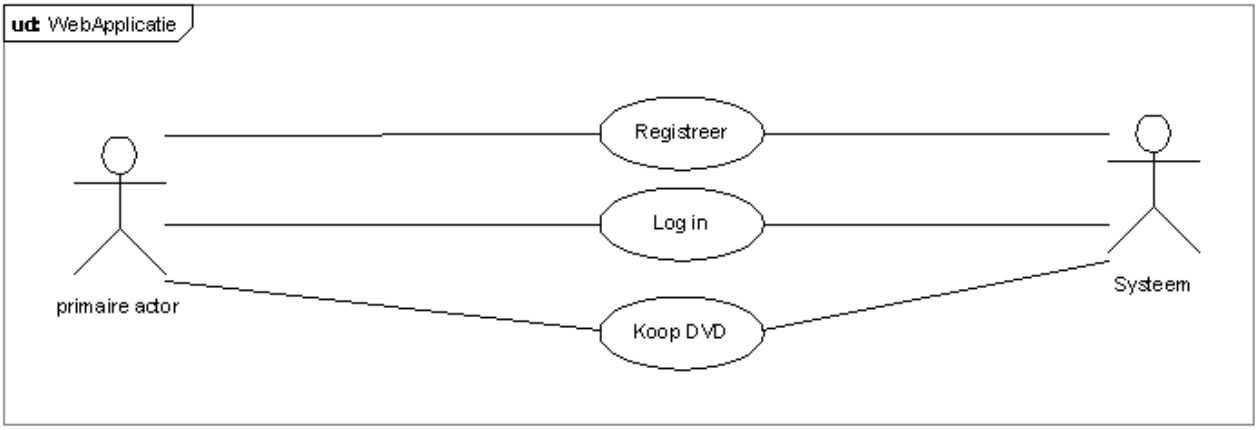
Een actor is een rol die een gebruiker in de context van het systeem speelt. Een actor is diegene die een bepaalde use-case uitvoert. Het is mogelijk dat één actor verschillende use-cases uitvoert, maar ook dat één use-case door verschillende actors kan worden uitgevoerd.

Een actor moet niet altijd een persoon zijn, het kan ook een extern syteem zijn dat wordt voorgesteld als een actor.

3 Use-case diagram

Naast het gebruik van use-case kan het ook handig zijn om een globaal overzicht te krijgen van hoe verschillende use-case tegenover elkaar staan en hoe ze tegenover het systeem enerzijds en de primaire actor anderzijds staan.

Hier ziet men bijvoorbeeld drie use-cases (Registreer, Log in en Koop DVD) waarbij er steeds een zowel een verband is tussen de Primaire actor (de klant) en het Systeem (diegene die de verwerkingen doet)

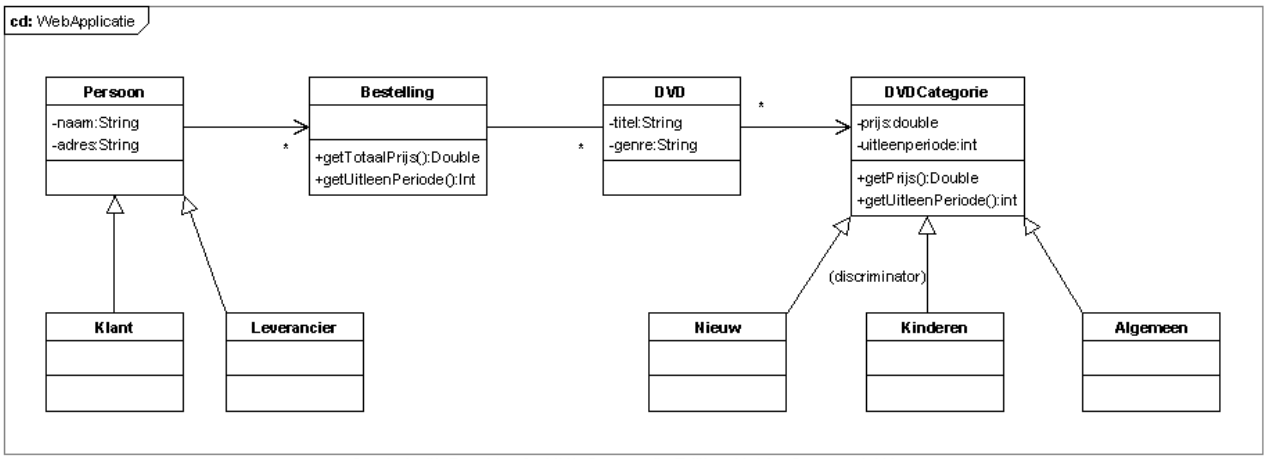


4 Klassendiagrammen

Een klassendiagram beschrijft de typen objecten die een systeem kent en laat zien welke statische relaties, waarvan er verschillende kunnen zijn, tussen deze objecten bestaan. Er zijn twee categorieën statische relaties:

- associaties (een klant huurt een DVD)
- subtypen (een klant is een type persoon)

Om nu goed uit te leggen hoe en wat een klassendiagram is beginnen we met een voorbeeld. Om verder te gaan met onze WebApplicatie waar ook de use-case Registratie klant bijhoort, gaan we nu een klassendiagram maken. Onze WebWinkel zal DVD's verhuren aan klanten. In ons voorbeeld is ook een Leverancier aanwezig, deze is enkel aanwezig om iets meer voorbeelden te creëren ivm associaties. Om het voorbeeld correct te maken zou er naast de klasse Bestelling ook een klasse Levering aanwezig moeten zijn zodat de Leverancier enig nut heeft.



Opmerking

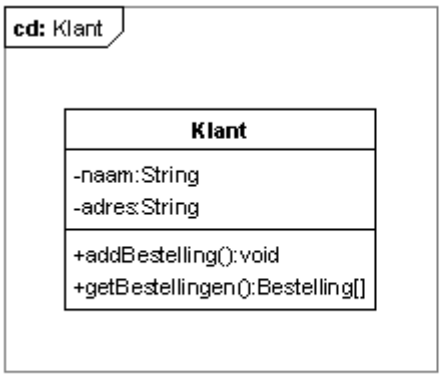
Niet alle methodes en attributen zijn voorzien in het voorbeeld, enkel diegene die nodig zijn om alles duidelijk te maken. Bepaalde linken zijn ook voor discussie vatbaar, maar het doel van deze tutorial is uitleggen waarvoor alles staat en niet om een correct beeld weer te geven van hoe een WebApplicatie er uit zou zien.

4.1 Weergave

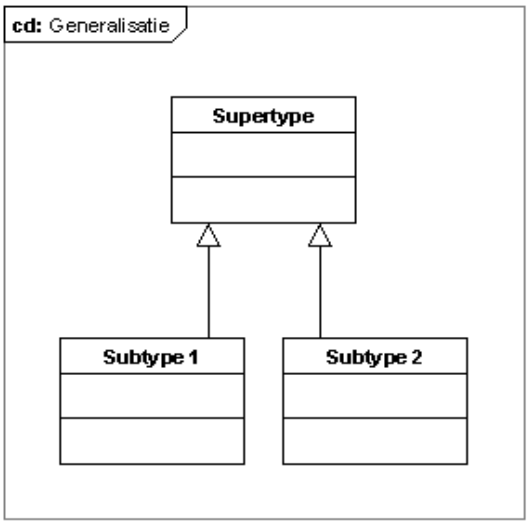
Aangezien UML vooral draait rond weergave ga ik hier een aantal basis structuren weergeven en waar nodig een woordje uitleg bij voegen. Het is namelijk niet nodig om bijvoorbeeld bij generalistatie een volledige uitleg te schrijven over wat generalisatie is aangezien dat niet behoort tot deze tutorial.

4.1.1 Klasse

Elke object georiënteerde programmeur weet wat een klasse is. Een klasse bestaat uit attributen en methodes. Onderstaande figuur is een voorbeeld van hoe een klasse wordt weergegeven binnen een klassendiagram. Deze klasse (Klant) bevat twee attributen (naam en adres) en twee methodes (addBestelling en getBestellingen). Naargelang de details die men wil meegeven gaat men bij de attributen ook het type (hier String) plaatsen en zal men bij de methodes ook de argumenten meegeven en het type dat er teruggegeven wordt (void indien niets wordt teruggegeven vanuit de klasse).



4.1.2 Generalisatie



Created with Poseidon for UML Community Edition. Not for Commercial Use.

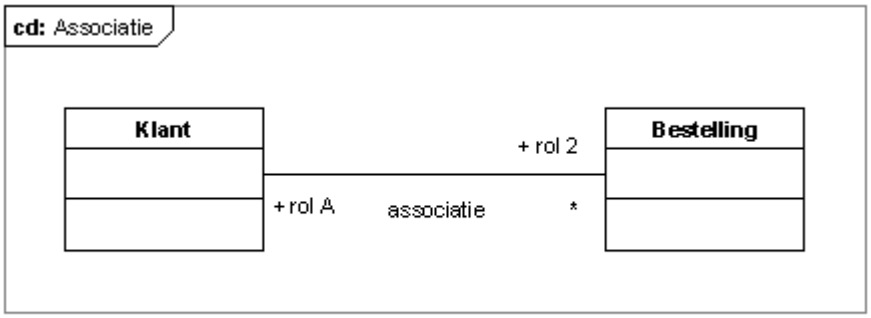
4.1.3 Associatie

Created with Poseidon for UML Community Edition. Not for Commercial Use.

Bij associaties is er ook sprake van veelsoortigheid (Multiplicity). Hierbij zijn er 4 mogelijke gevallen:

1 : precies één
* : veel (nul of meer)
0..1 : optioneel (nul of één)
m..n : waarden voor m..n opgegeven (bv: 2..5 : 2 tot 5)

Indien er geen multiplicity wordt opgegeven gaat men er dus van uit dat deze 0 of 1 is.
In het volgende voorbeeld (uit de WebApplicatie) heeft men een één op veel-relatie (*) wat dus wilt zeggen dat een Klant meerdere bestellingen kan hebben, maar een bestelling maar toebehoort aan één klant.



Created with Poseidon for UML Community Edition. Not for Commercial Use.

4.1.4 Aggregatie en compositie

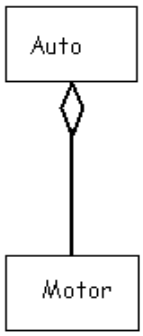
De aggregatie is de relatie 'onderdeel-van'. Een aggregatie beschrijft bijvoorbeeld dat de wielen en de motor onderdeel zijn van een auto. Het enige probleem met een aggregatie is dat het moeilijk uit te leggen is wat nu net het verschil is tussen een aggregatie en een associatie. Al sinds het ontstaan van UML is het altijd vaag geweest over wat nu precies het verschil is. Met als gevolg dat iedereen deze concepten dus op een andere manier gebruikt.

Een compositie is een sterkere vorm van aggregatie. Een compositie is nog steeds de relatie 'onderdeel-van', maar bij een compositie is het zo dat de onderdelen van een compositie slechts mogen toebehoren aan één totaal.

Het verschil tussen aggregatie en compositie kan het best uitgelegd worden aan de hand van een voorbeeld. Het voorbeeld van aggregatie met de wielen en de motor blijft een goed voorbeeld. Maar bij een aggregatie is het zo dat de motor en de wielen ook iets voorstellen zonder dat men er een auto van maakt.

Een goed voorbeeld van een compositie is een hotel met verschillende kamers. Hier is er ook een relatie 'onderdeel-van', want de kamers zijn een onderdeel van het hotel. Bij een compositie is het echter wel zo dat de onderdelen niets voorstellen zonder het geheel. Een hotelkamer bestaat niet zonder dat men een hotel heeft. Breekt men het hotel af zullen ook alle kamers verdwijnen.

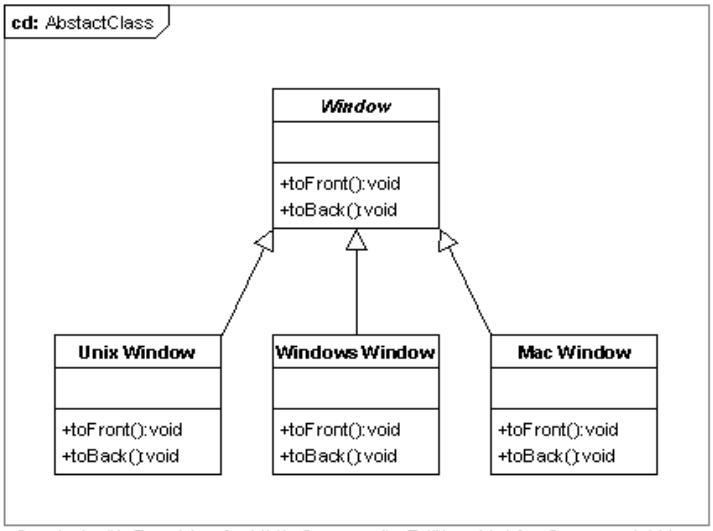
Aggregatie:



Compositie:

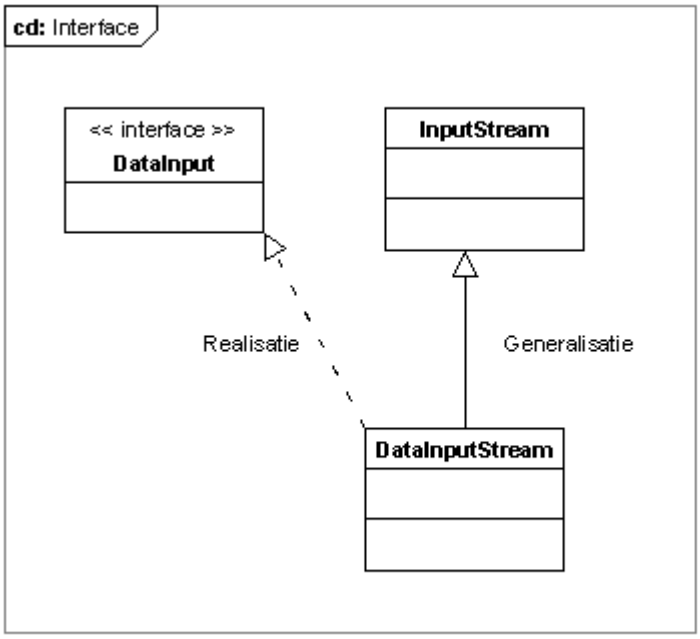
4.1.5 Abstracte klasse

Window is de abstract klasse, om dit duidelijk te maken kan er {abstract} bij de naam van de klasse worden bijgezet of zoals in dit voorbeeld kan de naam cursief worden gezet.



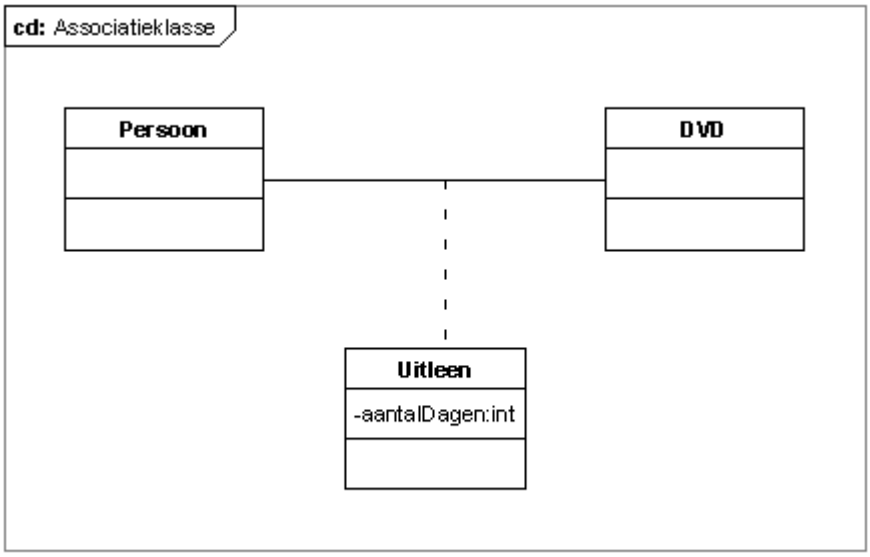
4.1.6 Interface

Het voorbeeld hiervan is ontleend aan de Java architectuur in verband met DataInput.



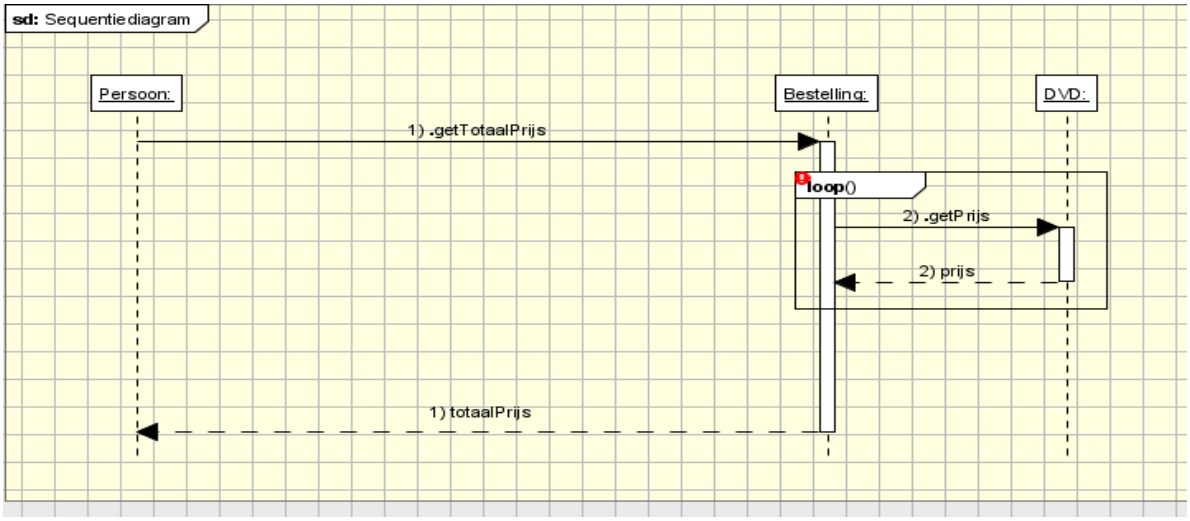
4.1.7 Associatie-klasse

In ons voorbeeld van de WebApplicatie kan een film slechts aan één klant tegelijkertijd worden uitgeleend. Daarom is het interessant om bij te houden van wanneer tot wanneer de film is uitgeleend aan een klant. Dit kan enerzijds bijgehouden worden door de bestelling, maar men kan ook (indien bestelling niet zou bestaan) opteren voor een associatie-klasse. Deze houdt dan de informatie bij over de uitleenperiode.



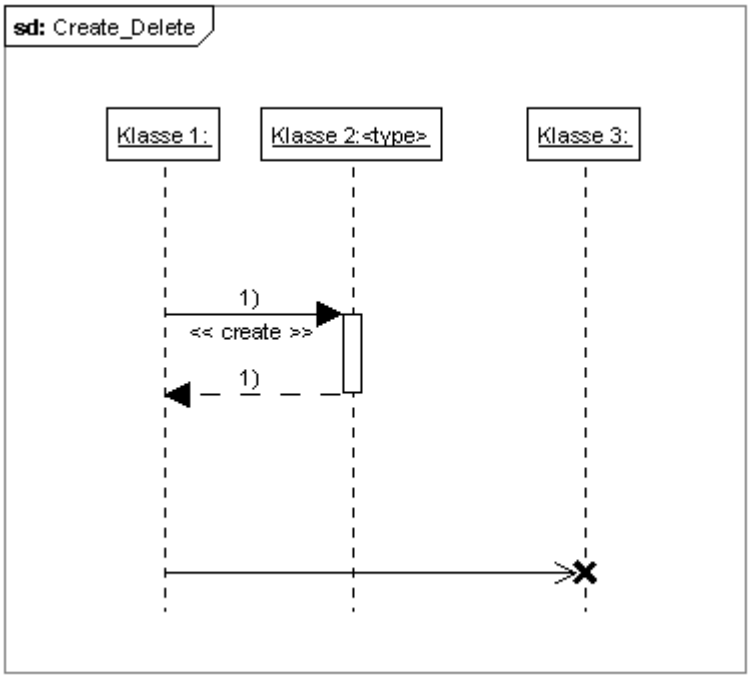
5 Sequentiediagrammen

Sequentiediagrammen geven een overzicht van welke klassen er samen werken om een bepaald proces tot een goed einde te brengen.



Een vrij simpel sequentiediagram enkel en alleen om de basis een beetje te verduidelijken. Dit sequentiediagram gaat het bedrag berekenen die de klant moet betalen. Het sequentiediagram toont hier dat een persoon de totale prijs zal vragen aan de bestelling via de functie getTotaalPrijs() dewelke dan op zijn beurt alle bestelde DVD's zal overlopen en de prijs aan de DVD zal vragen via de getPrijs() methode. Het feit dat de bestelling alle bestelde DVD's overloopt wordt in een sequentiediagram weergegeven met behulp van een loop.

Men kan in een sequentiediagram ook weergeven indien een bepaalde klasse een nieuwe instatie van een andere klasse aanmaakt (create) of de instatie van een bepaalde klasse verwijderd (delete).



6 Activiteitendiagrammen

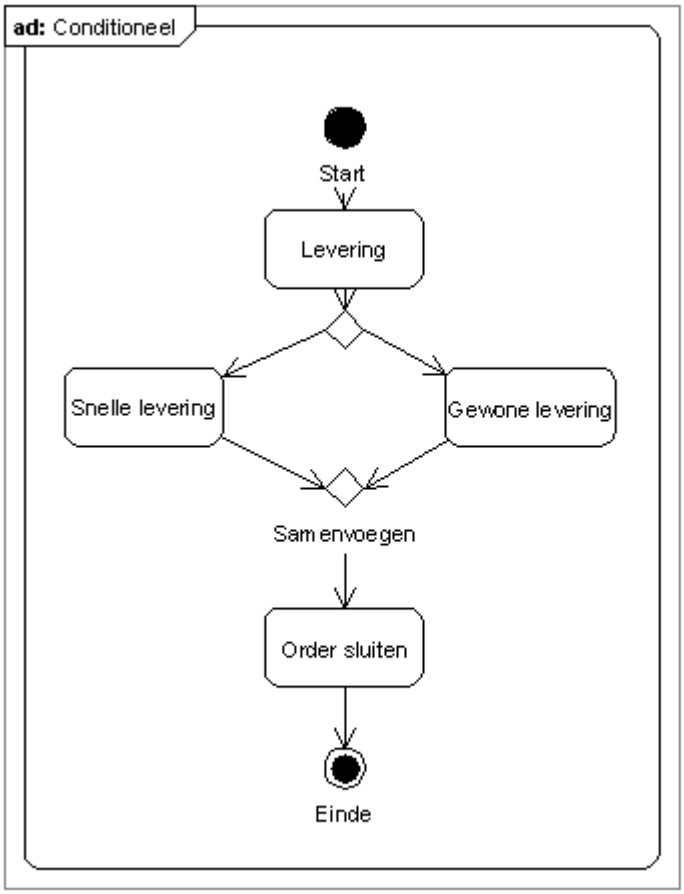
Een activiteitendiagram beschrijft de volgorde van activiteiten en biedt ondersteuning voor zowel conditioneel als parallel gedrag.

6.1 Conditioneel gedrag

Conditioneel gedrag wordt beschreven met vertakkingen en samenvoegingen.

Een vertaking heeft één binnenkomende overgang en meerdere uitgaande overgangen. De uitgaande overgangen worden "bewaakt" zodat er slechts één van de mogelijke uitgangen kan worden genomen. In dit voorbeeld kan men ofwel een snelle levering hebben ofwel een gewone levering maar geen snelle-gewone levering.

Een samenvoeging heeft verschillende binnenkomende overgangen en slechts één uitgang. Een samenvoeging geeft het einde aan van het conditioneel gedrag dat gestart was door de vertakking.

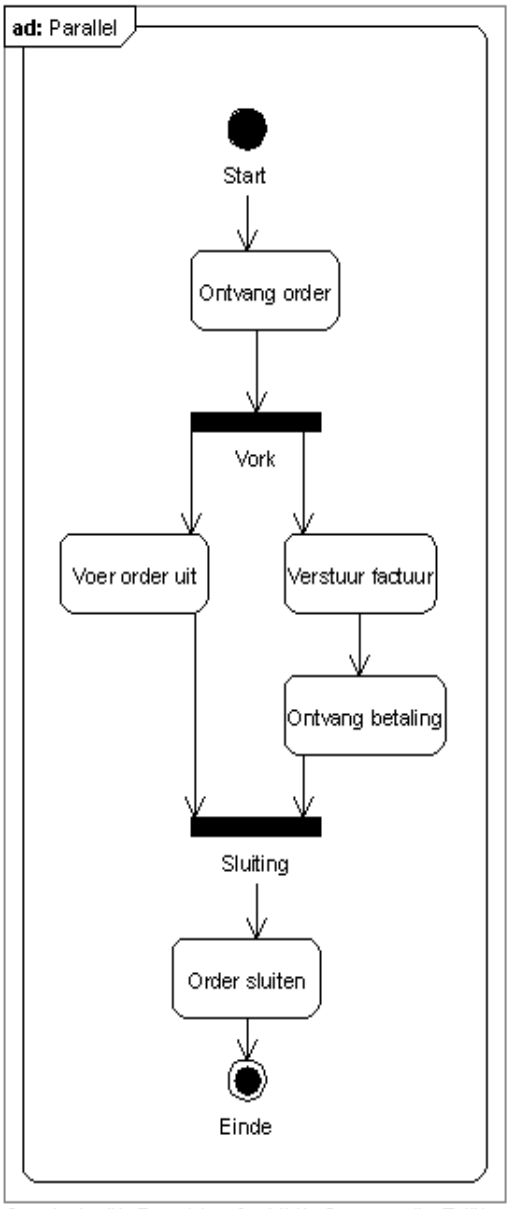


6.2 Parallel gedrag

Parallel gedrag wordt beschreven met vorken en sluitingen.

Een vork heeft één binnenkomende overgang en verschillende uitgaande overgangen. Zoals het type gedrag reeds zegt worden de verschillende uitgaande overgangen parallel uitgevoerd. In de essentie wilt dit vooral zeggen dat de volgorde waarin de verschillende taken worden uitgevoerd niet van belang is. In het onderstaande voorbeeld wil dit dus zeggen dat het uitvoeren van een order, het versturen van de factuur en het ontvangen van de betaling in een willekeurige volgorde kan gebeuren.

Het synchroniseren van de twee of meerdere parallele wegen gebeurt met een sluiting. In het onderstaande voorbeeld zullen we een order bijvoorbeeld niet sluiten alvorens de betaling is ontvangen en het order is uitgevoerd.



7 Samenvatting

UML kan in vele gevallen zeer handig zijn bij het tot een goed einde brengen van een project. UML zorgt er voor dat er nagedacht wordt over hoe problemen moeten worden opgelost vooraleer men begint met het programmeren van deze problemen. Zo verminderd men de kans dat er hele stukken code moeten worden weggegooid omdat een bepaald probleem verkeerd is opgelost.

Hoe men UML gaat toepassen en hoe men alles gaat weergeven verschilt van persoon tot persoon of van bedrijf tot bedrijf. Niet tegenstaand zal een persoon die eens UML heeft kunnen "lezen" altijd UML in grote lijnen kunnen "lezen". Tenslotte gebruiken de meeste mensen wel een soort van standaard.

Het handige met UML-tools is dat sommige van deze tools ook direct de code kunnen genereren vertrekkende van een bepaald diagram.

8 Tools

- **Poseidon for UML**
  Gratis tool voor niet commercieel gebruik dat ik heb gebruikt voor het maken van de UML schema's in deze tutorial. Poseidon biedt ook de mogelijkheid om code te genereren vanuit een diagram. Een nadeel aan Poseidon is dat enkele muisklikken er voor kunnen zorgen dat een heel (niet gewenst) schema tevoorschijn komt.

- **Borland Together**
  Hiermee heb ik mijn eerste stappen in UML gezet. Een vrij duur programma, maar beschikt ook over de mogelijkheid om code te genereren aan de hand van een schema en pas schema's aan aan de hand van zelf geschreven code. Is iets gebruiksvriendelijker dan Poseidon en beschikt over een betere GUI.

9 Referenties

- Martin Fowler & Kendall Scott: "UML beknopt" ISBN: 90-430-0199-6