UML

- Introductie

artesis

technologie

Situering

- Huidige producten en services bevatten software componenten
- Software ontwikkeling proces gedreven door kost en time-to-market
- Organisatie vraagt een efficient en competitief software ontwikkelingsproces
- Gestructureerd werken
  - Op plan en volgens bepaalde systematieken
  - Het resultaat moet een product zijn met een vooropgestelde kwaliteit tegen een vooropgesteld tijdstip

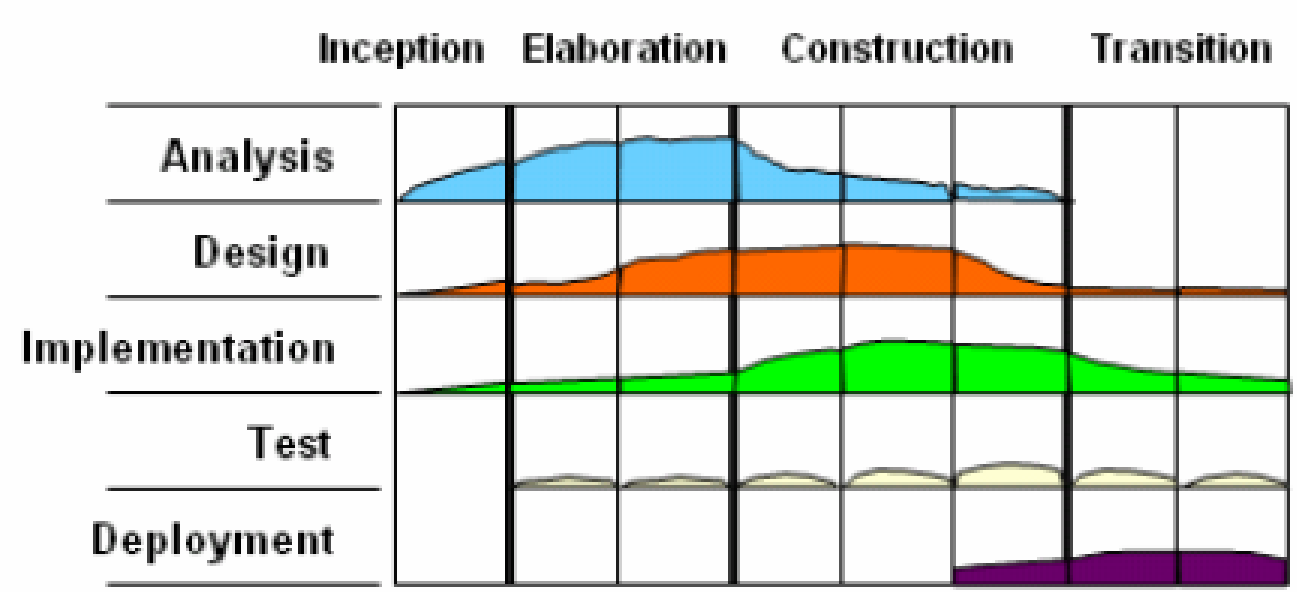
Wat is nu UML

- Is een gestandaardiseerde taal voor **specificatie**, **constructie** en **documentatie** van software.

- De Unified Modeling Language (UML) probeert bouwplannen in de wereld van de software te introduceren. De UML is een industriestandaard modelleertaal. Deze taal bestaat uit een
  - aantal grafische notaties waarmee u de hele architectuur van uw software kunt beschrijven.
  - Het bestaat dus uit notaties voor het beschrijven van alle aspecten van de softwareontwikkeling.

- Programmeurs, softwarearchitecten en analisten gebruiken m*odelleertalen om het ontwerp van software grafisch te beschrijven.*

Het Unified Process



- We onderscheiden 4 fasen in een project ontwerp. Elk van de 4 fasen heeft een bepaalde hoeveelheid analyse , design en testing
- Inception = feasibility study: het bepalen van de functionaliteiten en de scoop
- Elaboration : begrip van probleem : het specifieren van de architectuur
- Construction: het bouwen van de oplossen
- Transition : installatie van de oplossing: overdracht van het systeem aan de gebruiker

De workflows binnen het Unified Proces

- Analyse : bepalen wat het systeem moet doen
  - Begrijpen van het systeem
  - Wat is nodig
- Design : maak een technische specificatie
  - Hoe werkt het systeem technisch / intern
  - Perfectioneert en werkt analyse verder technisch uit
- Implementation:
  - Coderen van de design in een OO taal

Belangrijke modellen

- UML gebruikt verschillende modellen om het statisch en dynamisch gedrag van het systeem te modelleren. Vb.:
- Klassediagram
  - Laat zien welke klasse er zijn ontworpen
  - Welke attributen en bewerkingen of methodes deze klasses hebben
  - Welke relaties tussen deze klassen bestaan

  Het is dus een weergave van de statische structuur van een programma

- Sequentiediagram
  - Weergave van het berichtenverkeer tussen de verschillende klassen, dat samenhangt met een specifieke gebruiksmogelijkheid. Een sequentiediagram laat dus iets zien van de werking van een programma

- Use Case Diagram + Activity Diagram
  - Interactie tussen gebruiker en systeem (de te realiseren functionaliteit). (Een Use Case is de functionaliteit gezien door de gebruiker.)

In Detail van de belangrijkste

- **Klasse diagramma**
  - Een klasse definieert de attributes en de methoden van een set objecten. Alle objecten die van deze klasse afgeleid zijn (instanties van deze klasse) hebben eenzelfde gedrag, en hebben overeenkomstige verzamelingen met attributen (ieder object heeft zijn eigen set).

- **Sequentie diagram**
  - Volgordediagrammen geven de berichtenuitwisseling weer (bijv. methode-aanroep) tussen verscheidene objecten in een specifieke tijd-begrensde situatie. Objecten zijn instanties van klasses. Volgordediagrammen leggen speciale nadruk op de volgorde waarin en de tijdstippen waarop de berichten naar de objecten verstuurd worden.

- Use Case
  - Bepaalt welke actor interrageert met het systeem

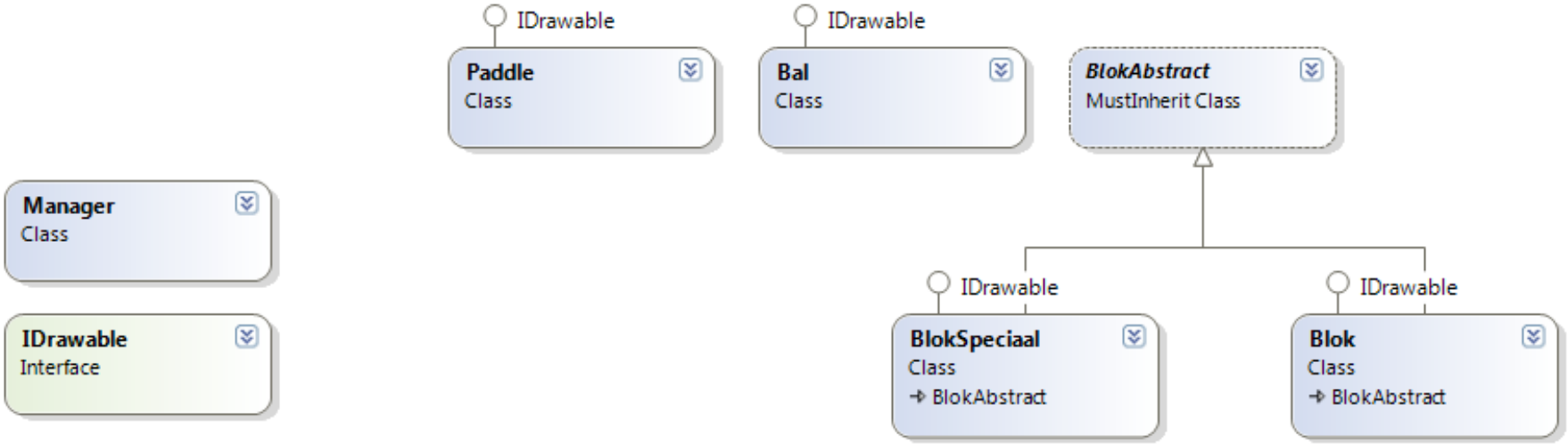
Klassediagram

- Coderen is feitelijk het laagste niveau aan documentatie voor uw software. (werkt de software, dan weet u zeker dat uw ontwerp gedocumenteerd is)

- => Maar voor vele mensen is dit niet begrijpbaar! Daarom ontwerp maken dat mensen in 1 oogopslag een begrip hebben van de structuur van de klassen
  - Op hoog niveau
  - Op detail niveau

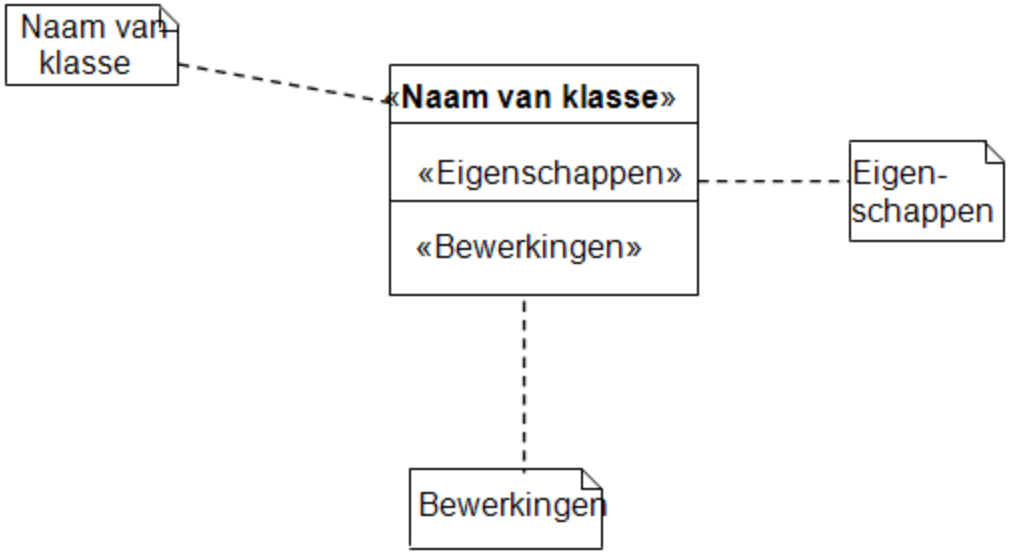
- Er een abstractie van het probleem wordt gemaakt
- Het is een blauwdruk voor ontwikkelaars
- Men krijgt beter inzicht in de structuur en organisatie indien er vb. onderhoud op bestaande software componenten moet gedaan worden.

Klassediagram voorbeeld



Basisnotatie voor klassen

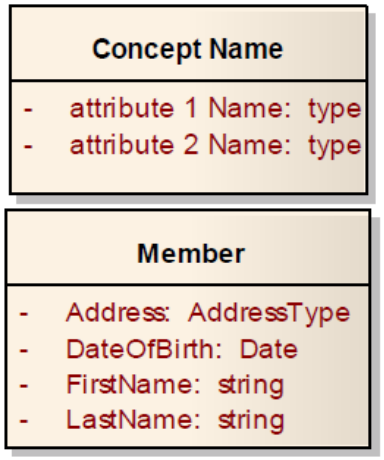
- Een klasse wordt in de UML gerepresenteerd door een vak.
  - Het bovenste vak bevat altijd de naam (in het vet gedrukt).
  - Het middelste vak bevat alle eventuele eigenschappen
  - Het onderste vak bevat bewerkingen of methoden



Basisnotatie voor klassen (2)

- **Attribuut / Properties**

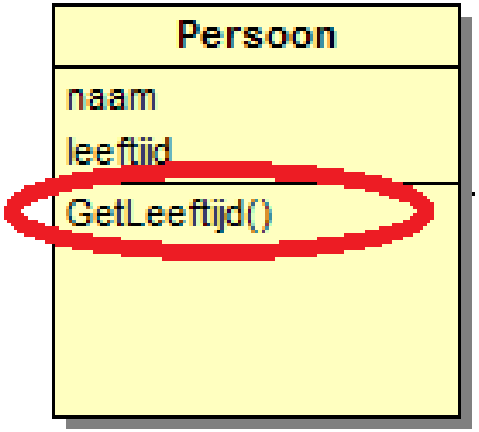
- = informatie die door een object beheerd wordt naam1,naam2,…)
- Ieder attribuut heeft precies **één waarde** voor ieder object van de klasse.
- De waarden van al de attributen representeert de toestand van een object. twee of meer objecten uit dezelfde klasse waarvan op een gegeven moment de waarden van al hun attributen dezelfde is, zijn in dezelfde toestand, maar blijven altijd verschillend.



Basisnotatie voor klassen (3)

- **Operatie**

  = een service die een object levert. De verzameling operaties bij een object representeert het **gedrag.** Een operatie kan argumenten hebben en een resultaat opleveren.

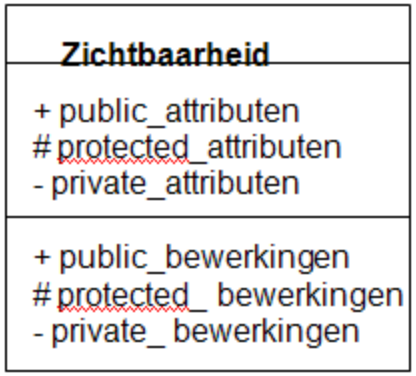


Basisnotatie voor klassen (4)

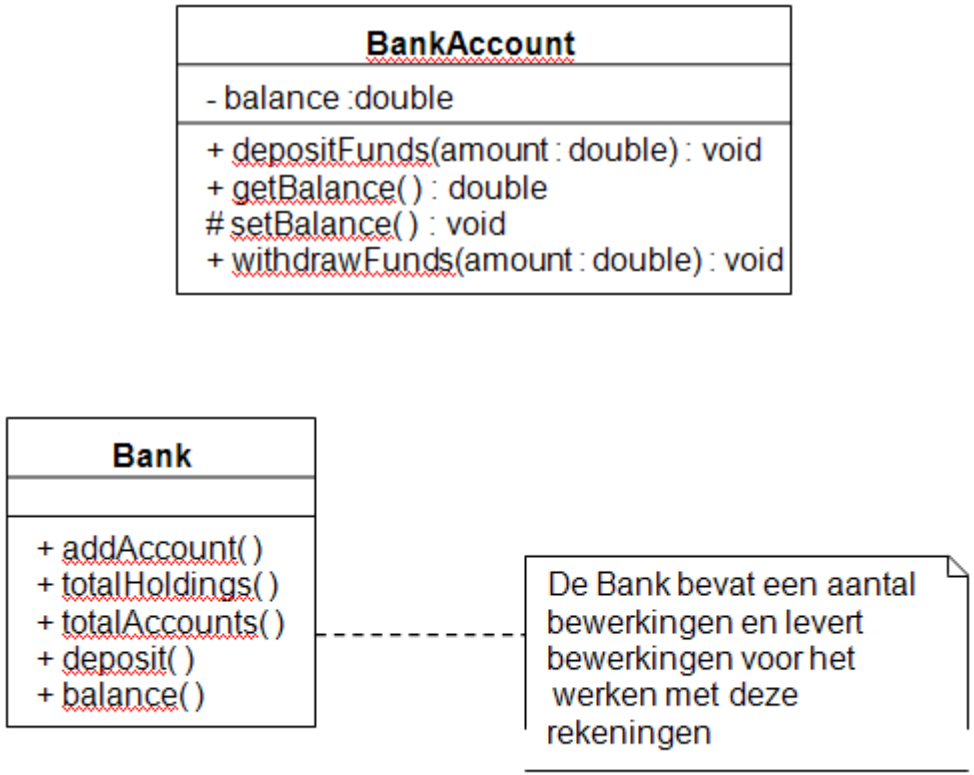
- Voor de eigenschappen wordt volgende syntax gehanteerd:
  - Naam [:type] [=waarde]
  - *(alles tussen rechte haken mag weggelaten worden)*
  - Naam en type geven de naam en het type weer
  - Waarde geeft een standaardwaarde aan die het attribuut krijgt bij initialisatie

- Voor de methoden wordt volgende syntax gebruikt:
  - Naam([parameterlijst]) [: resultaattype]

- Volgende tekens kunnen gebruikt worden voor de *zichtbaarheid*
  - - : private
  - +: public
  - #: protected



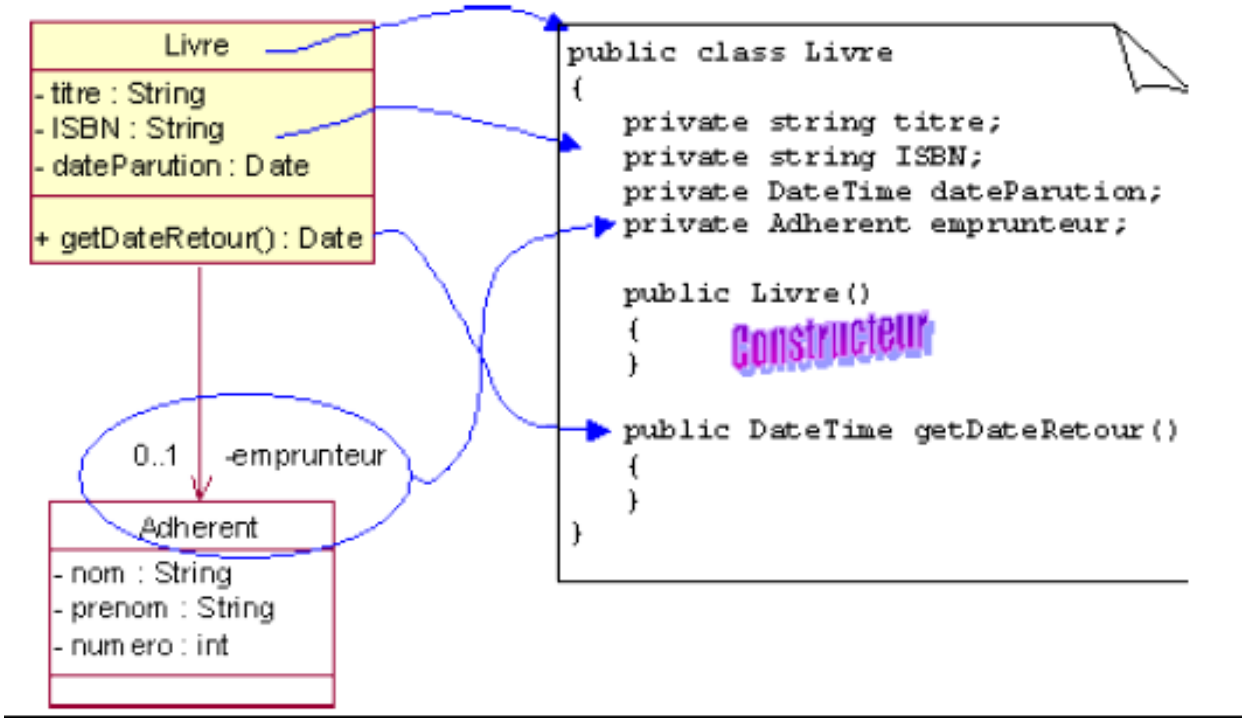
Voorbeeld klassediagram



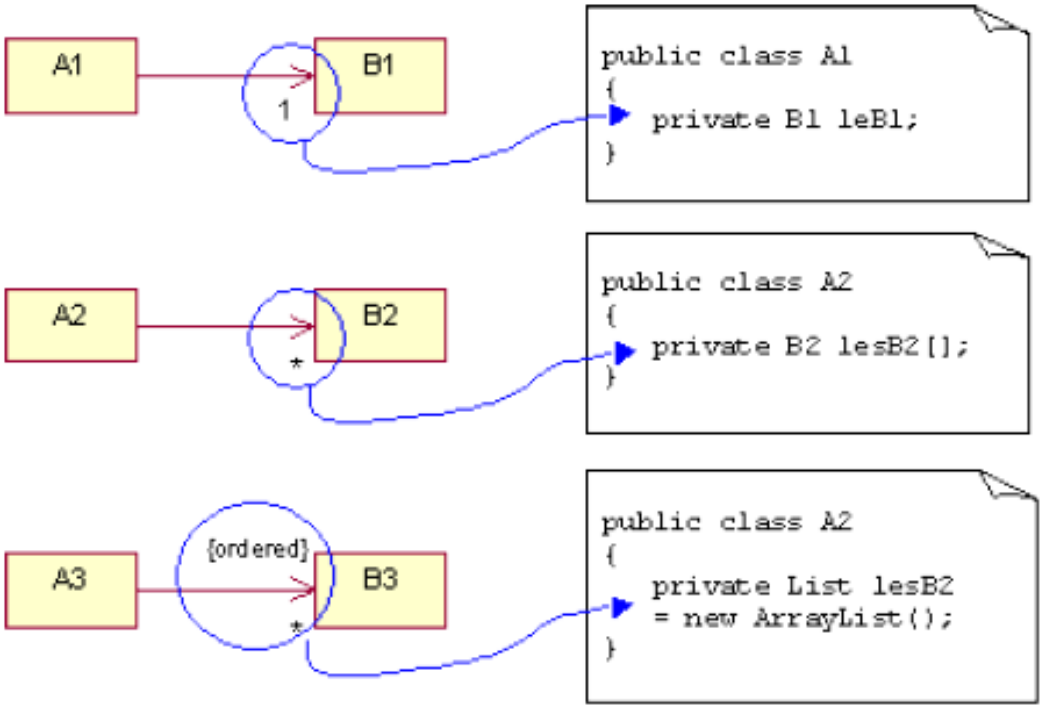
Relaties tussen klassen modelleren

- Een *relatie* beschrijft hoe klassen samenwerken. Een relatie wordt in de UML aangegeven met een verbinding tussen 2 of meer elementen in de notatie.
- Volgende relaties tussen objecten:
  - **Associatie**: geeft aan dat een object een ander object bevat
  - **Generalisatie:** Overerving is een van de fundamentele concepten van objectgeoriënteerd programmeren, waarbij een klasse “toegang krijgt tot” (bijna) alle attributen en operaties van de klasse waar het van erft, en sommige ervan opnieuw kan implementeren (overriding), alsook meer attributen en operaties van zichzelf kan toevoegen.

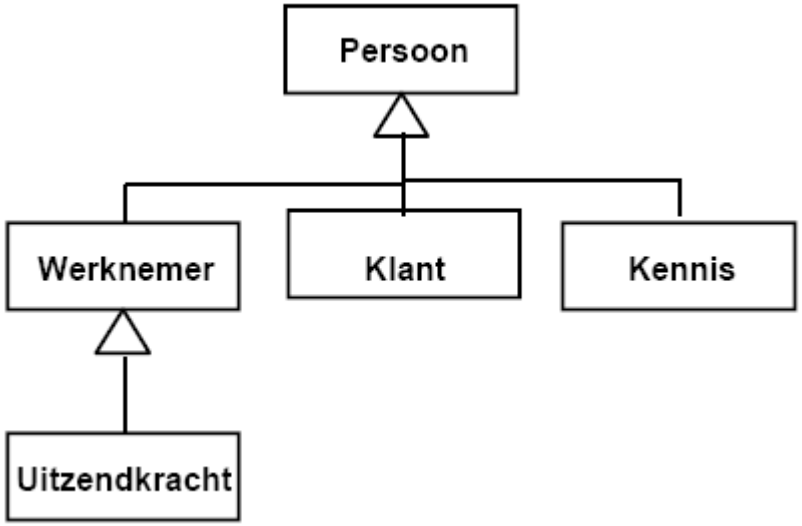
Naar code toe .. (associatie)



Associatie naar code..



generalisatie



Tips voor het effectief modelleren

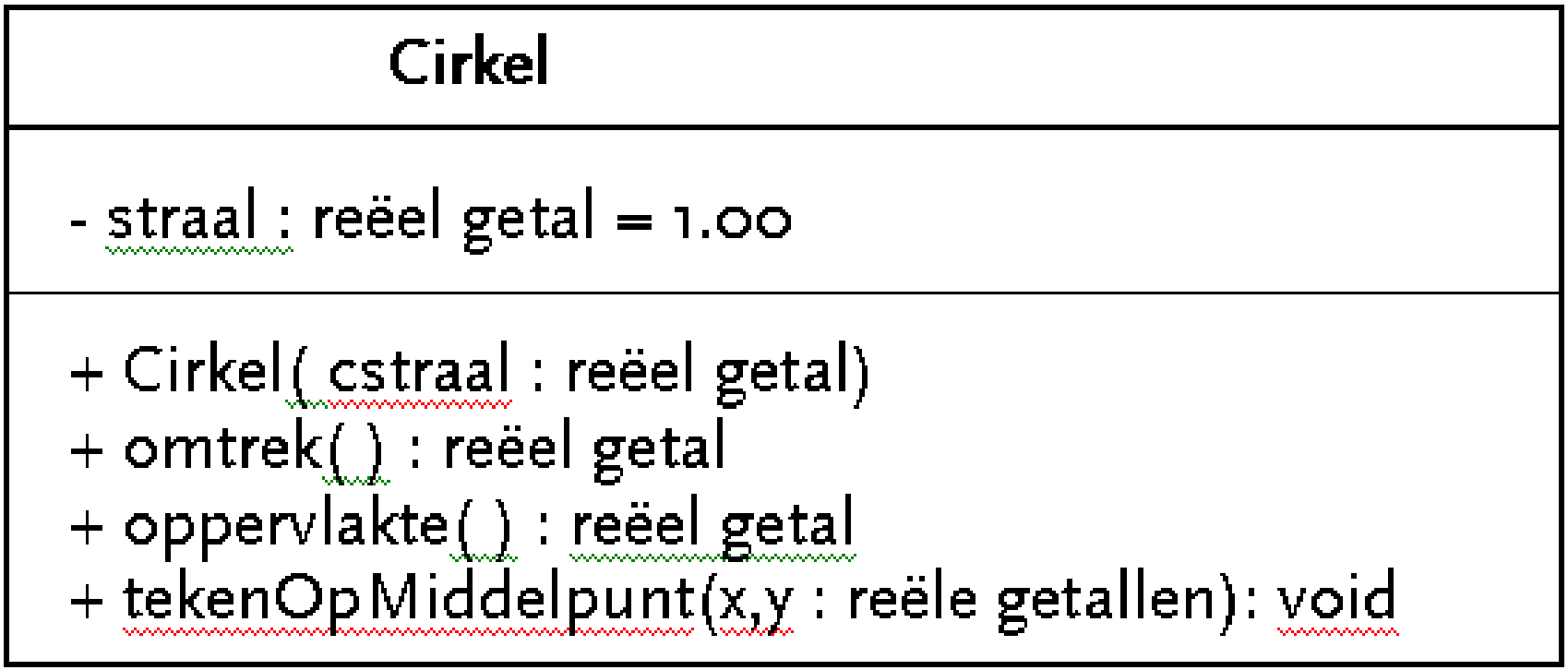
- Probeer altijd het eenvoudigste model te maken dat er nog in slaagt uw ontwerp over te dragen.

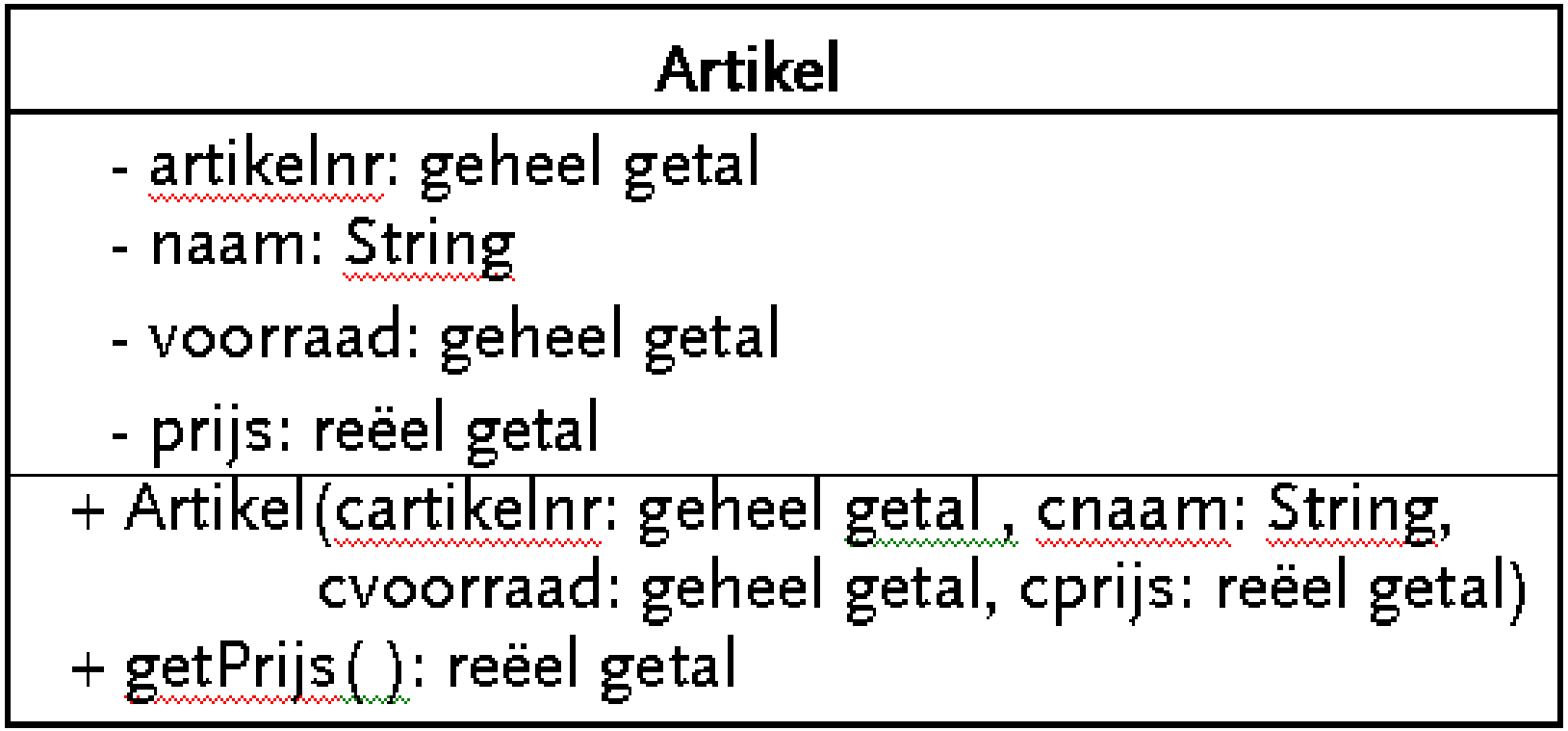
- UML is slechts een hulpmiddel dat tot doel heeft u te helpen uw ontwerp over te dragen. U zult nog steeds code moeten schrijven!

Oefeningen

- Gegeven een klasse Cirkel, met een attribuut straal met standaardwaarde 1. De klasse heeft methoden om de omtrek en het oppervlak van de cirkel te berekenen, alsmede een methode om de cirkel in een vlak te plaatsen met het middelpunt op gegeven coördinaten. Teken een klassediagram waarin deze informatie over attributen en methoden is opgenomen.

- Maak een klasse Artikel. Hiermee wordt een artikel bedoeld die bijvoorbeeld in een winkel verkocht wordt. Deze klasse heeft attributen artikelnummer, prijs, naam en voorraad. De klasse beschikt eveneens over een methode getPrijs( ). Teken een klassediagram waarin deze informatie over attributen en de methode is opgenomen.


Cirkel
- straal : reëel getal = 1.00
+ Cirkel(cstraal : reëel getal)
+ omtrek( ) : reëel getal
+ oppervlakte( ) : reëel getal
+ tekenOpMiddelpunt(x,y : reële getallen): void


Artikel
- artikelnr: geheel getal
- naam: String
- voorraad: geheel getal
- prijs: reëel getal
+ Artikel(cartikelnr: geheel getal , cnaam: String,
cvoorraad: geheel getal, cprijs: reëel getal)
+ getPrijs( ): reëel getal

Requirements

- Functionele eisen van een systeem
- Testpaden
- Niet-functionele aspecten (denk bijvoorbeeld aan security, performantie,…)

*Veelal gedefinieerd in **use cases** met bijhorende **tekstuele uitleg** of **activity diagram***

Waarom: Use case (UC) diagrammen

- Use Cases zijn eenvoudig te begrijpen voor de niet-informaticus, en wordt vooral gebruikt tijdens de analysefase ter bepaling van de functionaliteiten van het te bouwen systeem.

- UC's beschrijven de gewenste functionaliteit van het systeem NOOIT hoe die functionaliteit moet worden geimplementeerd of hoe het systeem intern zal werken

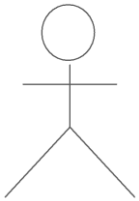
Een use-case

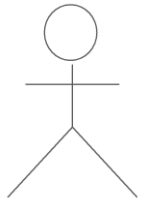
- Een use-case is een beschrijving van de interactie tussen één of meer actoren en het systeem, anders gezegd:
- Een use-case is een scenario, dat een doelstelling van een gebruiker met het systeem ondersteunt.

Vb. Een rekening openen, een order plaatsen, een formulier invullen

➔ Dus een Use case is een functionaliteit van het systeem als reactie op actor !

Het begrip ACTOR

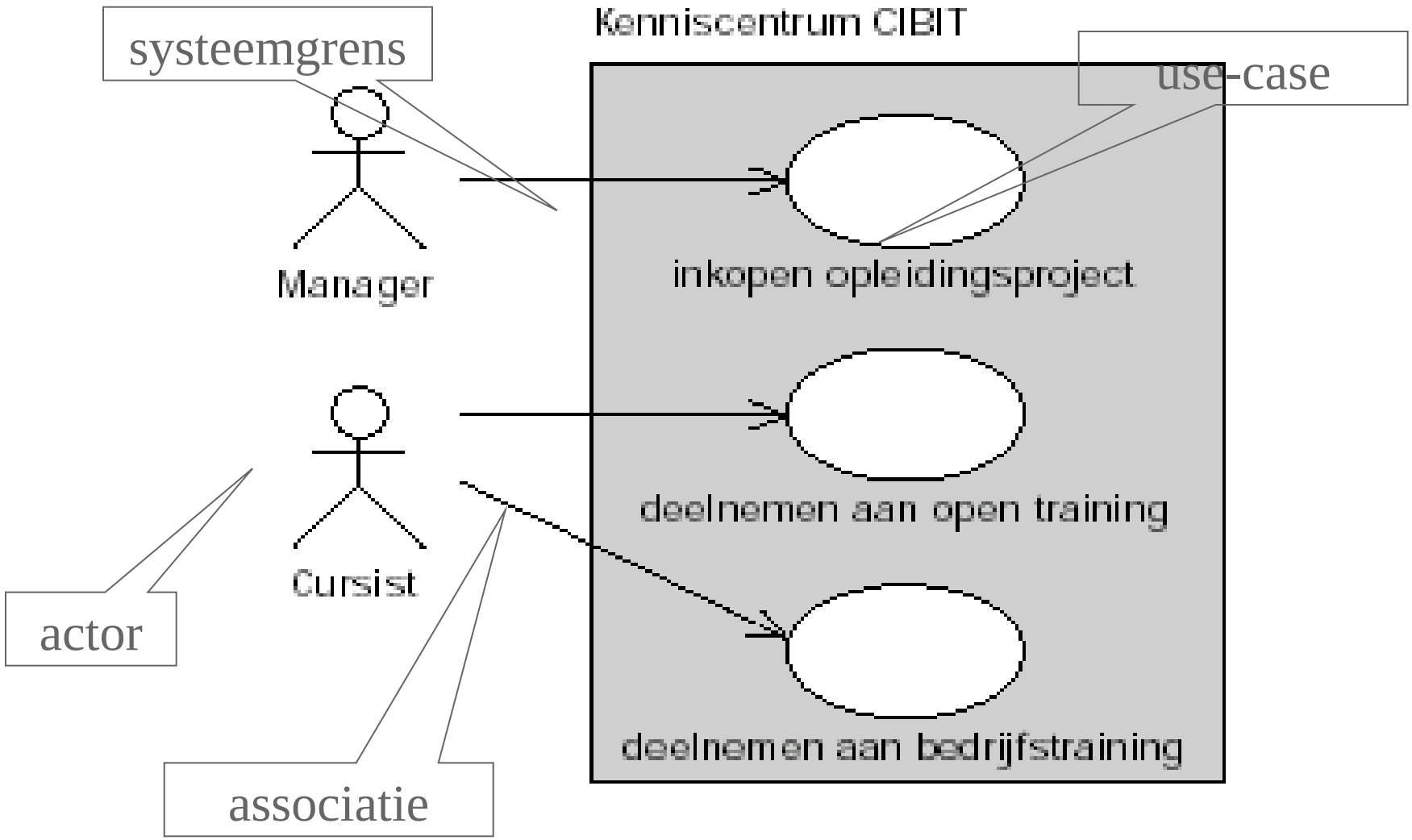




- Een actor is een entiteit die buiten het systeem staat en direct communiceert met het systeem.

- Een actor is een rol die iets of iemand in de context van een systeem speelt.

Het use-case diagram



Use-cases stappenplan

Definitie: een verzameling van door een systeem uitgevoerde sequenties, die voor een bepaalde actor een waarneembaar waardenresultaat opleveren.

(Waardenresultaat: communicatie, berekeningen, andere werkzaamheden.)

a. Identificeer de grenzen van het systeem en vind de actoren.
b. Definieer use-cases voor iedere actor.
c. Bepaal per use-case de aannamen of precondities.
d. Bepaal per use-case de interactie.
e. Bekijk per use-case mogelijke uitzonderingen.
f. Beschrijf elke use case en maak het use-case diagram.

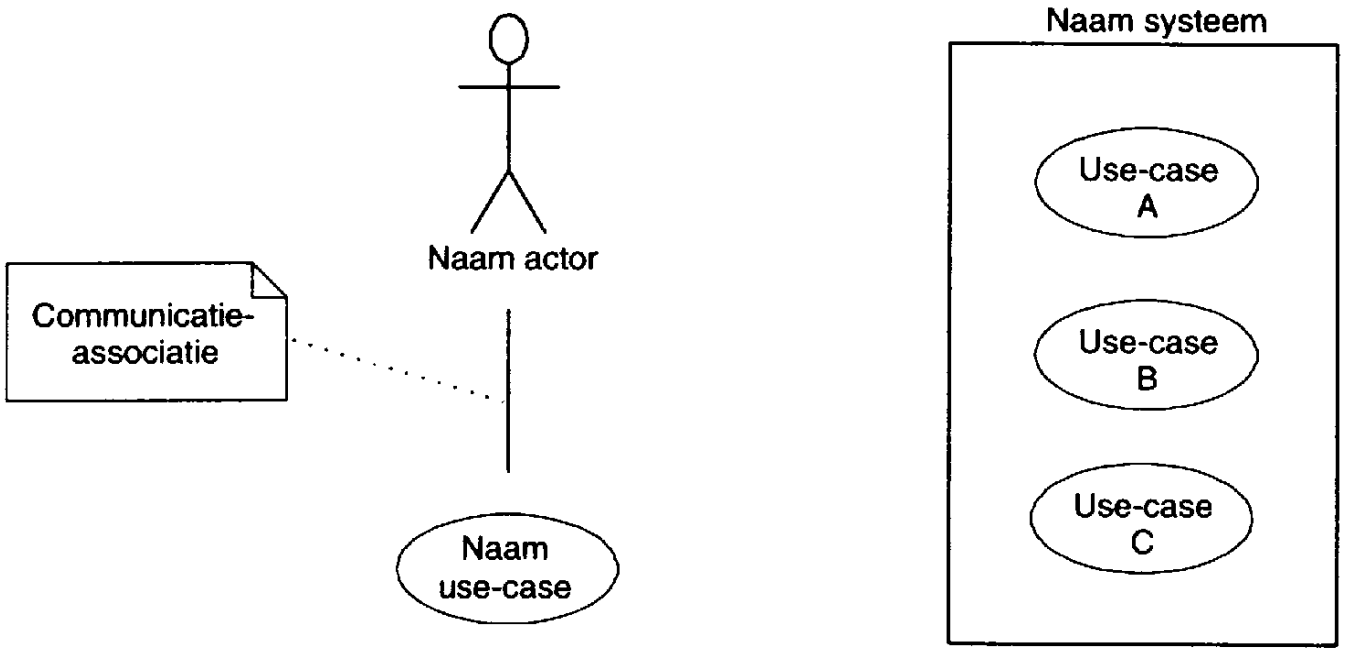
De beschrijving van use-cases

Elke use case wordt verder gedetailleerd.

- of Tekst
- of Activity diagram

Naam	Rekening toevoegen.
Samenvatting	Er wordt een nieuwe rekening aangemaakt voor een klant.
Actoren	Baliemedewerker.
Aannamen	Baliemedewerker heeft beschikking over NAW gegevens van de klant.
Beschrijving	(1) De baliemedewerker maakt aan het systeem bekend dat een nieuwe rekening aangemaakt moet worden en geeft de NAW gegevens van de klant. Als de klant een bedrijf is dan wordt ook het kamer van koophandel nummer ingevuld. (2) Het systeem checkt of de klant al bekend is. Is dit het geval dan worden de klantrekeningen gecontroleerd op roodstand. Als de klant roodstaat dan treedt een uitzondering op. Het systeem maakt het nieuwe rekeningnummer aan de baliemedewerker bekend.
Uitzonderingen	[Roodstaan] Als één rekening van de klant roodstaat dan wordt hiervan door het systeem melding gegeven. De baliemedewerker kan dan naar de use-case “Storten” overgaan om de klant de gelegenheid te geven het saldo aan te vullen. Als het saldo is aangevuld, dan wordt de rest van deze use-case uitgevoerd.
Resultaat	De klant heeft minstens één rekening.

Use-cases in UML



UC: probleemstelling

- Een supermarkt wil een geautomatiseerd kassasysteem bouwen. In dit nieuwe systeem zal de klant zich eerst dienen te identificeren door middel van een klantenkaart. De kassier leest de gegevens van de klantenkaart door middel van een barcodelezer. Naam - en adresgegevens worden opgehaald uit het klantenbestand. Eventueel worden deze gegevens manueel ingebracht of verbeterd. Daarna worden de gekochte artikelen ingebracht. Dit gebeurt door het inlezen van de barcode van het product. De prijs wordt opgehaald uit het artikelbestand en wordt automatisch op de factuur afgedrukt. Eventueel kan deze prijs manueel worden ingebracht of gewijzigd door de kassier. Dagelijks zal de manager een lijst afdrukken van de verkopen binnen zijn supermarkt.

UC : actoren zoeken

- Een actor is iemand of iets die van het systeem gebruik maakt en informatie zal uitwisselen met het systeem. (een actor hoeft niet altijd een persoon te zijn (machine,..))
- Symbool = “stokventje”

- Volgende vragen kunnen helpen op actors van een systeem te bepalen:
  - Wie zal er gebruik maken van de belangrijkste functionaliteiten van het systeem
  - Wie heeft de functionaliteit nodig om de dagelijkse taken te vervullen
  - Wie zal het systeem onderhouden en operationeel houden
  - Wie heeft er belangstelling voor de resultaten die door het systeem worden geproduceerd
  - ..

UC: actoren kassasysteem

Actor	verklaring
Klant	De klant heeft geen directe interactie met het systeem, dus is daarom geen actor
Kassier	Brengt klanten en productgegevens in en wordt dus beschouwd als een actor
Manager	Drukt dagelijks resultaten af en is dus manager
Computer	Indien gegevens worden geleverd door een reeds bestaande toepassing , kan de computer als actor beschouwd worden, maar dit is hier niet het geval, dus daarom geen actor

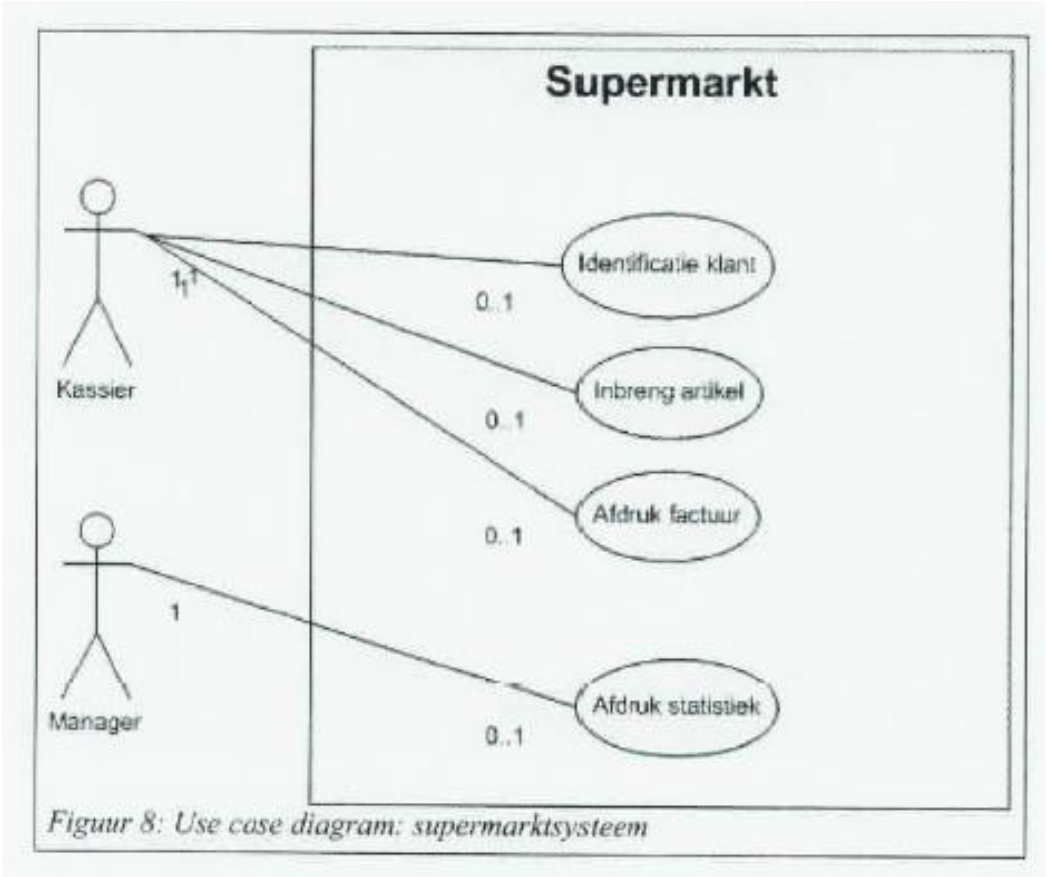
UC

- Een klassieke fout bij het bouwen van “Use Case Diagrams” is dat een analist “Use Cases” gebruikt om programmeerfuncties aan te duiden.

⇒DIT IS NIET DE BEDOELING: Use cases dienen te verwijzen naar de algemene functionaliteit van het systeem en niet naar de manier waarop de implementatie zal gebeuren!

Volgende vragen kunnen helpen om use cases te bepalen:

- Welke functies verwacht de actor van het systeem
- Moet de actor gegevens creeren, wijzigen, vernietigen,..
- Welke input moet de actor geven aan het systeem
- Welke output krijgt de actor van het systeem
- Welke gebeurtenissen zijn voor de actor van nut



Figuur 8: Use case diagram: supermarktsysteem

Relaties in use cases

- Include => een use case maakt gebruik van een andere use case
  - Voorgesteld door onderbroken pijl

- *(Extend => gedrag van de use case wordt uitgebreid)*

Use Case in software architectuur

- Uitvoeren van een Use Case:
  - Preconditie checken
  - Output is Postconditie (uitvoeren van een stappenplan)
  - => Naar uniformiteit wordt er *soms* gebruik gemaakt van task patroon : elke use case wordt een klasse en afgeleidt van class Task *(misschien niet de ideale manier voor je game design, maar dit verduidelijkt onderhoudbare code schrijven!)*

```
public abstract class Task
{
    protected abstract bool Validate(Task t);
    protected abstract bool Execute(Task t);

    public bool Run(Task t)
    {
        if (Validate(t))
            return Execute(t);
        return false;
    }
}
```

```
public class InzienProfiel: Task
{
    protected override bool Validate(Task t)
    {
        return true;
    }

    protected override bool Execute(Task t)
    {
        return true;
    }
}
```

```
InzienProfiel p = new InzienProfiel();


List<Task> aLIst = new List<Task>();
aLIst.Add(p);

foreach (Task t in aLIst)
{
    t.Run(t);
}
```

Tekstuele beschrijving use cases

- WAAROM:
  Use cases geven slechts een visueel schema van de belangrijkste functionaliteiten. Om latere discussies bij het opleveren en testen te vermijden **wordt elke use case meer gedetailleerd beschreven:**
- Voornaamste elementen in de tekstuele beschrijving:
  - Naam: de naam van de use case
  - Doelstelling: een samenvatting van de doelstelling van de use case
  - Actors: een opsomming van de actors die in relatie staan met de use case
  - Begintoestand / Preconditities: de toestand van het systeem bij start van de use case
  - Korte Beschrijving: samenvatting van de use case
  - Eindtoestand / Postconditie: de toestand van het systeem bij het einde van de use case.
  - Stappenplan: een gedetailleerde beschrijving van de volledige werking van de use case
  - Uitzonderingen

Tekstuele beschrijving UC: naam & doel

- Bijvoorbeeld een UC: Aanvragen abonnement:

Naam = Aanvragen abonnement

Doel = bezoeker vraagt een abonnement aan

⇒Het doel kan per actor verschillend zijn: bijvoorbeeld voor UC “inzien profiel” bij een dating website betekent dit dat deze abonnee een profiel bekijkt om een partner te vinden, terwijl een bezoeker een profiel inziet om te kijken of een abonnement wel zin heeft.

Daarom per actor doel aanduiden -> in het stappenplan van de tekstuele beschrijving zullen we doelen afzonderlijk behandelen.

Tekstuele beschrijving: korte beschrijving / pre condities / post condities

- Vb.:

Use case: “zoeken profiel”

Korte beschrijving: de applicatie toont een webpagina. De actor vult criteria in. De applicatie toont een lijst met passende profielen. Als er geschikte profielen zijn, selecteert de actor er een.

- Precondities = de voorwaarden waaronder de use case geldig is

Vb:

Use case: versturen bericht

Precondities: profiel is bekend

Actor is ingelogd

Tekstuele beschrijving: Postcondities

- Postcondities = beschrijving van dat deel van de uitkomst van de use case dat voor de buitenwereld van belang is

Vb:

Use case: Zoeken profiel

Postconditie

Actor heeft bevestigd en profiel is bekend of actor heeft geannuleerd

=> Het woordje “ of ” geeft aan dat er meerdere uitkomsten zijn.

Stappenplan

- Het stappenplan van een use case beschrijft de opeenvolging van acties die worden ondernomen om de postcondities waar te maken.

- Een streven om het stappenplan kort en bondig te houden!

Onderwerp + werkwoord + [lijdend voorwerp] + [additionele bepalingen]

Vb. use case: versturen bericht

Precondities:	profiel is bekend
	Actor is ingelogd
Postcondities:	bericht is verstuurd aan profiel of actor heeft geannuleerd
Stappenplan:	haal profiel op
	toon webpagina
	actor typt nieuw bericht in
	actor bevestigt versturen bericht
	Applicatie verstuurt bericht aan profiel
	Actor krijgt bevestiging van versturen bericht

Stappenplan – 2

- Het komt voor dat in een stappenplan een andere use case wordt uitgevoerd:
  - Benoem dit dan vb. **voer uit “Use Case x”**

Beslismomenten in het stappenplan

- Vb. in stappenplan:

- Als creditcard valide, sla abonnee en abonnement op
- Voer uit versturen bevesting per email
- als creditcard niet valide, toon melding.

- '*voer uit versturen bevestiging per email*' is niet eenduidig beschreven: is deze stap nu conditioneel of niet? (enkel mail versturen indien creditcard valide, of altijd? -> niet duidelijk)

➔Daarom bestaat er een ***activity diagram***

➔Maar dergelijke beslissingsmomenten kunnen we ook tekstueel weergeven :

Beslissingsmomenten

- Use case: inloggen abonnee
- Stappenplan:

1. Valideer aantal ongeldige logings
2. Als aantal ongeldige meer dan 2, stop
3. Toon webpagina
4. Actor voert login en paswoord in
5. Actor bevestigt
6. Applicatie valideert login
7. Als login geldig

7.1 Markeer actor als abonnee

7.2 stop

8. Als login ongeldig

8.1 verhoog aantal ongeldige logins

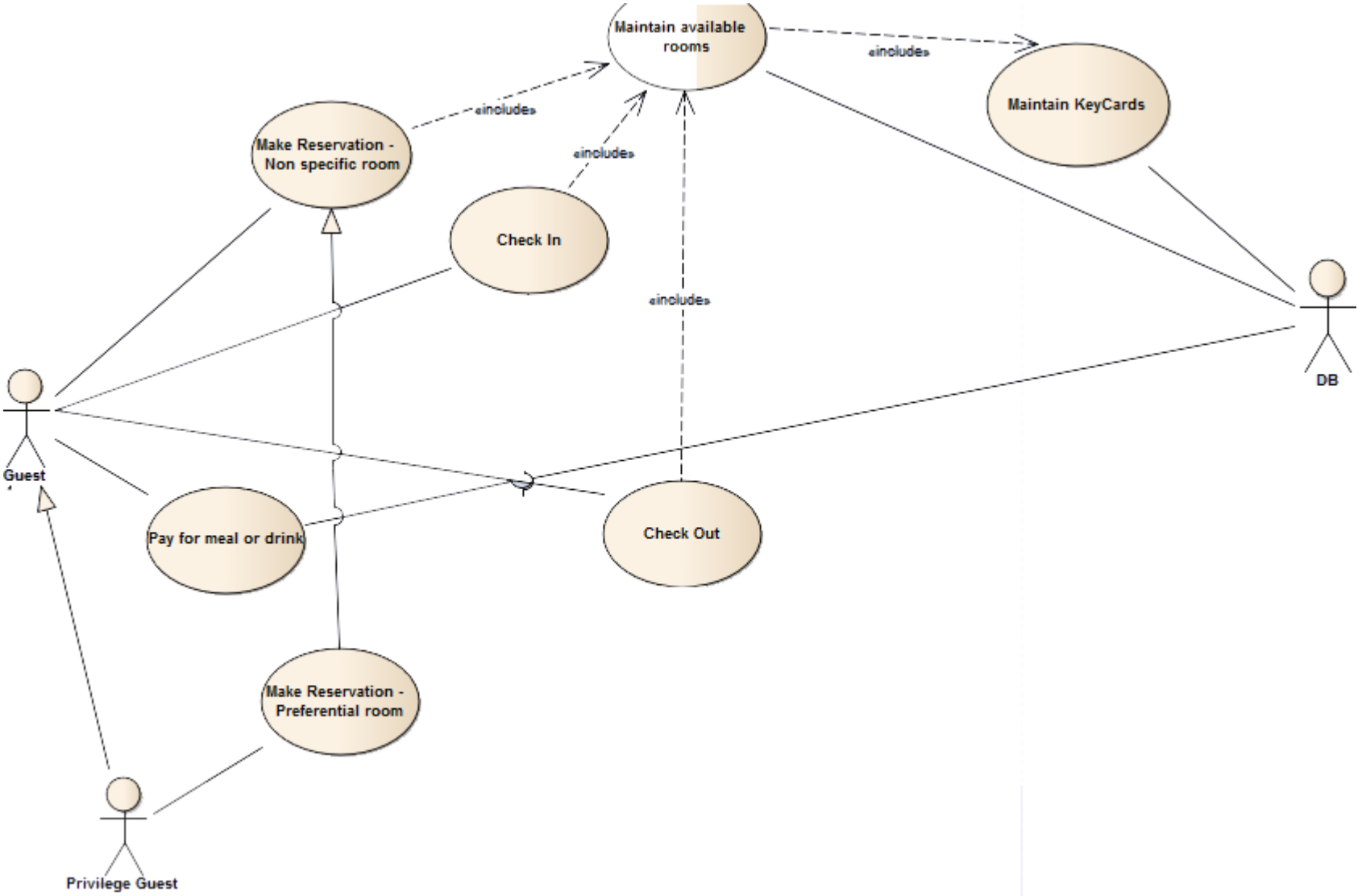
8.2 herhaal vanaf 1

Oefening identificeren van Use Cases

The hotel management system is used to deal with all room booking and allocations, and is mostly operated at the general reception desk.
Reservations for rooms made by phone or in person are entered into the system, allocating non-specific room(s) to a particular guest (eg. A guest books a double room, not room 104). On arrival the guest is given a choice of available rooms, and personal and payment details are confirmed.
The general guest is provided with a personal key card for identification purposes, and a magnetic door key
The management system is linked to all bar and restaurant tills, to enable the guest to charge drinks and meals to their room account, if required.
The system allows for the identification of privilege guest accounts, with added benefits to the guest, such as express check-in, check-out and preferential room-reservation bookings.

The development of the hotel management system will initially be limited to the room and guest management facilities, such as reservations, check-in room charging and check-out.

Oplossing


Maintain available rooms
«include»
Maintain KeyCards
Make Reservation - Non specific room
«include»
«include»
Check In
«include»
DB
Guest
Pay for meal or drink
Check Out
Make Reservation - Preferential room
Privilege Guest

Activity Diagram

- Daar bijvoorbeeld beslissingsmomenten, of uitzonderingen moeilijk tekstueel weer te geven zijn, kan er ook gebruik gemaakt worden van een “**Activity Diagram**”.

- Geven een gedetailleerde beschrijving van alle activiteiten die binnen een methode worden uitgevoerd
- Het activity diagram legt de focus op flows => te gebruiken in situaties waar alle events aanleiding geven tot het afwerken van een actie

- *Veelal gebruikte diagram voor testers*

Elementen van een Activity Diagram

- Startpunt: *zwart bolletje*

- Eindpunt van een activiteit: (alle mogelijke opeenvolgingen van acties in een activiteit zijn afgelopen)

  Zwart bolletje omringd door een cirkel
- Eindpunt van een reeks acties (einde van een enkele flow in activiteit aan te duiden)

  X in een cirkel

- Activiteit:

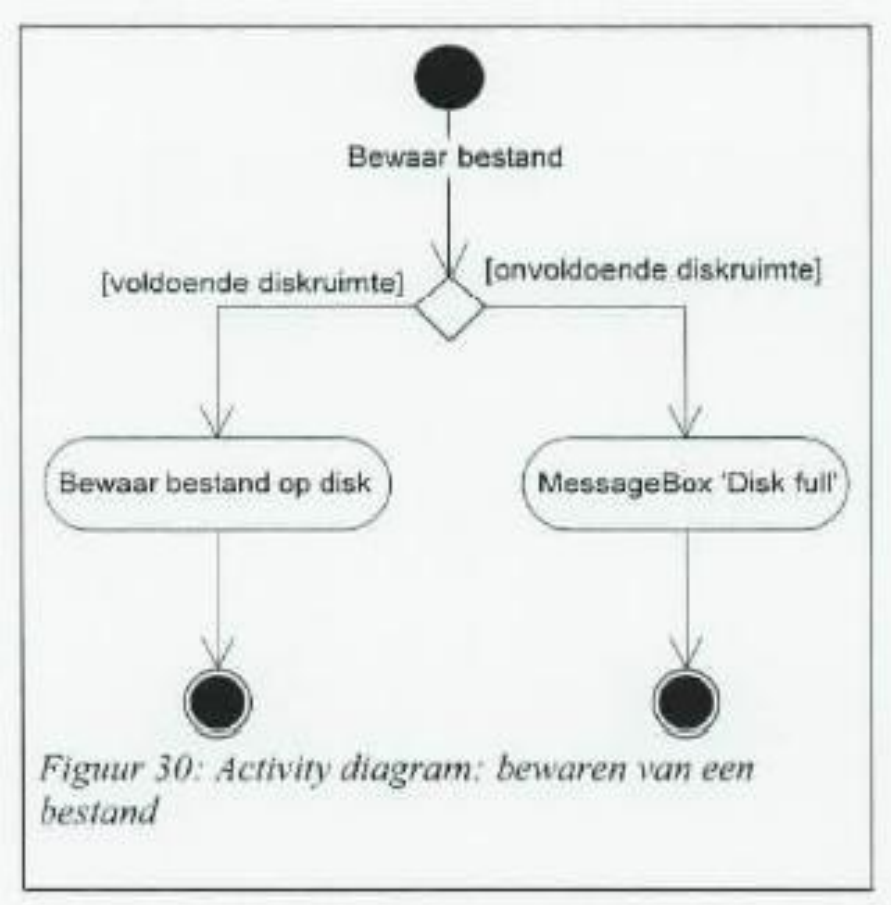
  Rechthoek met afgeronde hoeken
- Transitie:

  Pijl (het einde van een actie leidt naar een volgende actie)

- Beslissingssymbool

  ruit

Activity Diagram voorbeeld



Figuur 30: Activity diagram: bewaren van een bestand

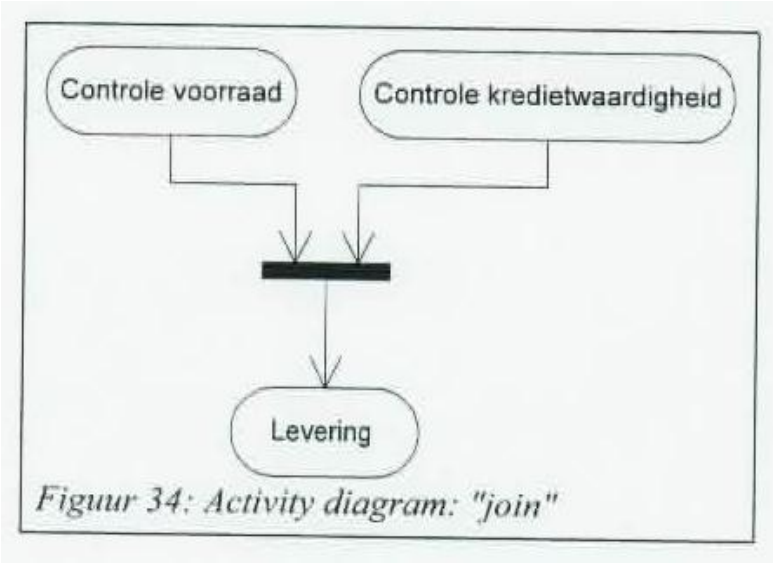
Fork & Join Nodes

- Activity diagrams zullen dikwijls gebruik maken van synchronisatiepunten: alles voor dit punt moet afgewerkt zijn vooraleer de activiteiten na het synchronisatiepunt opgestart kunnen worden.

=> Fork Nodes & Join Nodes: (fork = start voor parallel uit te voeren flows / Join =geeft aan dat parallelle flows gereed zijn (zie onderstaand voorbeeld)

  - Voorstelling door dikke horizontale lijn

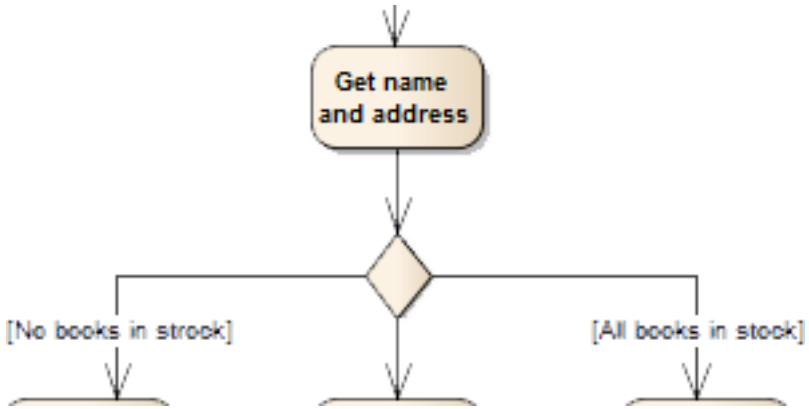
Onderstaande figuur worden 2 activiteiten parallel uitvoerd

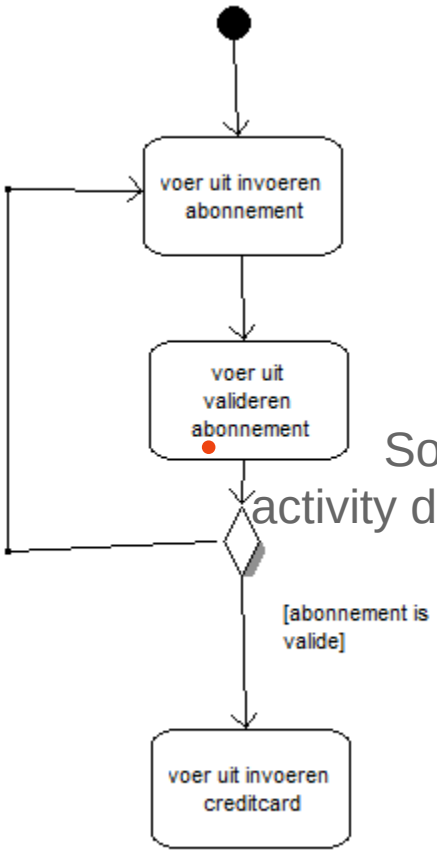


Figuur 34: Activity diagram: "join"

Activity Diagram - herhalingsmomenten

- Vrijwel iedere activiteit kan goed of fout gaan
  - ➔ Ga dus alle mogelijke uitkomsten na, zijn er meerdere uitkomsten?



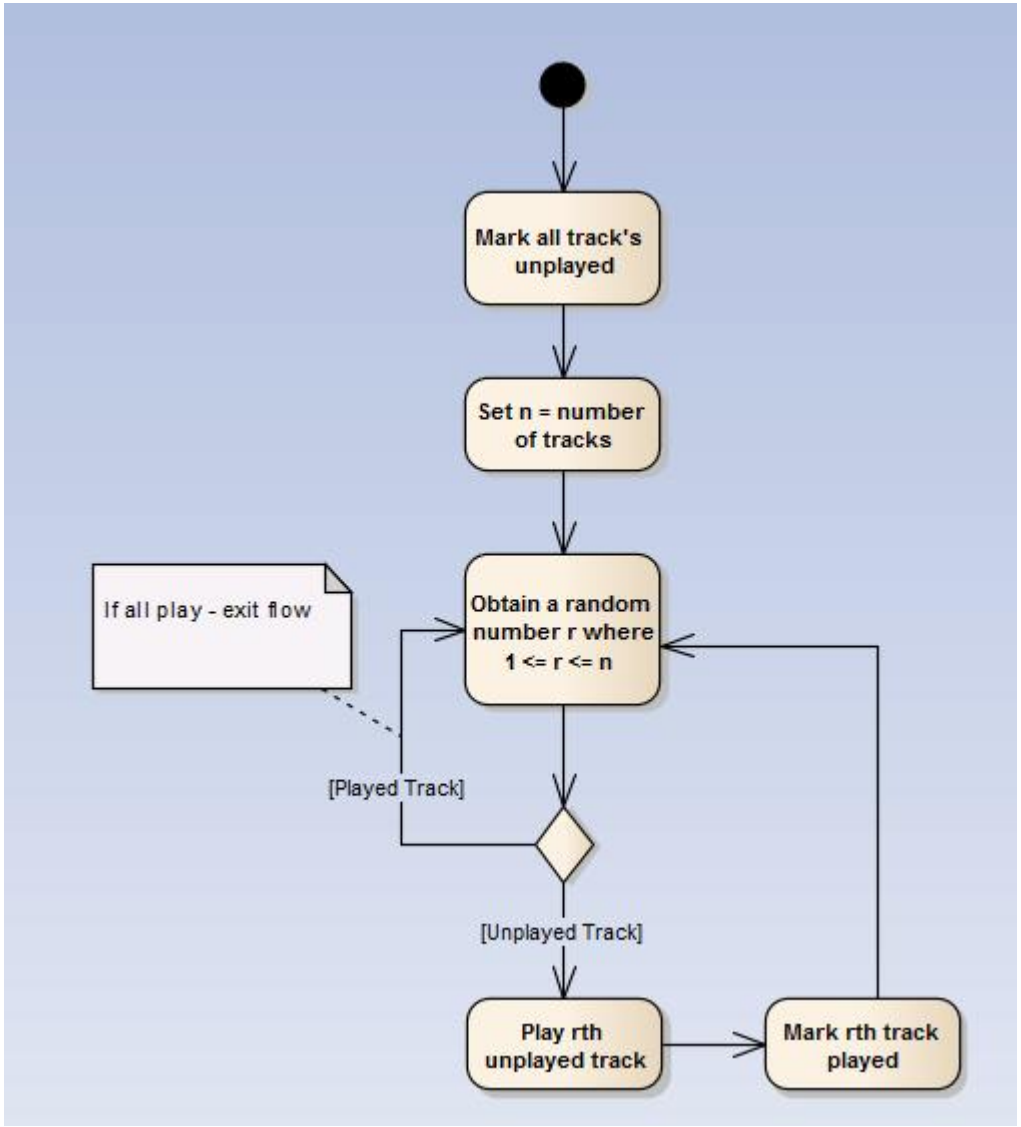


Soms is het nodig een of meer activity nodes te herhalen in een activity diagram (bijvoorbeeld je krijgt 3 pogingen om in te loggen..)

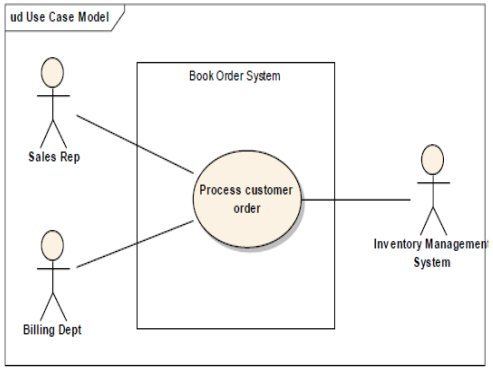
Oefening

- Maak een activity diagram van een shuffle (of random) functie van een mp3 speler.

Oplossing oefening 1



Oefening 2 : Modelleer “Activity diagram” voor onderstaande use case

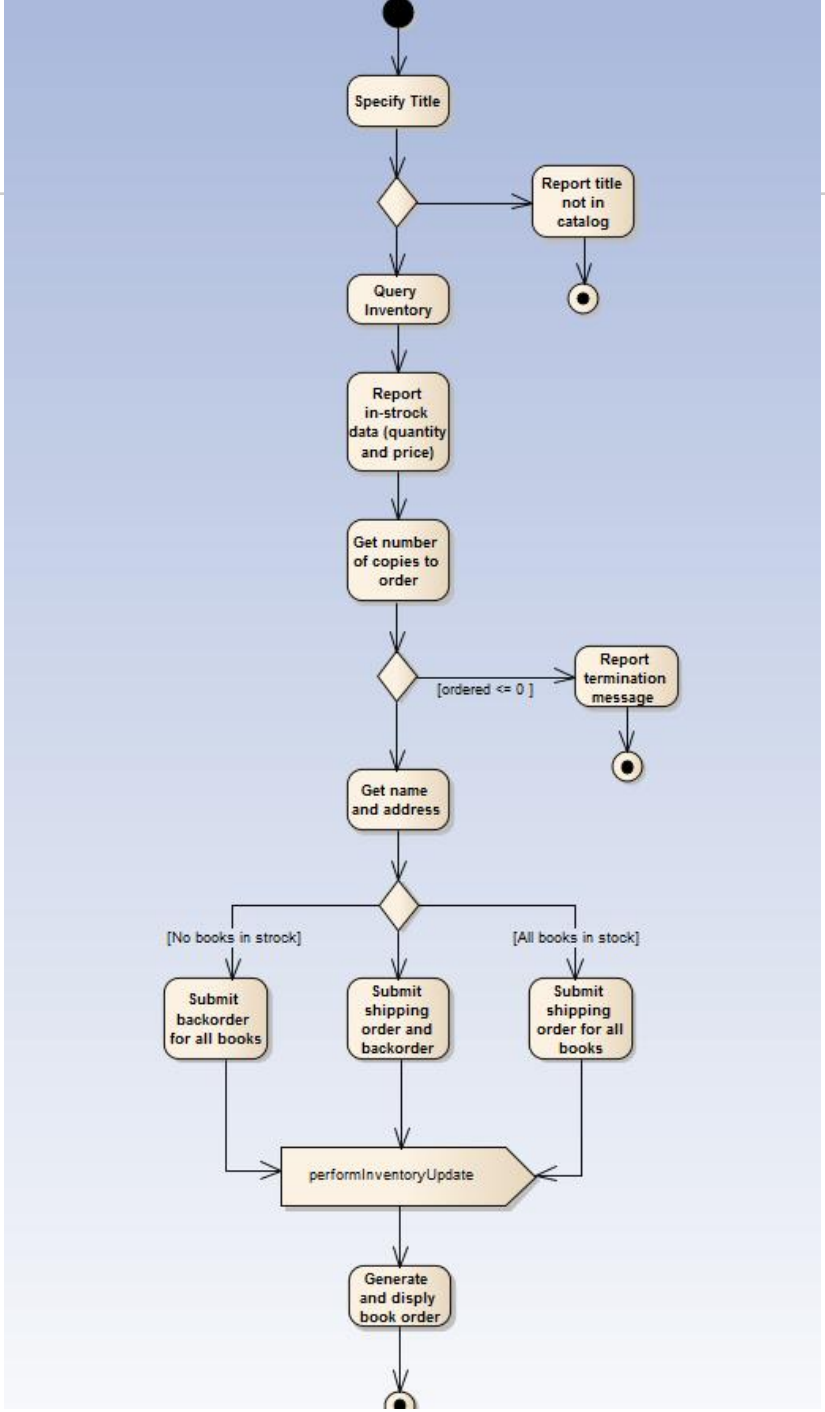


1. User (Sales Rep) is told to specify Book Title.
2. User enters Title.
3. Application queries InventoryManager determining if Title is in catalog
4. If Title is in catalog, Application queries InventoryManager to determine the ISBN, the price of the book, and the number of copies in stock.
5. Application reports the book price and the number of copies in stock to the User
6. Application requests the number of copies to be ordered.
7. User enters the number of copies.
8. Application requests the name and address to ship the books.
9. User enters name and address.
10. If all the needed books are in-stock, Application submits a shipping order for the books to the specified address. This order will update the inventory.
11. Application creates a book order, specifying the shipping name and address, the number of copies of the book that were backordered and the number that are being shipped, as well as the total cost of the books being shipped. This information is displayed to the user.

Alternate Scenarios:

- If Title is not in catalog in step 3, Application reports this fact to User, and terminates.
- In step 7, if the number of copies is less than or equal to 0, the Application terminates.
- In step 10, if some (but not all) of the required books are in-stock, Application submits a shipping order for just the copies of the book that are in-stock. This order will update the inventory.
- In step 10, if not all the required copies are in stock, Application submits a back order (to the Inventory Manager) for the number of additional copies that are needed. This backorder will update the inventory.

Oplossing


Specify Title
Report title not in catalog
Query Inventory
Report in-strock data (quantity and price)
Get number of copies to order
[ordered <= 0]
Report termination message
Get name and address
[No books in strock]
[All books in stock]
Submit backorder for all books
Submit shipping order and backorder
Submit shipping order for all books
performInventoryUpdate
Generate and disply book order

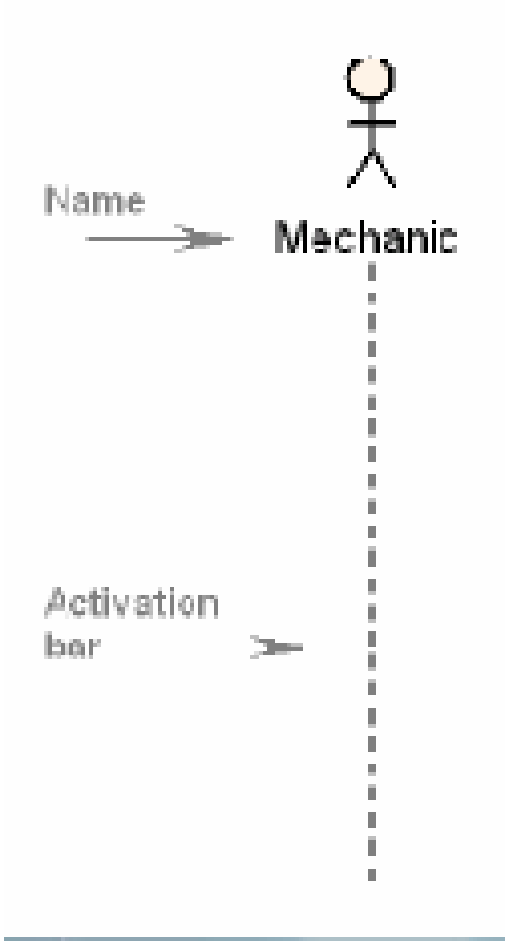
Sequentiediagram

- Klassediagram -> statische structuur van programma
  -> welke methoden de klassen hebben

- Sequetiediagram :
  -> toont wanneer de methoden gebruikt worden of samenhangen
  -> toont berichtenverkeer tussen een aantal objecten voor een bepaald gebruik van het systeem
  = = >2 assen
  * horizontaal: links -> rechts (objecten)
  * vertikaal: boven -> onder (tijdas)

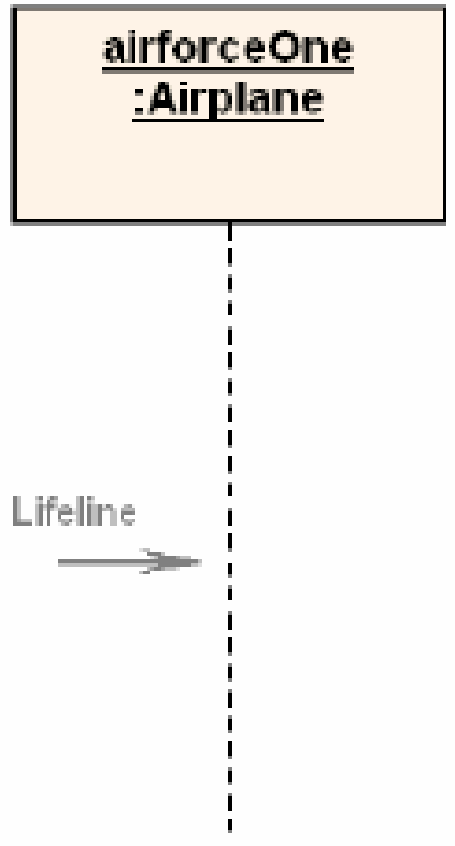
Actor

- Meestal het object welke een berichten stroom start



Object

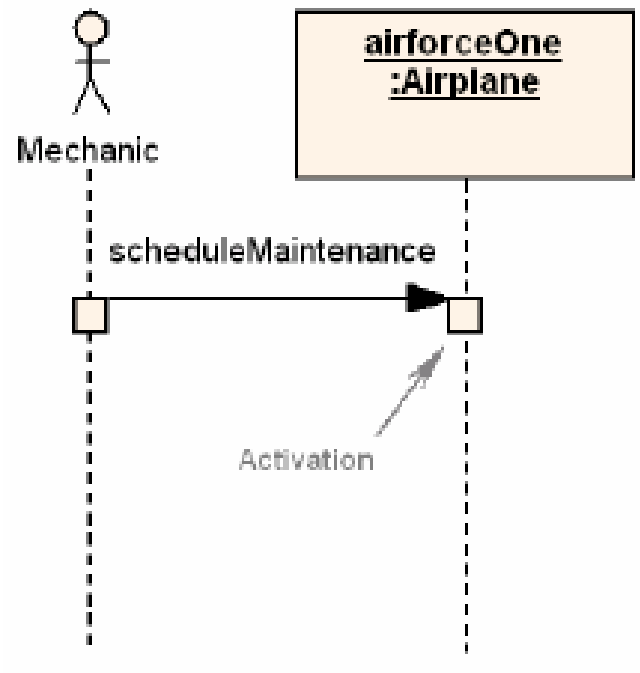
- Representeert een instantie van een klasse
  - Een object kan berichten ontvangen en/of versturen
- De Instance naam is onderlijnd
- Lifeline geeft levensduur van een object weer.



Message

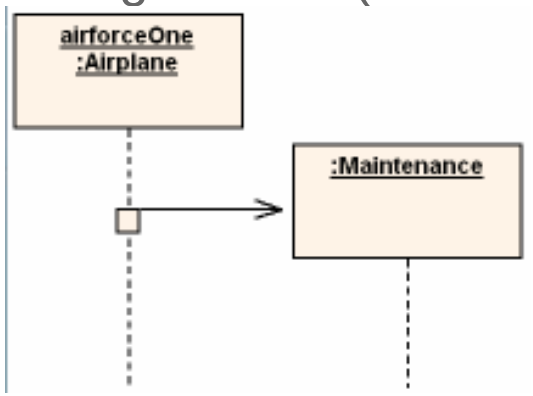
- Geeft een “method call” weer
  - Gezien vanuit ontvangende object
  - Argumenten en return type kunnen afgebeeld worden

- De activation laat de “duur” van de operatie zien

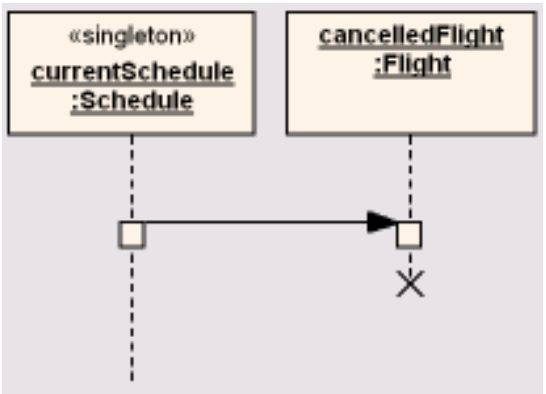


Object Creation & Deletion

- Pijl wijst naar het object zelf
- Object dat wordt gecreerd wordt aan pjileinde getekend (niet bovenaan de swinlane)

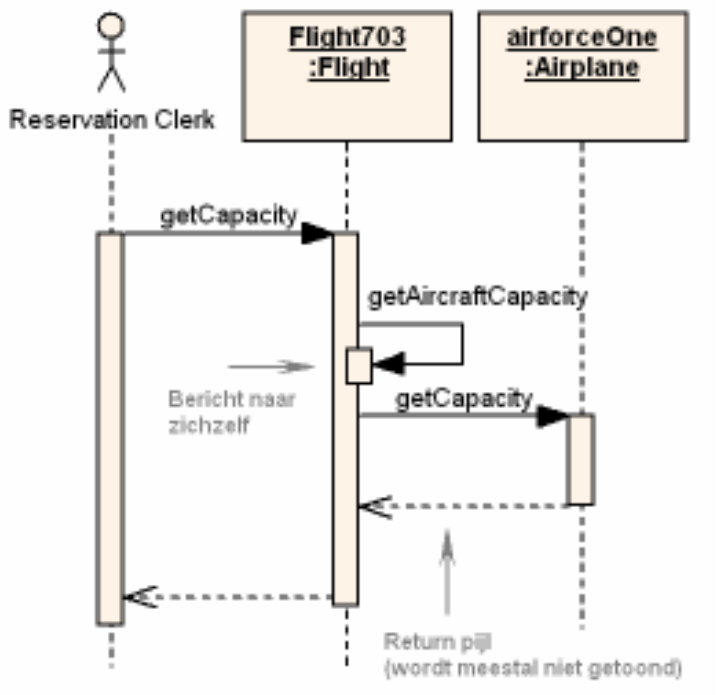


- Object deletion wordt aangegeven door einde van de lifeline



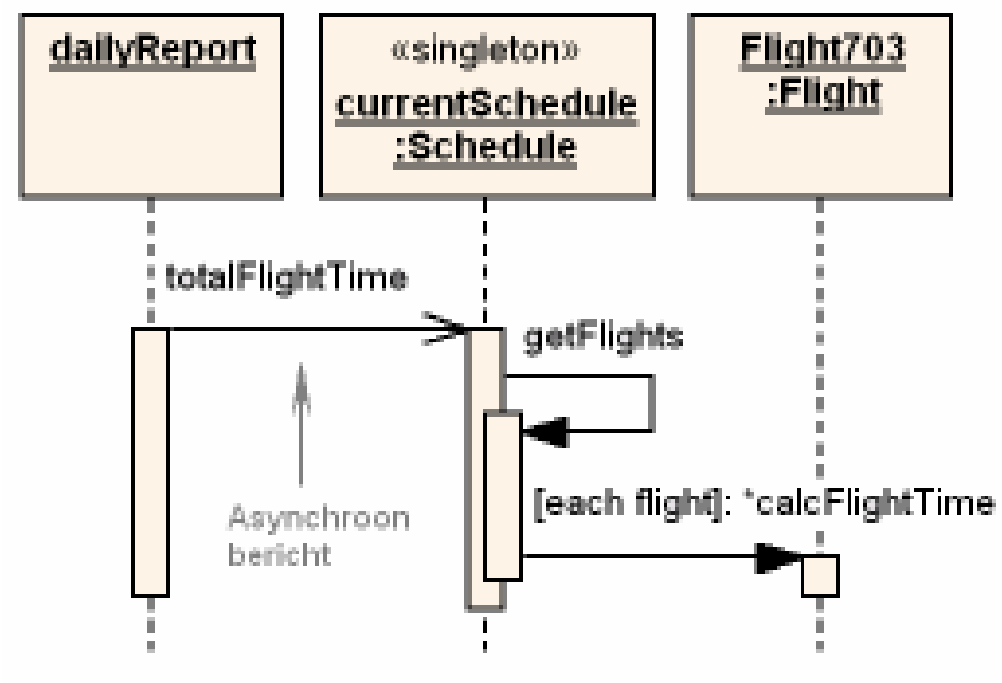
Message to self

- Een object kan een bericht naar zichzelf versturen
- Tonen als een bijkomende activatiebox



Asynchroon bericht

- Aangegeven met een open pijl
- De tijdsorde van asynchrone berichten is irrelevant



Sequetiediagram: voorbeeld