

HOOFDSTUK 3: DOMAIN
MODELS EN CLASS
DIAGRAMS

SOFTWARE DESIGN ESSENTIALS



OVERZICHT

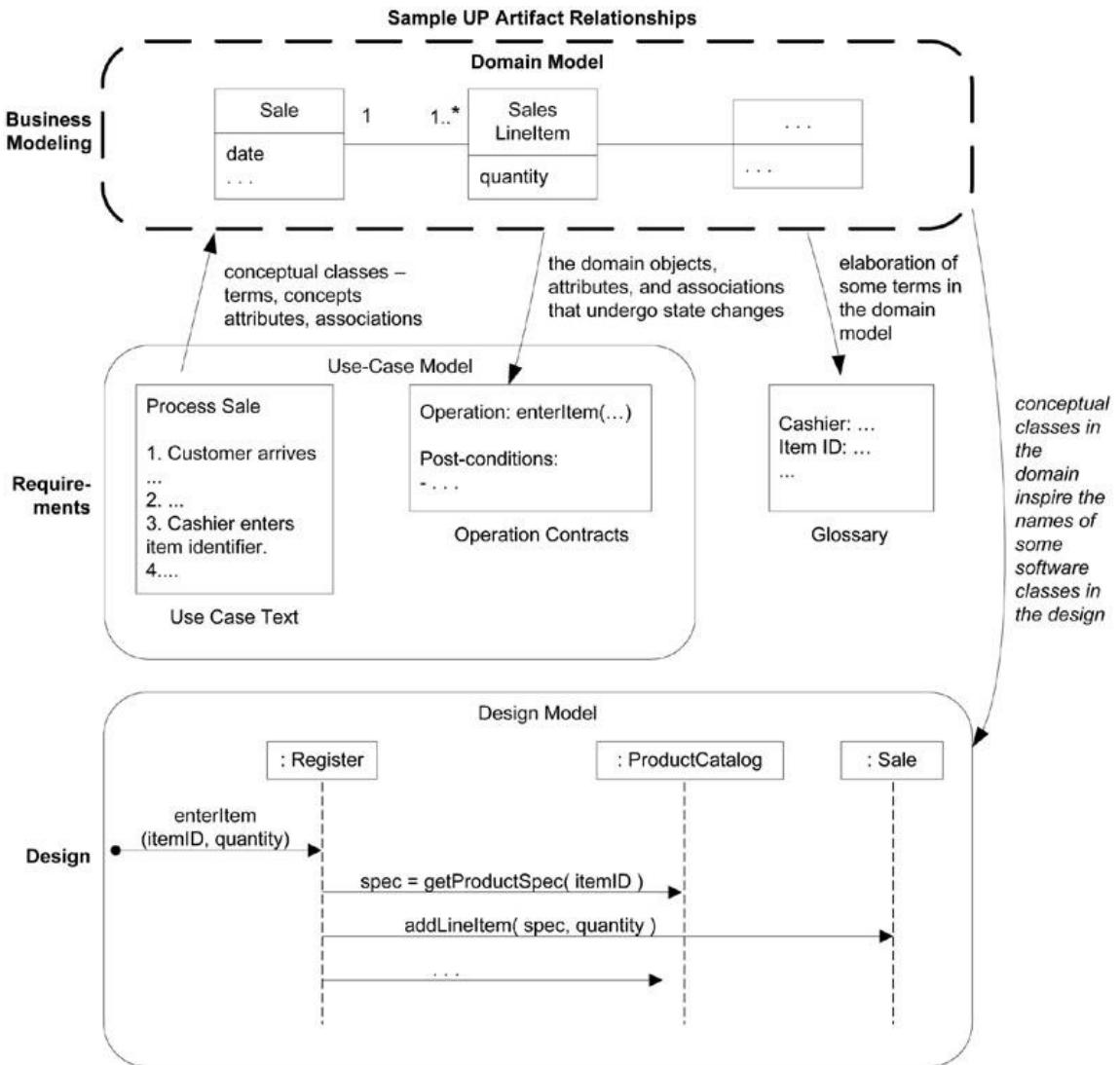
Domain models

Class diagrams

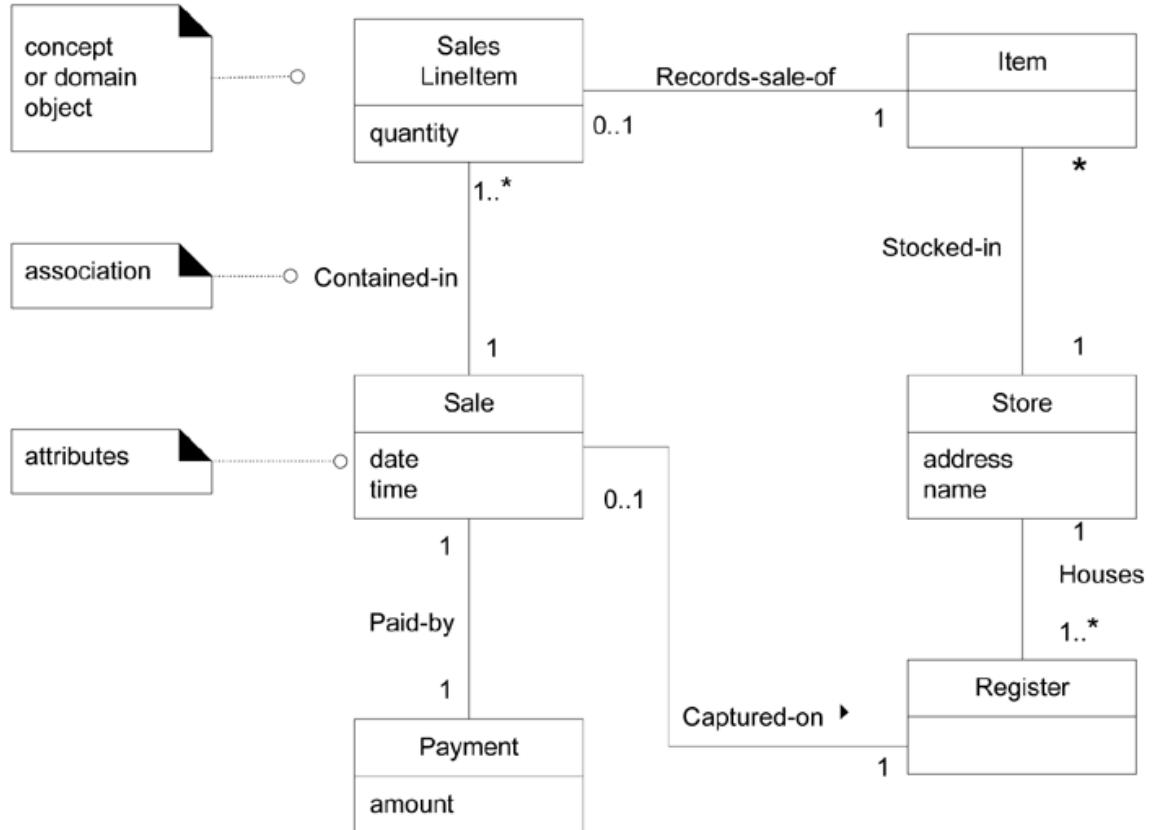
The background of the slide features a dark, abstract digital landscape composed of numerous small, glowing green and blue particles. These particles are arranged in a way that creates a sense of depth and motion, resembling rolling hills or waves. The overall effect is futuristic and dynamic.

DOMAIN MODELS

DOMAIN MODELS



DOMAIN MODELS



DOMAIN MODEL

- Een domeinmodel geeft een **visuele en conceptuele weergave** van de **belangrijke entiteiten** en hun **relaties** binnen een specifiek domein.
 - Het model **vertegenwoordigt real-world concepten** die relevant zijn voor het domein en die in software moeten worden gemodelleerd.
 - **Gebruik van domeinspecifieke taal** zorgt ervoor dat het model begrijpelijk is voor **niet-technische belanghebbenden**.
 - Het **helpt bij het analyseren en oplossen van problemen** binnen het domein.

DOMAIN MODEL

- **Belangrijk:**
 - Geen technische implementatie → Een domeinmodel beschrijft **concepten**, geen databases of softwarecomponenten.
 - Duidelijke communicatie → Doordat het model de natuurlijke taal van het domein gebruikt, kan het eenvoudig worden begrepen door zowel ontwikkelaars als business-experts.
- **Kort samengevat:**
 - Een domeinmodel helpt bij het structureren en verduidelijken van de **essentiële concepten en relaties binnen een softwaretoepassing**, zonder technische details!

CONCEPTUEEL INZICHT?

- Conceptueel inzicht betekent dat je een probleem of systeem **op een hoog niveau begrijpt**, zonder meteen in de technische details te duiken.
- Het gaat over het herkennen van de belangrijkste elementen en hun **onderlinge relaties**, zodat je het geheel beter kunt begrijpen en ontwerpen.

CONCEPTUEEL INZICHT

- In softwareontwikkeling gebruiken we conceptueel inzicht om **eerst na te denken over het probleem** voordat we het technisch uitwerken.
- **Voorbeeld:** Stel, je maakt een **bibliotheekbeheersysteem**. Voordat je programmeert, denk je na over:
 - Welke **objecten** en **concepten** belangrijk zijn (bijv. Boek, Lid, Lening).
 - Hoe deze objecten **met elkaar verbonden** zijn (bijv. Een lid kan meerdere boeken lenen).
 - Welke **informatie elk object** moet bevatten (bijv. Een boek heeft een titel en auteur).
- **Waarom is conceptueel inzicht belangrijk?**
 - Je **begrijpt het systeem** beter voordat je begint met programmeren.
 - Je maakt **betere ontwerpen** zonder fouten.
 - Je voorkomt dat je vastloopt in **technische details** te vroeg in het proces.

HOE MAAK JE EEN DOMEINMODEL?

IDENTIFICEREN VAN CONCEPTUELE KLASSEN

- Zoek de belangrijkste objecten en concepten
 - Bepaal welke objecten en concepten centraal staan binnen het domein.
 - Gebruik voorbeelden uit de praktijk om deze te herkennen.

- Gebruik de natuurlijke taal van het domein
 - Definieer klassen op basis van termen die in het domein gangbaar zijn.
 - Vermijd technische termen die niet direct bijdragen aan het begrip van het model.

VISUALISEER HET MODEL

- **Visualiseer het model in een UML-diagram**
 - Teken de klassen en geef ze een duidelijke naam.
 - Laat zien hoe de klassen zich tot elkaar verhouden.
 - Gebruik multipliciteiten om aan te geven hoeveel objecten met elkaar verbonden kunnen zijn.
 - Houd het model overzichtelijk en vermijd overbodige details.

- **Voeg associaties, multipliciteiten en attributen toe**
 - Definieer de relaties tussen de klassen en geef multipliciteiten aan (bijv. 1.., 0..1,).
 - Identificeer de belangrijkste kenmerken (attributen) van elke klasse.
 - Vermijd technische implementatiedetails zoals databasesstructuren.

MULTIPLICITEIT IN UML?

- Multipliciteit in UML geeft aan **hoeveel objecten** van een klasse gekoppeld kunnen zijn aan objecten van een andere klasse binnen een relatie.

Notatie	Betekenis
1	Precies één object
0..1	Optioneel (nul of één object)
*	Onbeperkt aantal (nul of meer objecten)
1..*	Minstens één object
2..5	Tussen een specifiek aantal objecten (bijv. tussen 2 en 5)

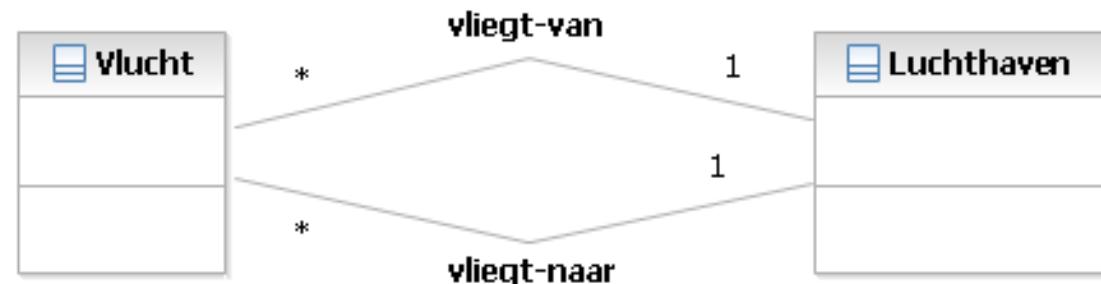
MULTPLICITEIT VS KARDINALITEIT

- Multipliciteit wordt gebruikt in UML, terwijl kardinaliteit een databaseconcept is. Ze beschrijven hetzelfde idee, maar in een andere context!

Aspect	Multipliciteit (UML)	Kardinaliteit (Database)
Waar gebruikt?	UML-klassendiagrammen	Relationale databases
Weergave	1, 0..1, *, 1..*	1:1, 1:N, M:N
Doel	Aangeven hoeveel objecten met elkaar verbonden kunnen zijn	Aangeven hoeveel rijen in tabellen gekoppeld zijn

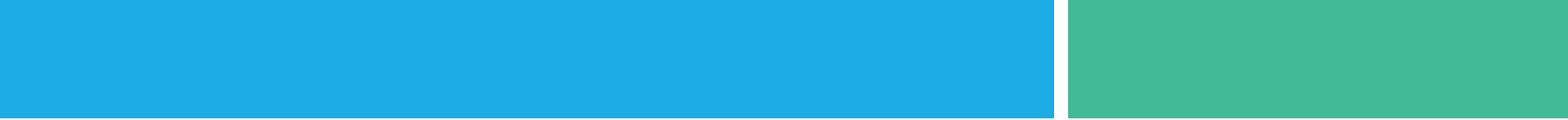
MULTIPLICITEITEN

- Multipliciteiten van associaties
 - 1 exact 1
 - 1..5 minstens 1, maximaal 5
 - 1..10 minstens 1, maximaal 10
 - 0..5 maximaal 5
 - 3,5,8 exact 3, 5 of 8
 - 0..* of *0 tot eender hoeveel
 - 1..* 1 tot eender hoeveel
- Multiple associaties





VOORBEELD



ZOEK DE BELANGRIJKSTE OBJECTEN EN CONCEPTEN

- *Bepaal welke objecten en concepten centraal staan binnen het domein.*

- **Voorbeeld:**
Bij een **bibliotheekbeheersysteem** kunnen de belangrijkste objecten en concepten zijn:
 - *Boek*
 - *Lid*
 - *Bibliothecaris*
 - *Lening*



GEBRUIK DE NATUURLIJKE TAAL VAN HET DOMEIN

- *Definieer klassen op basis van termen die in het domein gangbaar zijn.*

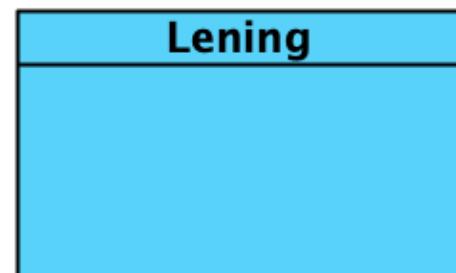
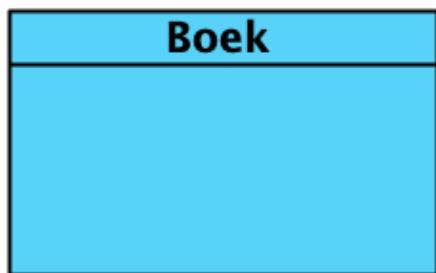
- **Voorbeeld:**
 - In een bibliotheek heet een boek gewoon "Boek", niet "Item" of "DataObject".
 - Een persoon die boeken leent, heet "Lid", niet "Gebruiker".
 - Een transactie waarbij een lid een boek leent, noemen we "Lening".

- **Fout:** Technische namen zoals Tbl_Book of Entity123 maken het model minder begrijpelijk.

- *Teken de klassen en geef ze een duidelijke naam.*

VISUALISEER HET MODEL IN EEN UML- DIAGRAM

- Voorbeeld UML-klassenmodel:
 - Boek
 - Lid
 - Lening



VOEG ASSOCIATIES EN ATTRIBUTEN TOE

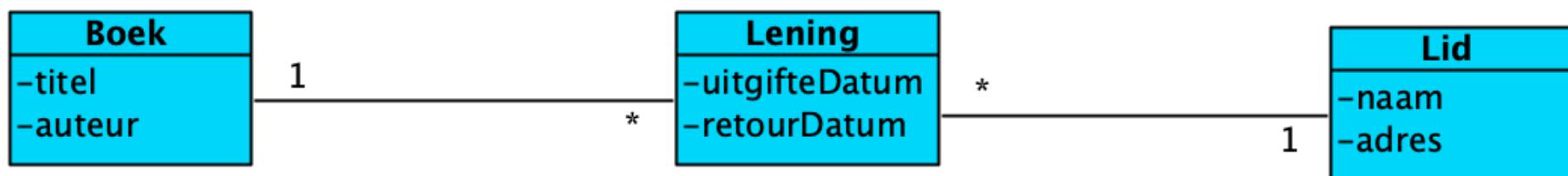
- Definieer *relaties* en belangrijke kenmerken (*attributen*) van elke klasse.
- Voorbeeld:
 - Een **Lid** kan meerdere **Boeken** lenen → Relatie: "Lid heeft meerdere leningen".
 - Een **Boek** heeft eigenschappen zoals *titel* en *auteur*.
 - Een **Lening** heeft een *uitgifteDatum* en een *retourDatum*.
- Zonder technische details zoals databases!
- Fout: datatypes of implementatie-details zoals Integer of List horen hier niet thuis.



* Visual Paradigm voegt automatisch het “-” teken toe. In principe hoort visibiliteit echter niet thuis in een domain model.

MULTIPLICITEITEN

- Een lid (1) kan meerdere leningen (*) hebben.
- Een lening (*) hoort bij precies één lid (1).
- Een lening (*) is gekoppeld aan precies één boekexemplaar (1).
- Een boekexemplaar (1) kan in de tijd meerdere leningen (*) hebben (maar niet tegelijk).



* Visual Paradigm voegt automatisch het “-” teken toe. In principe hoort visibiliteit echter niet thuis in een domain model.



PRAKTISCHE TIPS VOOR HET IDENTIFICEREN VAN CONCEPTUELE KLASSEN



GEBRUIK DE "ZELFSTANDIGE NAAMWOORDEN"- METHODE

- **Wat?**
 - Analyseer beschrijvende teksten en zoek **zelfstandige naamwoorden**.
 - Sommige worden **klassen**, andere worden **attributen**.
- **Voorbeeld:**

"Een klant plaatst een bestelling en betaalt via een betaalsysteem."

 - **Mogelijke klassen:** *Klant, Bestelling, Betaalsysteem*
 - **Mogelijke attributen:** *Bestelnummer, Betalingsstatus*

HERGEBRUIK BESTAANDE MODELLEN

- In veel vakgebieden bestaan **standaardmodellen** die als basis kunnen dienen bij het ontwerpen van een systeem. Deze modellen bevatten veelvoorkomende concepten en structuren die in eerdere projecten zijn bewezen.
- Hoe kun je standaardmodellen gebruiken?
 - Raadpleeg bronnen zoals boeken, referentiemodellen of eerdere projecten binnen het domein.
 - Gebruik bestaande **frameworks** zoals ERD-modellen (Entity-Relationship Diagrammen), UML Library Patterns of industry best practices.
- Voorbeeld:
 - Een **CRM-systeem** zal altijd *Klant*, *Contactgegevens*, en *Transacties* bevatten.
 - Een **E-commerceplatform** heeft bijna altijd *Product*, *Winkelmandje* en *Bestelling*.

DENK IN CATEGORIEËN

- **Wat?**
- Gebruik bestaande categorieën om klassen te identificeren.

Categorie	Voorbeeldklassen
Personen	Klant, Werknemer, Student
Locaties	Filiaal, Magazijn, Kamer
Dingen	Product, Boek, Voertuig
Organisaties	Bedrijf, Afdeling, School
Gebeurtenissen	Bestelling, Factuur, Betaling
Toestellen/Systemen	Betaalsysteem, Scanner, Printer

VERMIJD OVERBODIGE KLASSEN

- **Wat?**
 - Niet elk concept verdient een eigen klasse.
 - Vermijd **technische details en te gedetailleerde modellen**.
- **Slechte voorbeelden:**

"Prijs" als een aparte klasse → Dit is een attribuut van *Product*.
"Login" als een klasse → Authenticatie is een proces, geen entiteit in het domein.
- **Goede voorbeelden:**

"Factuur" als een klasse → Het is een belangrijk document met unieke eigenschappen.
"Leverancier" als een klasse → Dit kan meerdere attributen en relaties bevatten.

WANNEER IS IETS EEN KLASSE EN NIET EEN ATTRIBUUT?

- Bij het modelleren van een systeem is het belangrijk om te bepalen of een bepaald concept als **klasse** of als **attribuut** moet worden weergegeven. Een eenvoudige vuistregel is:
- Als een concept in de echte wereld meer is dan alleen een getal of tekst, dan is het meestal een klasse en geen attribuut.

Vraag	Attribuut	Klasse
Is het een eenvoudige eigenschap van een object?	Ja	Nee
Heeft het zelf kenmerken (attributen)?	Nee	Ja
Heeft het relaties met andere objecten?	Nee	Ja
Kan het als een op zichzelfstaand concept worden gezien?	Nee	Ja



VOORBEELD: BESTELLING IN EEN WEBSHOP

- **Attribuut:** Een bestelling heeft een **bestelnummer** en **besteldatum** → eenvoudige eigenschappen van de bestelling.
- **Klasse:** Een bestelling bevat meerdere producten en een klant → het is een complex concept dat een **eigen klasse** verdient.

ATTRIBUTEN & WANNEER EEN APART DATATYPE DEFINIËREN

- Soms is een attribuut **complex genoeg** om als een **eigen datatype** te worden gedefinieerd, in plaats van een eenvoudige String of Integer.
- **Wanneer een apart datatype gebruiken?**
 - Samengesteld uit meerdere delen
 - Voorbeeld: *Telefoonnummer* (*landcode + nummer*)
 - Voorbeeld: *Naam* (*voornaam + achternaam*)
 - Bevat specifieke operaties zoals parsing of validatie
 - Voorbeeld: *Rijksregisternummer* (*moet gevalideerd worden*)
 - Bestaat uit meerdere onderliggende attributen
 - Voorbeeld: *Promotieprijs* (*heeft een start- en einddatum*)
 - **Het is een kwantiteit met een eenheid**
 - Voorbeeld: *Prijs* (*bedrag + valuta, bv. Money-class*)

VOORBEELD: ADRES

- **Als attribuut:** Een adres kan worden opgeslagen als een simpele tekstregel: "Stationsstraat 10, 1000 Brussel".
- **Als klasse:** Als je aparte gegevens zoals **straatnaam**, **huisnummer**, **postcode** en **stad** nodig hebt, is het beter om een **Adres**-klasse te maken.

WANNEER LEG JE EEN ASSOCIATIE TUSSEN KLASSEN?

- Als de relatie belangrijk is voor het domein en onthouden moet worden.
- Voor essentiële verbindingen zoals verkochte goederen gekoppeld aan een verkoop.

WANNEER VERMIJD JE ASSOCIATIES?

- Voorkom overbelasting van het model → Beperk je tot de noodzakelijke relaties.
- Niet elke theoretische link modelleren → Een model met 20 klassen kan 190 mogelijke relaties hebben!
- Domeinmodel ≠ implementatie → Het beschrijft de probleemwereld, niet de technische structuur (zoals een class diagram).
- Tip: Focus op de **essentiële verbanden** die bijdragen aan het begrip van het domein!

NAAMGEVING VAN ASSOCIATIES

- Gebruik duidelijke en specifieke namen voor associaties.
- **Volg de structuur:** Klasse1 – werkwoord – Klasse2 voor een logische en leesbare naam.
- Goede voorbeelden (duidelijk en actief)
 - "Verkoop wordt betaald door Cashbetaling"
 - "Speler staat op Positie"
- Slechte voorbeelden (vaag of passief)
 - "Verkoop gebruikt Cashbetaling" (te algemeen, zegt niets over de relatie)
 - "Speler heeft Positie" (te breed, geen duidelijke betekenis)
- Tip: Gebruik **actieve en betekenisvolle werkwoorden** om relaties precies te beschrijven!



DESCRIPTION CLASS



DESCRIPTION CLASS

- Een **Description Class** in een domeinmodel wordt gebruikt om **eigenschappen en beschrijvende informatie** over een ander object op te slaan. Dit voorkomt **onnodige duplicatie** van gegevens en zorgt voor flexibiliteit.
- **Voorbeeld:**
- In een **bibliotheekbeheersysteem**:
 - De klasse **BoekExemplaar** stelt een fysiek boek voor (met een unieke barcode).
 - De klasse **BoekBeschrijving** bevat algemene informatie over het boek, zoals *titel, auteur, genre*.
- **Belangrijk:**
 - Een **BoekBeschrijving** wordt gedeeld door meerdere **BoekExemplaren**.
 - Dit voorkomt dat je bij elk exemplaar opnieuw *titel en auteur* moet opslaan.

DESCRIPTION CLASS

voorbeeld

Bibliotheek met 3 exemplaren van elk boek
Een boek heeft een titel, auteur, ISBN, genre

Boek
Titel
Auteur
ISBN
genre
exemplaarnr

VS.

Boekexemplaar
Exemplaarnr

Boek
Titel
auteur
ISBN
genre



CONCLUSIE



WAT IS EEN DOMEINMODEL?

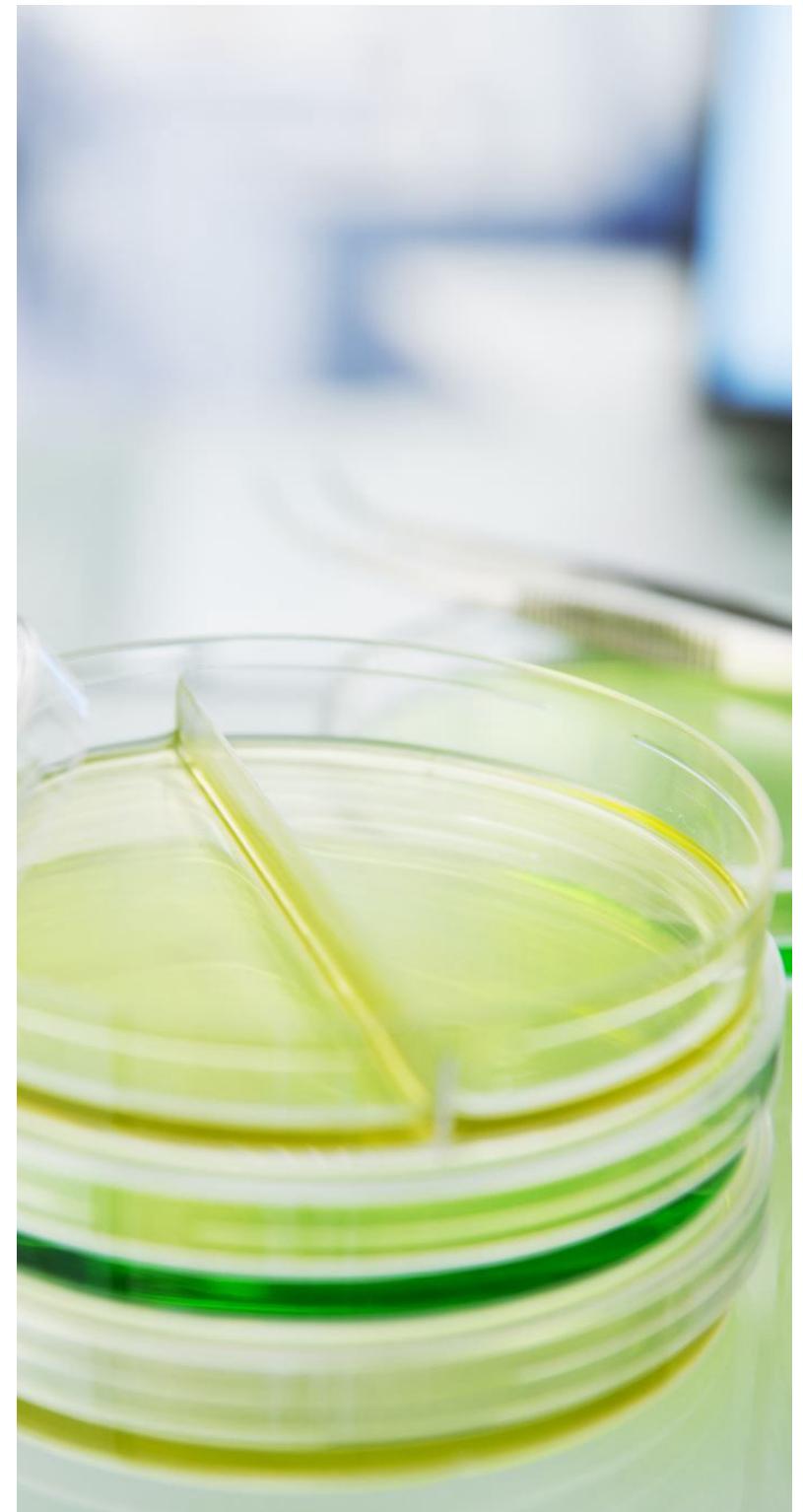
- Een visuele representatie van concepten (entiteiten) en hun relaties binnen een domein.
- Gebruikt de natuurlijke taal van het domein en is begrijpelijk voor zowel technische als niet-technische stakeholders.
- Beschrijft de probleemwereld, niet de technische implementatie.

BELANGRIJKE PRINCIPES

- **Identificeer de conceptuele klassen** → Gebruik tekstuele analyse, bestaande modellen en categorieën.
- **Definieer associaties** → Enkel als ze noodzakelijk zijn om het domein correct te beschrijven.
- **Gebruik duidelijke attribuutnamen** → Voeg alleen toe wat relevant is voor het domein.
- **Maak aparte datatypes indien nodig** → Bij samengestelde attributen of gegevens met validatieregels.

WAAROM IS EEN GOED DOMEINMODEL BELANGRIJK?

- Helpet bij analyse en ontwerp van een softwareoplossing.
- Ondersteunt communicatie tussen business en ontwikkelaars.
- Dient als basis voor verdere modellering (zoals class diagrams en databases).
- **Conclusie:** Een goed domeinmodel legt de **fundamenten** voor een correct ontworpen softwaresysteem en voorkomt misverstanden tijdens de ontwikkeling!



CLASS DIAGRAMS



INLEIDING TOT CLASS DIAGRAMS

- Een Class Diagram is een structureel UML-diagram dat de klassen in een systeem visualiseert, inclusief hun attributen, methoden en onderlinge relaties. Het wordt gebruikt in objectgeoriënteerd ontwerp om de structuur van een applicatie te definiëren en vormt een blauwdruk voor de softwarearchitectuur.

WAAROM EEN CLASS DIAGRAM?

- Geeft een overzicht van de structuur van het systeem.
- Toont de relatie tussen klassen, zoals associaties, aggregaties en composities.
- Maakt duidelijk hoe data wordt georganiseerd en hoe objecten interageren.
- Wordt gebruikt voor analyse, ontwerp en documentatie van software.

WAT BEVAT EEN CLASS DIAGRAM?

- **Klassen** (*naam van de klasse, attributen, methoden*).
- **Relaties** (*associatie, afhankelijkheid, overerving, aggregatie, compositie*).
- **Toegankelijkheidsmodificatoren** (*public, private, protected*).

CLASS DIAGRAM VS DOMAIN MODEL

- Een Domeinmodel helpt bij het begrijpen van het probleem en de concepten.
- Een Class Diagram is een technische uitwerking en wordt gebruikt voor implementatie.
- Eerst maak je een domeinmodel om het probleem te begrijpen. Daarna werk je dit uit in een class diagram voor de technische implementatie!

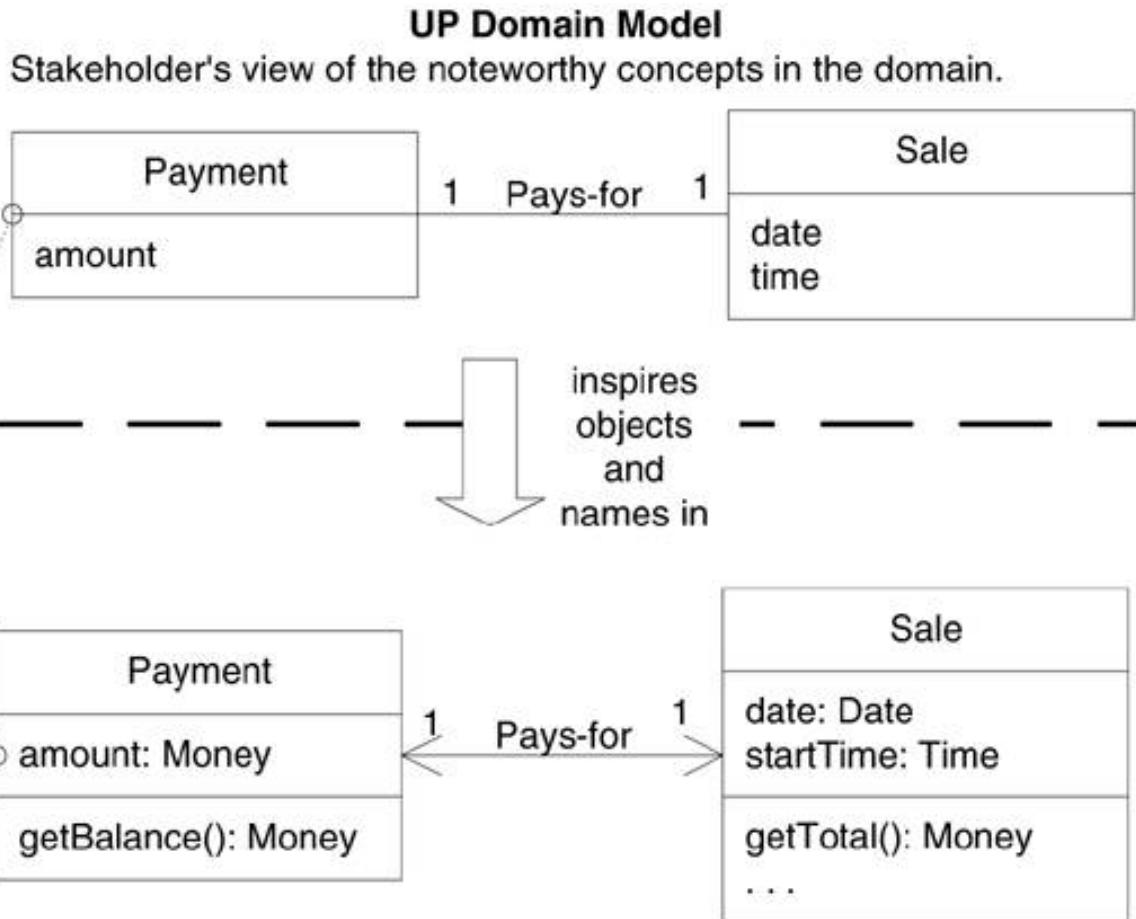
VERSCHIL TUSSEN EEN CLASS DIAGRAM EN EEN DOMEINMODEL

Aspect	Class Diagram	Domeinmodel
Doel	Technisch ontwerp van een systeem	Conceptuele weergave van het domein
Focus	Klassen, attributen, methoden, technische structuren	Concepten en hun relaties binnen het domein
Details	Bevat methoden en datatypes	Alleen concepten en hun eigenschappen
Technische details	Ja, bevat implementatiegerichte elementen	Nee, abstract en onafhankelijk van technologie
Gebruik	Ontwikkeling en implementatie van software	Analyseren en begrijpen van het domein
Relaties	Technische associaties zoals aggregatie, compositie en afhankelijkheden	Eenvoudige associaties zonder technische details

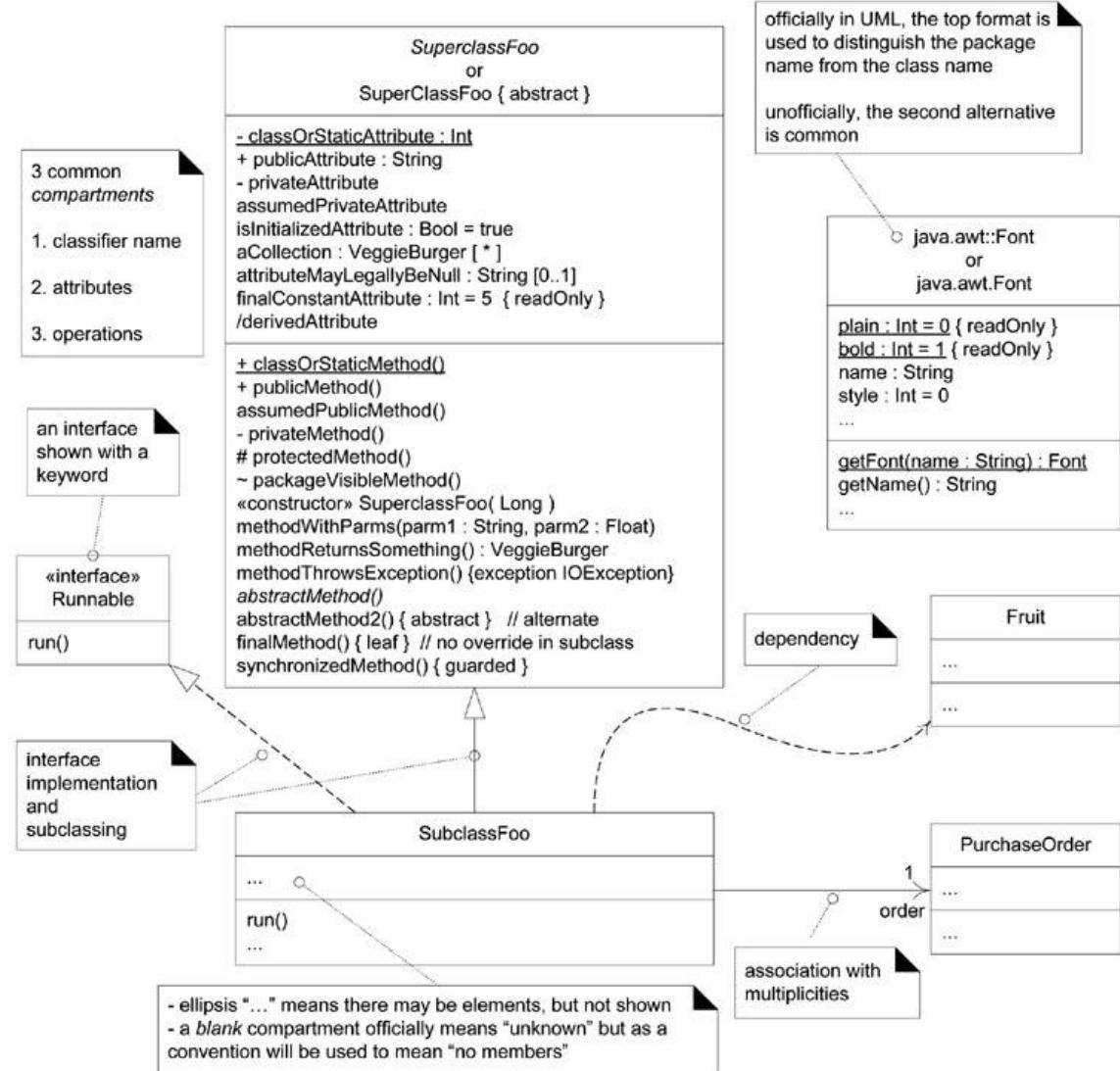
A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former *inspired* the naming and definition of the latter.

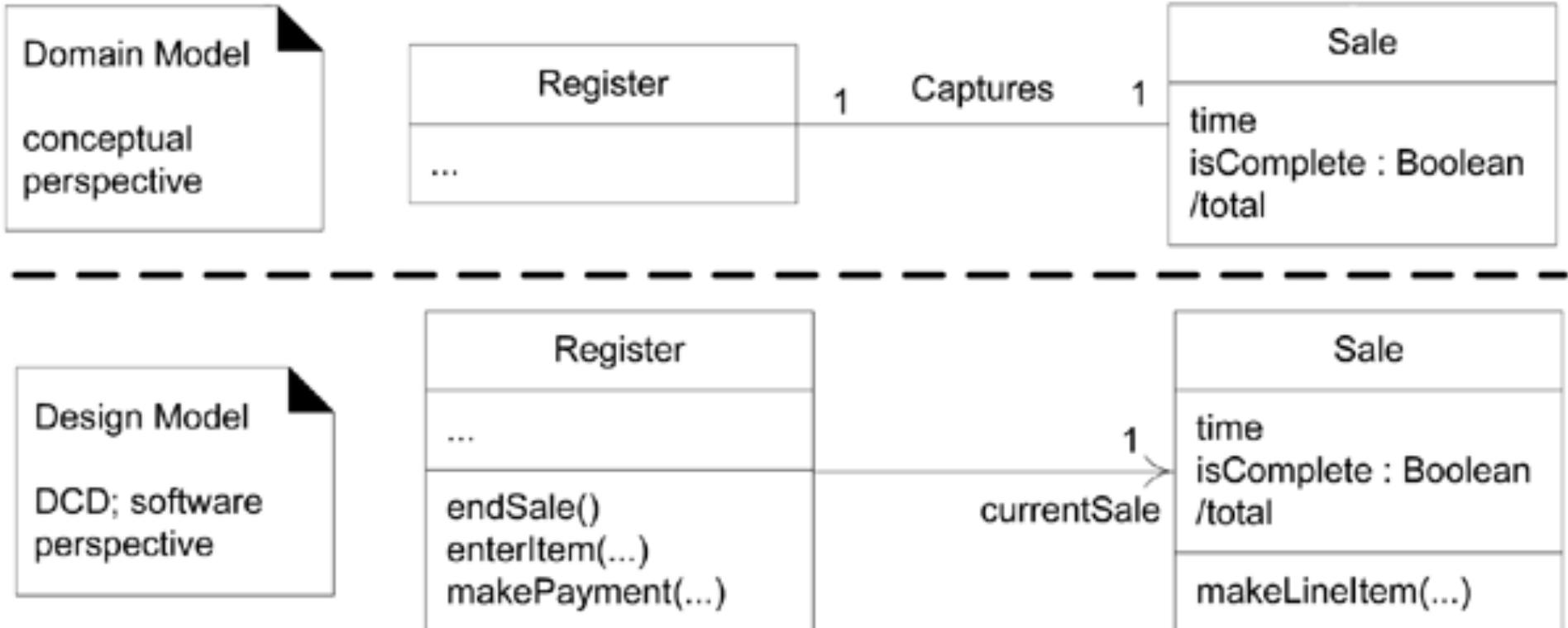
This reduces the representational gap.

This is one of the big ideas in object technology.



CLASS DIAGRAM





CLASS DIAGRAM

- Verschil domain model – class diagram



VAN EEN DOMEINMODEL NAAR EEN CLASS DIAGRAM



STAP 1: IDENTIFICEER DE CONCEPTEN ALS KLASSEN

- Elk concept in het domeinmodel wordt meestal een **klasse** in het class diagram.
- **Voorbeeld:** In een bibliotheekdomeinmodel heb je **Lid**, **Boek**, en **Lening**. Deze concepten worden *klassen in het class diagram*.

STAP 2: IDENTIFICEER ATTRIBUTEN PER KLASSE

- Attributen zijn de **eigenschappen** van de concepten.
- Gebruik het domeinmodel om te bepalen welke informatie bij elke klasse hoort.
- Voorbeeld:
 - Lid heeft een **naam** en **lidnummer**.
 - Boek heeft een **titel** en **auteur**.
 - Lening heeft een **uitgifteDatum** en **retourDatum**.
- Voeg deze attributen toe aan het *class diagram*.

STAP 3: BEPAAL DE RELATIES EN MULTIPLICITEITEN

- Kijk naar de **associaties** in het domeinmodel en bepaal hoeveel objecten met elkaar verbonden kunnen zijn.
- Gebruik multipliciteiten zoals 1..*, 0..1, * om aan te geven hoeveel instanties van de ene klasse aan de andere zijn gekoppeld.

STAP 4: VOEG METHODEN TOE AAN KLASSEN

- Bepaal welke acties de klassen uitvoeren.
- Dit zijn meestal de **werkwoorden** in de domeinbeschrijving.
- Voorbeeld:
 - Een **Lid** kan een boek **lenen()**.
 - Een **Lening** kan een boek **terugbrengen()**.
- Voeg deze methoden toe in het class diagram.

STAP 5: VOEG TECHNISCHE DETAILS TOE

- Voeg **datatypes** toe aan attributen, zoals String, int, Date.
- Voeg **constructors, getters, en setters** toe als nodig.
- Toegangsmodifiers: geef de visibiliteit van attributen en methoden aan (public, private, protected)

VISIBILITEIT

	Symbool	Verklaring
+		Public
-		Private
#		Protected

SAMENVATTING

Identificeer klassen uit het domeinmodel.

Voeg attributen toe aan elke klasse.

Definieer relaties en multipliciteiten tussen klassen.

Voeg methoden toe op basis van acties in het domein.

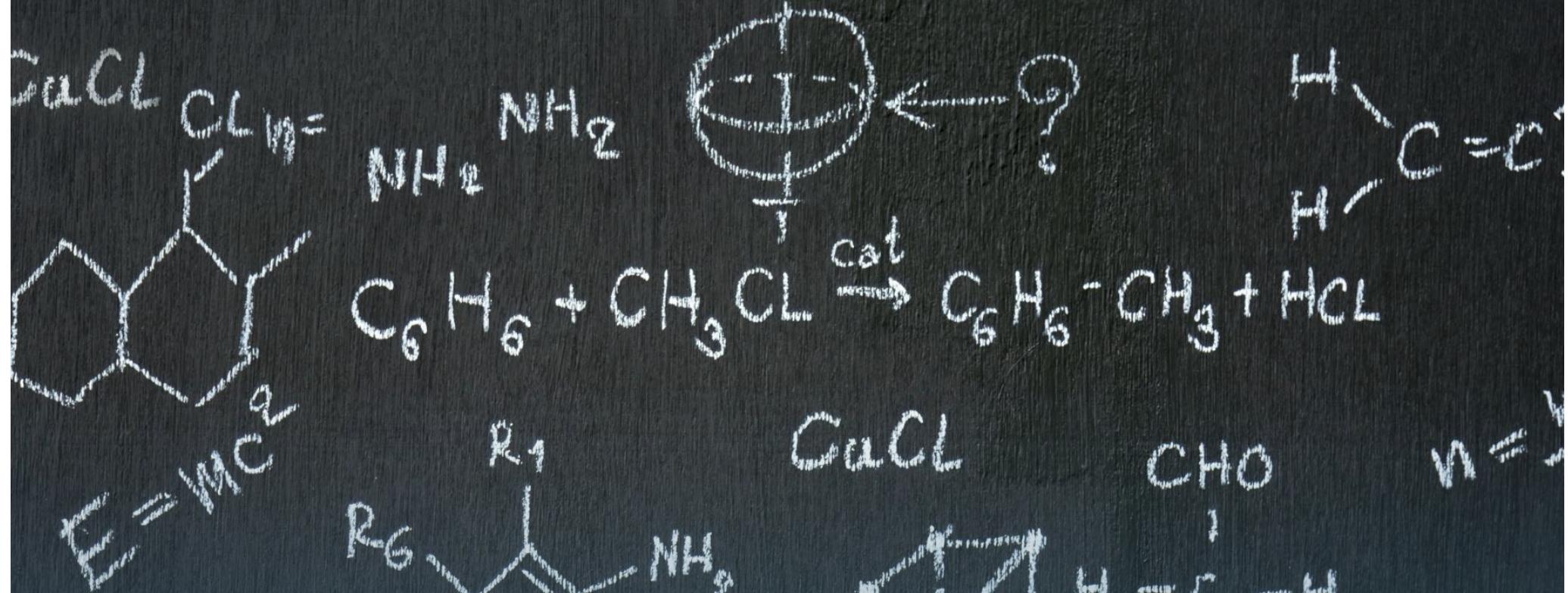
Voeg implementatiedetails toe.

VOORBEELD

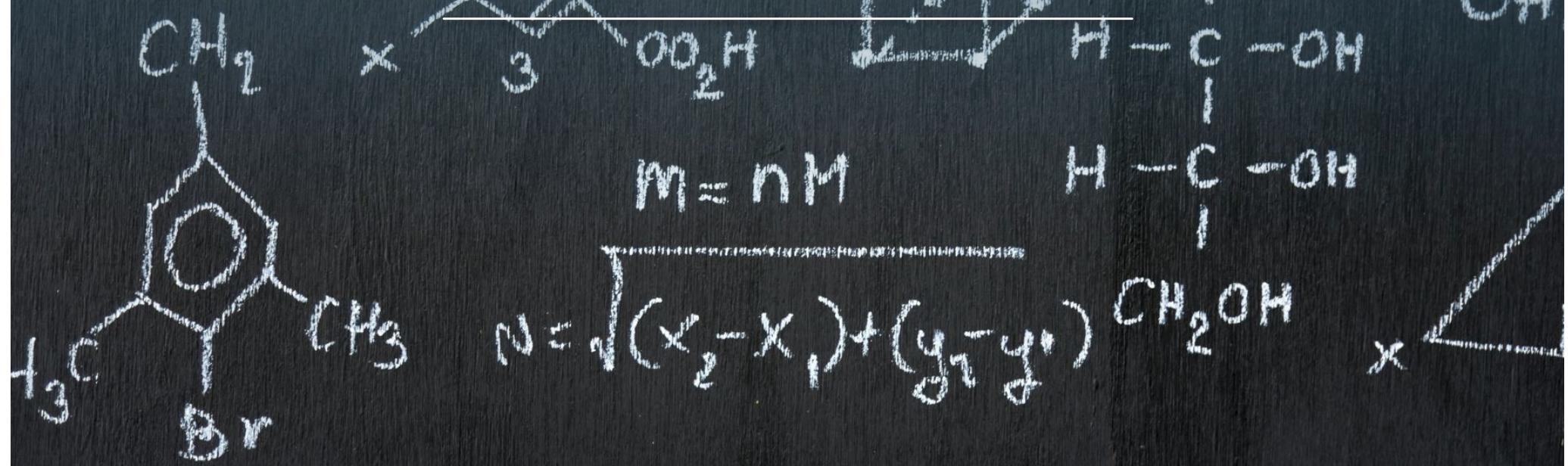
- UML operaties (methoden)

Bankrekening	
- nummer:	String
- saldo:	double = 0.0
+ openRekening():	void
+ stortBedrag(bedrag: double):	void
+ haalGeldAf(bedrag: double):	void

```
class Bankrekening{  
    private:  
        string nummer;  
        double saldo;  
    public:  
        void BankRekening::openRekening() {  
            //code  
        }  
        void BankRekening::stortBedrag(double bedrag) {  
            //code  
        }  
        void BankRekening::haalGeldAf(double bedrag) {  
            //code  
        }  
}
```



SOORTEN RELATIES IN EEN CLASS DIAGRAM



RELATIES TUSSEN KLASSEN

- We kennen de volgende relaties tussen Klassen:
 - eigenaars voeden huisdieren, huisdieren maken eigenaars blij (associatie)
 - een staart is een deel van zowel honden als katten (aggregatie / compositie)
 - een kat is een soort huisdier (overerving / generalisatie)

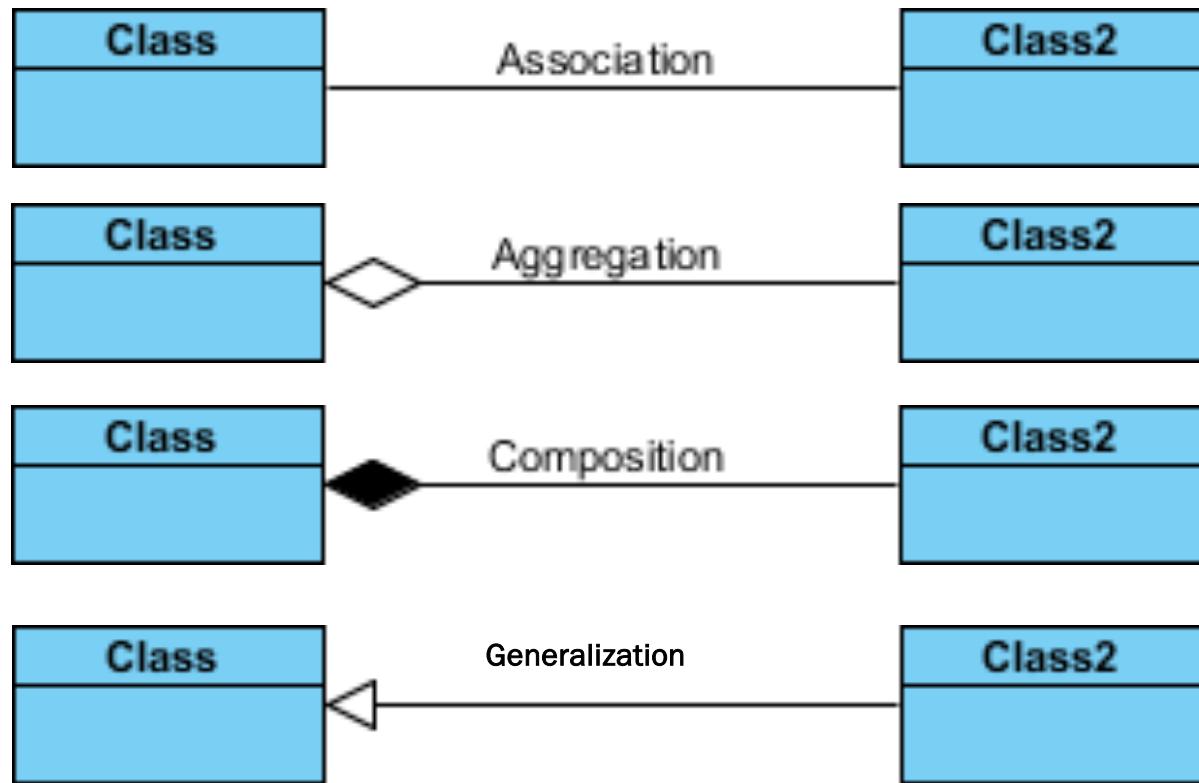
ASSOCIATIE VS AGGREGATIE VS COMPOSITIE

- **Associatie:** Dit is een algemene relatie tussen twee klassen waarbij objecten van de ene klasse kunnen communiceren met objecten van een andere klasse.
 - Voorbeeld: Een student en een docent kunnen met elkaar communiceren.
- **Aggregatie:** Dit is een specifieke vorm van associatie waarbij een 'heeft-een'-relatie bestaat, en het kindobject onafhankelijk van het ouderobject kan bestaan.
 - Voorbeeld: Een klas (ouder) en studenten (kind). Als de klas wordt verwijderd, blijven de studenten nog steeds bestaan.
- **Compositie:** Dit is een sterkere vorm van aggregatie waarbij het kindobject niet onafhankelijk van het ouderobject kan bestaan.
 - Voorbeeld: Een huis (ouder) en kamers (kind). Als het huis wordt verwijderd, kunnen de kamers niet meer bestaan.

OVERERVING/GENERALISATIE

- **Overerving/Generalisatie:** Dit is een "is-een"-relatie waarbij een subklasse eigenschappen en methoden overneemt van een superklasse. Hierdoor wordt hergebruik van code gestimuleerd en duplicatie voorkomen.
 - **Voorbeeld:** Een **Auto** en een **Vrachtwagen** zijn beide een type **Voertuig** en erven de algemene eigenschappen van de klasse **Voertuig**.

ASSOCIATIE VS AGGREGATIE VS COMPOSITIE VS GENERALISATIE

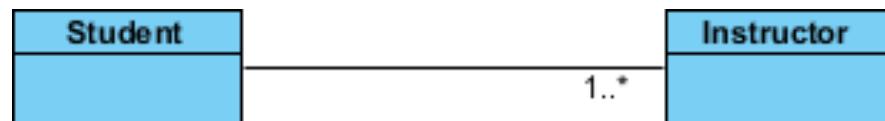


ASSOCIATIE

- In het geval dat twee klassen in een model met elkaar moeten communiceren, moet er een link tussen hen zijn, en dat kan worden weergegeven door een associatie (connector).
- We kunnen de multipliciteit van een associatie aangeven door multipliciteitsdecoraties toe te voegen aan de lijn die de associatie aangeeft.

ASSOCIATIE

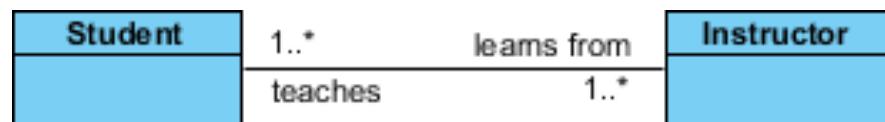
- Een enkele student kan zich associëren met meerdere docenten:



- Het voorbeeld geeft aan dat elke docent één of meer studenten heeft:



- We kunnen ook het gedrag van een object in een associatie aangeven (d.w.z. de rol van een object) met behulp van rolnamen.

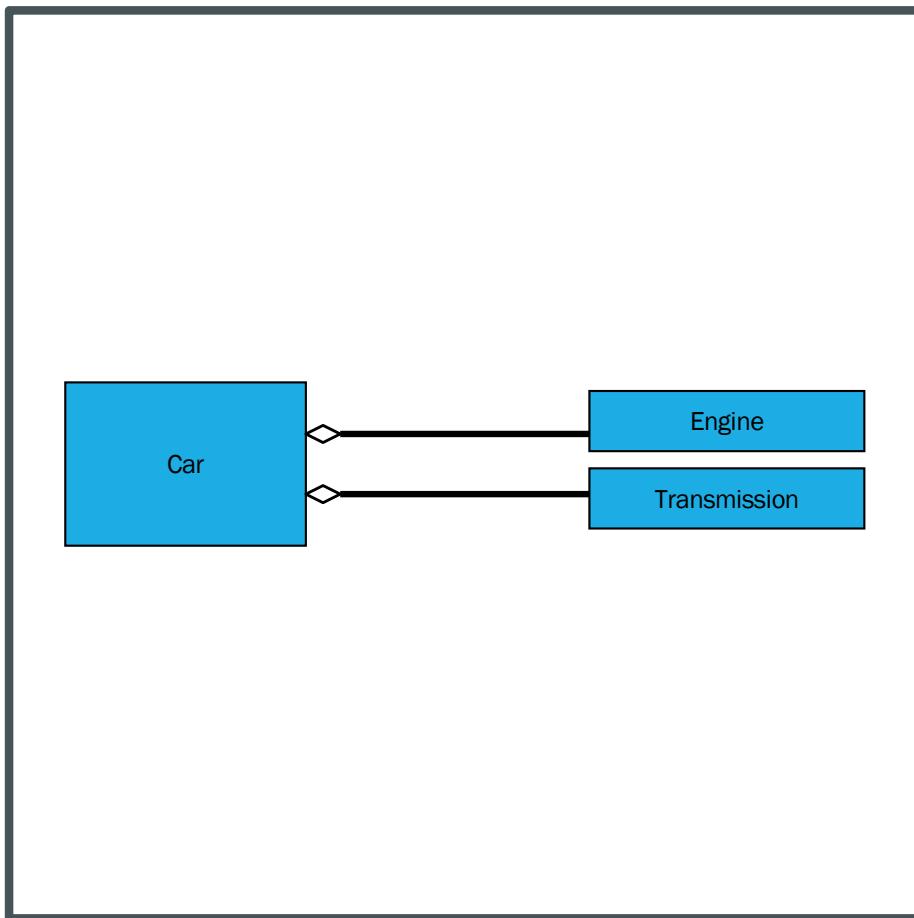


AGGREGATIE EN COMPOSITIE ZIJN SUBSETS VAN ASSOCIATIE

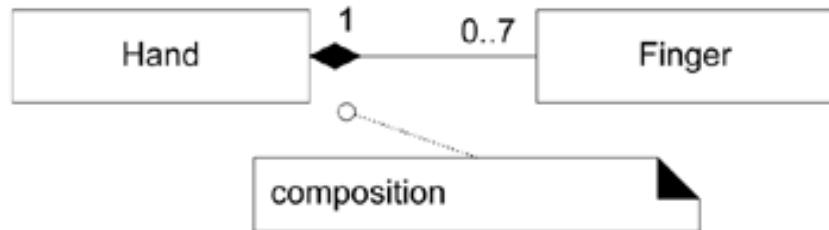
- Aggregatie en compositie zijn subsets van associatie, wat betekent dat ze specifieke gevallen van associatie zijn. In zowel aggregatie als compositie 'bezit' een object van de ene klasse een object van een andere klasse. Maar er is een subtiel verschil:
 - **Aggregatie** impliceert een relatie waarbij het kind onafhankelijk van de ouder kan bestaan.
 - Voorbeeld: Klas (ouder) en Student (kind). Verwijder de Klas en de Studenten bestaan nog steeds.
 - **Compositie** impliceert een relatie waarbij het kind niet onafhankelijk van de ouder kan bestaan.
 - Voorbeeld: Huis (ouder) en Kamer (kind). Kamers bestaan niet los van een Huis."



UML AGGREGATION RELATIE



- Een whole-part relatie tussen aggregaat (whole) en één van zijn onderdelen
- Het onderdeel kan op zich bestaan (levensduur part kan dus langer zijn dan levensduur whole)



composition means
 -a part instance (*Square*) can only be part of one composite (*Board*) at a time

-the composite has sole responsibility for management of its parts, especially creation and deletion

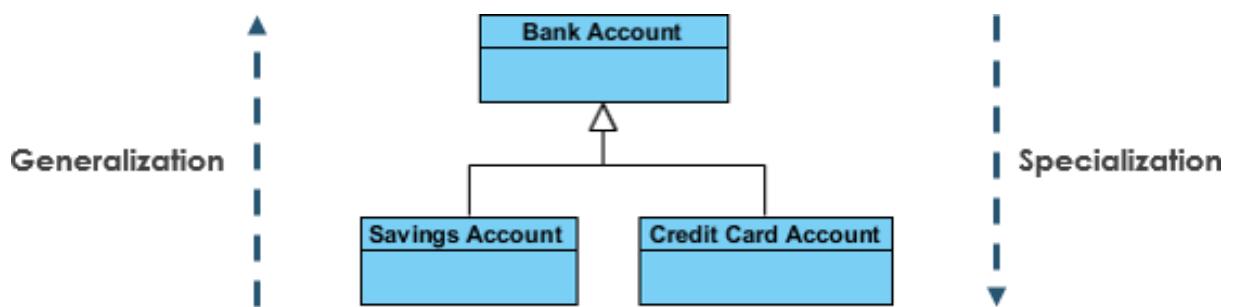


UML COMPOSITION RELATIE

- Whole-part relatie (geheel-deel)
- 3 voorwaarden
 - Instantie van deel behoort tot juist 1 geheel op gegeven tijdstip
 - Deel kan niet op zichzelf bestaan
 - Het geheel staat in voor het aanmaken van de delen

GENERALISATIE VS SPECIALISATIE

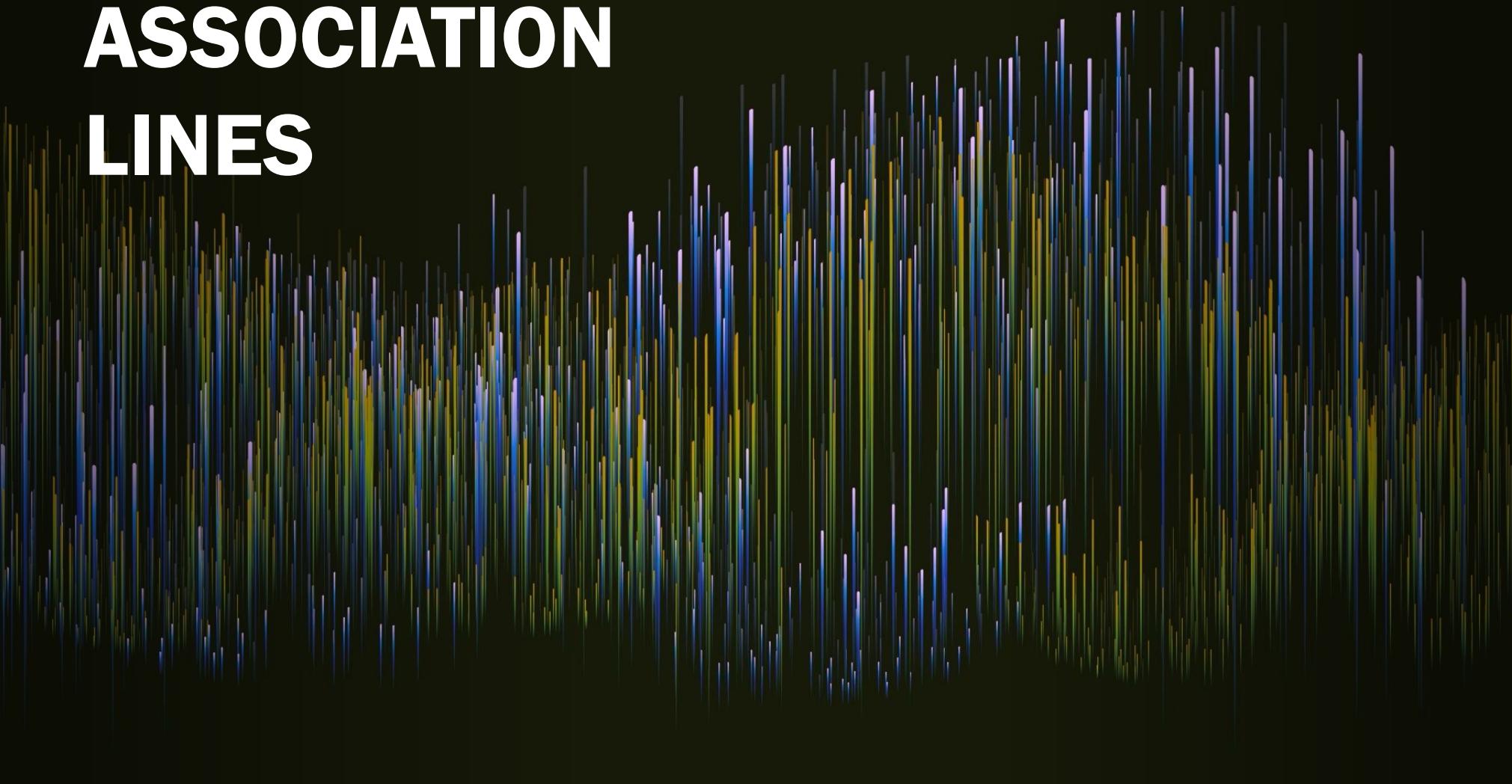
- Generalisatie is een mechanisme om vergelijkbare klassen van objecten te combineren in een enkele, meer algemene klasse. Generalisatie identificeert overeenkomsten tussen een reeks entiteiten. De overeenkomst kan betrekking hebben op attributen, gedrag of beide. Met andere woorden, een superklasse heeft de meest algemene attributen, operaties en relaties die kunnen worden gedeeld met subklassen. Een subklasse kan meer gespecialiseerde attributen en operaties hebben.
- Specialisatie is het omgekeerde proces van Generalisatie, wat betekent dat er nieuwe subklassen worden gecreëerd uit een bestaande klasse.



GENERALISATIE VS OVERERVING

- Generalisatie is de term die we gebruiken om de abstractie van gemeenschappelijke eigenschappen in een basisklasse aan te duiden in UML.
- De Generalisatie-associatie in een UML-diagram staat ook bekend als Overerving.
- Wanneer we Generalisatie implementeren in een programmeertaal, wordt dit vaak Overerving genoemd. Generalisatie en overerving zijn hetzelfde. De terminologie verschilt alleen afhankelijk van de context waarin het wordt gebruikt.

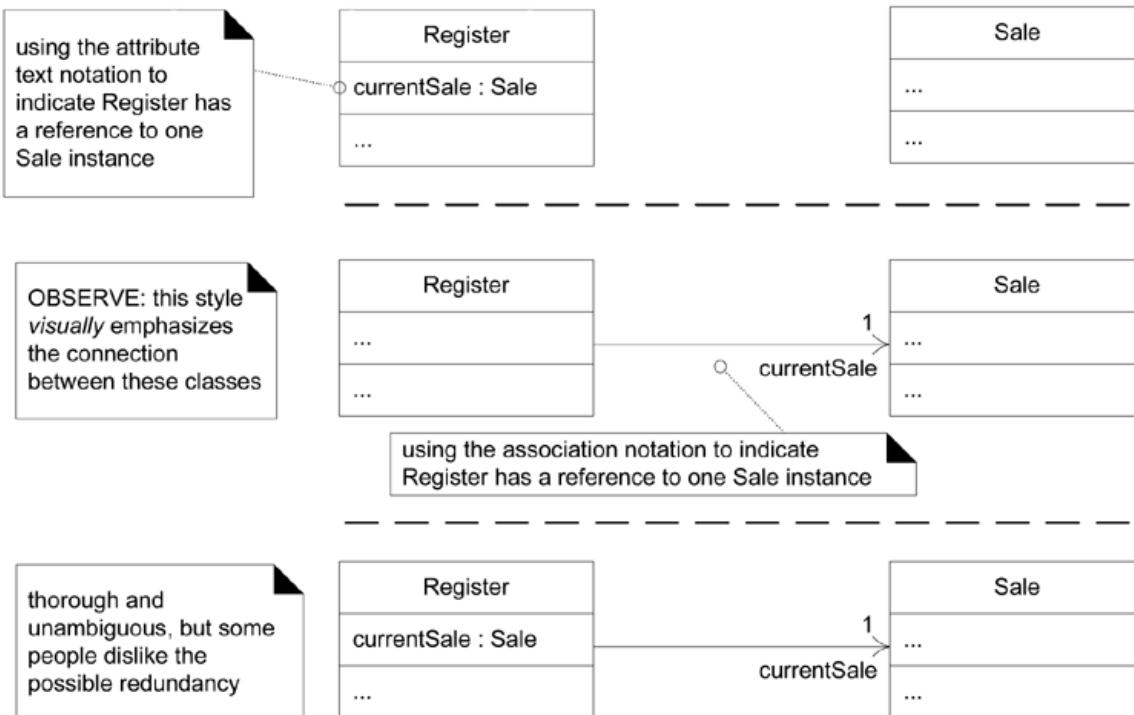
ATTRIBUTE TEXT VS. ASSOCIATION LINES



CLASS DIAGRAM

UML attributen

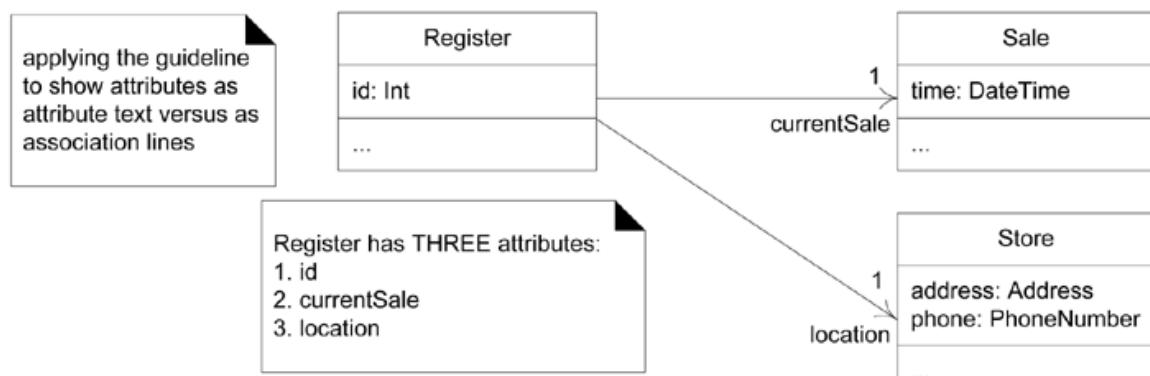
- Attribute text vs. association lines



CLASS DIAGRAM

UML attributen

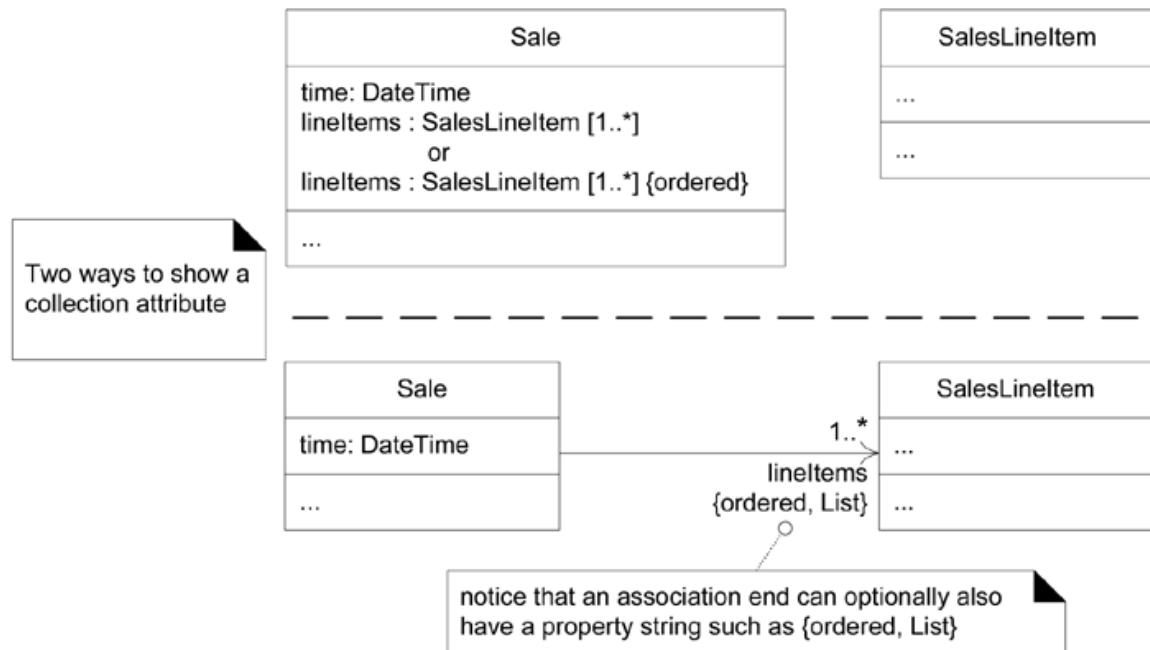
- Attribute text vs. association lines
 - Tip
 - attribute text notatie voor data type objects (Boolean, Date, Time, Number, String, ...)
 - association lines voor alle andere attributen

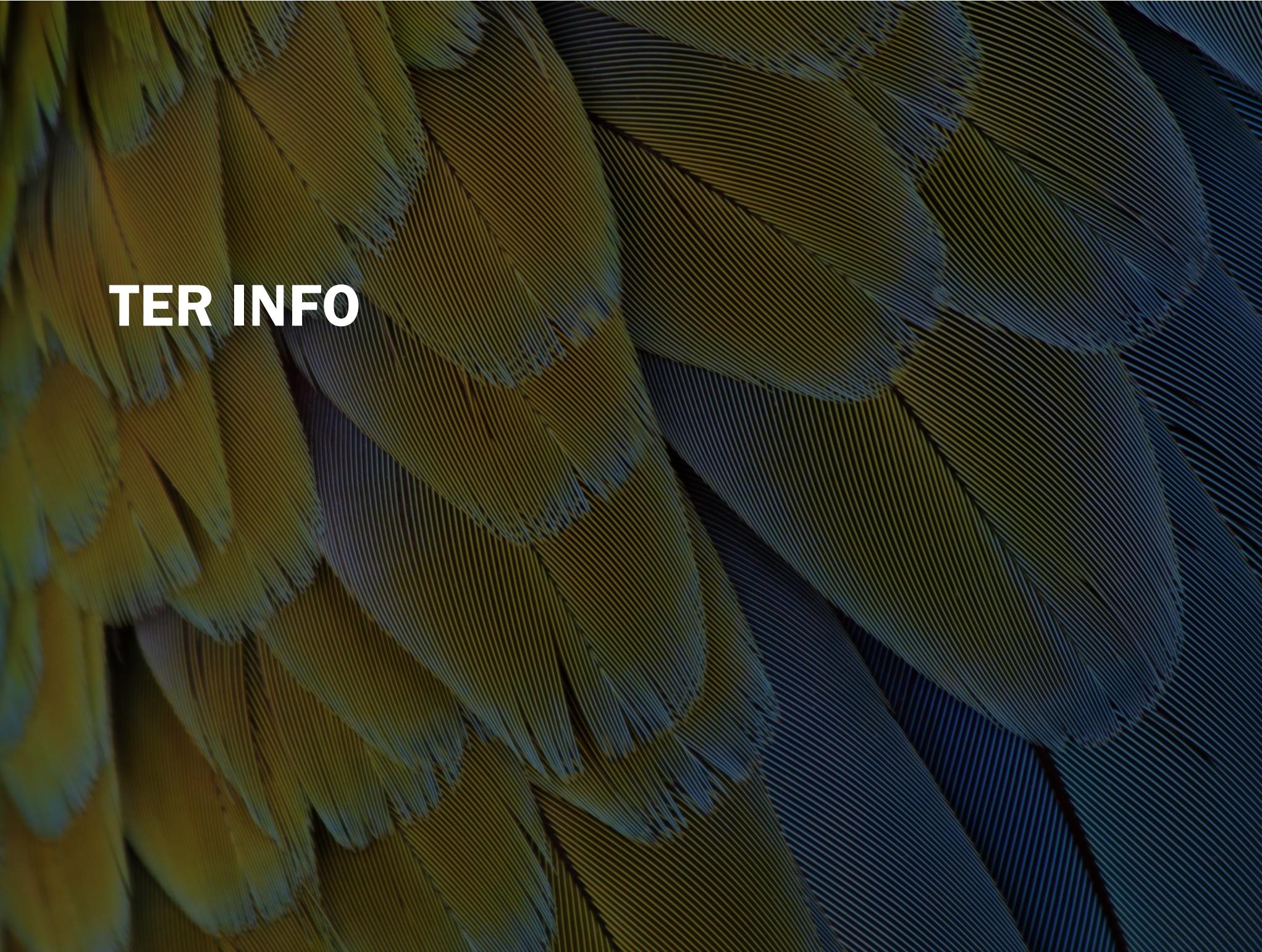


CLASS DIAGRAM

UML attributen

- Association lines



A close-up photograph of a bird's feathers, showing a vibrant pattern of yellow, green, and blue. The feathers are densely packed and curve upwards and outwards. The lighting highlights the fine texture and color variations of the plumage.

TER INFO

UML ATTRIBUTEN

- Afgeleide attributen (Derived Attributes)
 - Worden berekend op basis van andere attributen.
 - Worden aangeduid met een "/" (scheuine streep) voor de naam.
 - Voorbeeld:
 - /leeftijd → Wordt berekend op basis van het attribuut *geboortedatum*.

```
«method»  
// pseudo-code or a specific language is OK  
public void enterItem( id, qty )  
{  
    ProductDescription desc = catalog.getProductDescription(id);  
    sale.makeLineItem(desc, qty);  
}
```



UML operaties (methoden)

- Uitzonderlijk kan een stukje code bijgevoegd worden

CLASS DIAGRAM