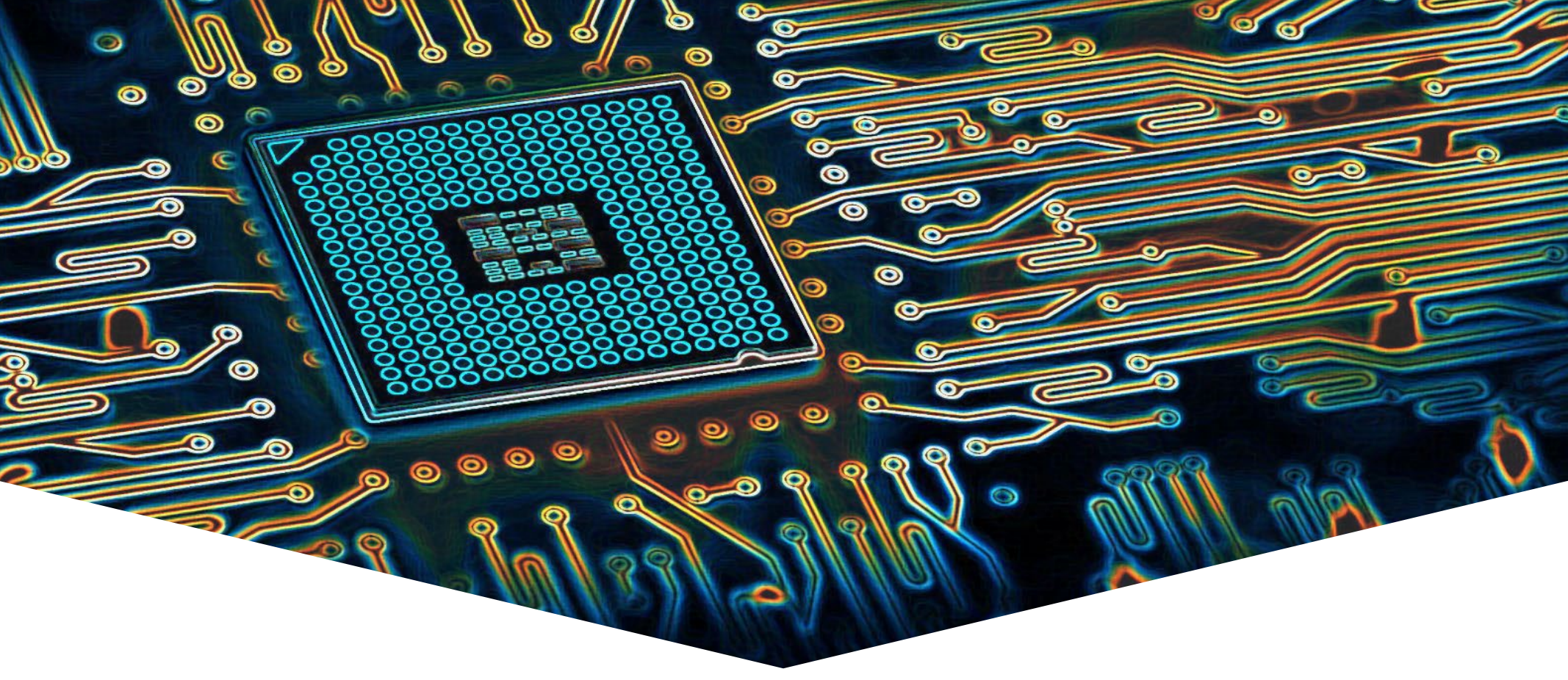


EMS31 Kwartaal 4 Week 7: UML deel 1

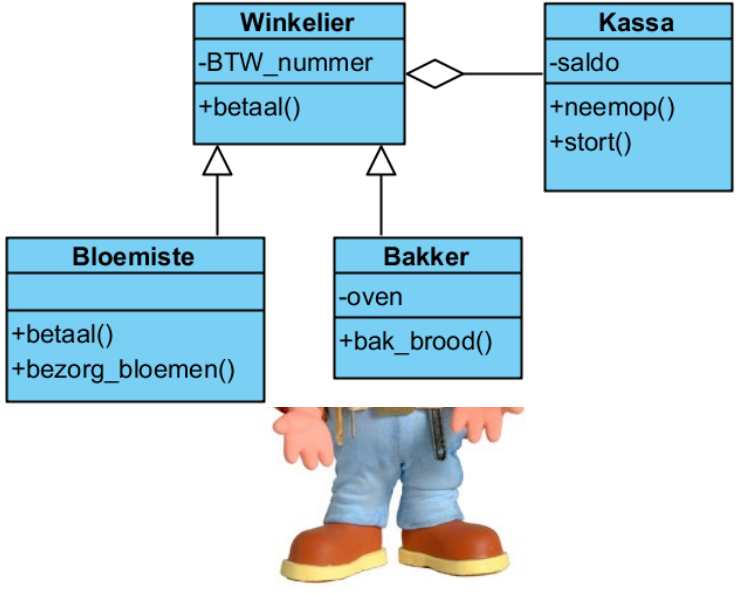
Leerdoelen Week 8 Les 1

Leerdoelen week 8 les 1. Je leert hoe je:

- UML kunt gebruiken om een model van de software te maken;
- Een klassediagram kunt maken.

Software Engineering Fases

- Wat wil de klant?
  - Analyse (wat zijn de eisen?).
- Hoe kun je het maken?
  - Ontwerp (gestructureerd of OO).
- W…ken?
  - Implem… (schrijven van de code).
- Werkt he…
  - Testen (verifieren and valideren).



Verificatie:
Hebben we het system goed gebouwd?
Voldoet het system aan de specificatie?

Validatie:
Hebben we het goede system gebouwd?
Voldoet de specificatie aan we wensen van de klant?

Software Engineering

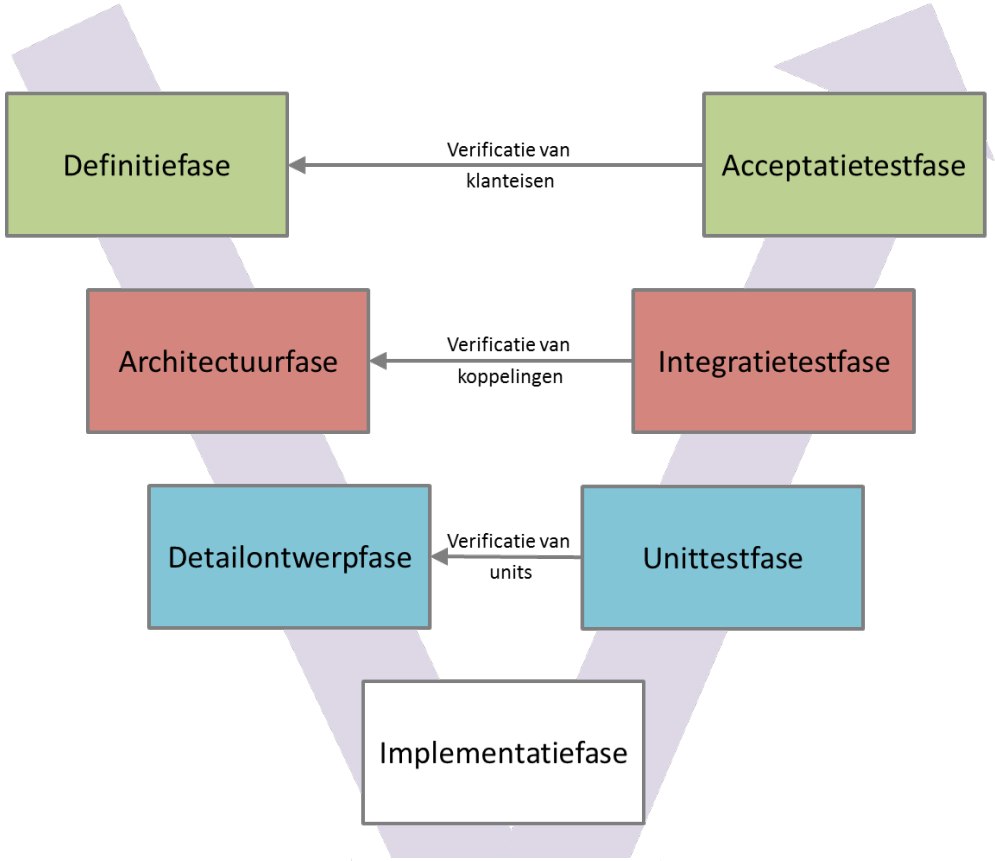
V-Model

- Traditionele methode.
- Geschikt voor projecten met weinig risico.
- Past goed bij gestructureerde aanpak.

Spiraal methode (Evolutionaire methode.)

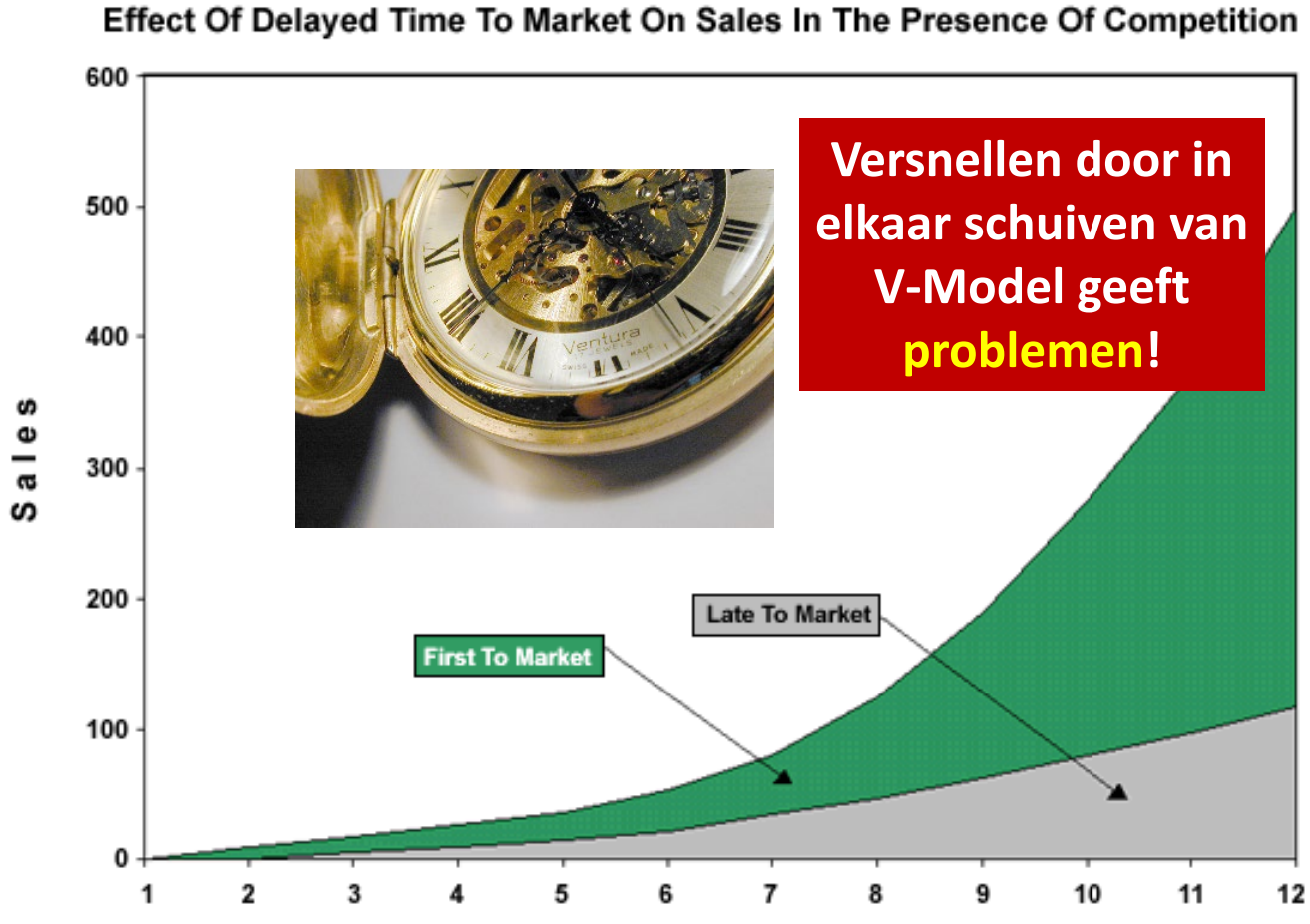
- Moderne methode.
- Geschikt voor projecten met veel risico (onzekerheid).
- Past goed bij OO aanpak.

V-Model



Gehele systeem in één keer….

Tijdsdruk...

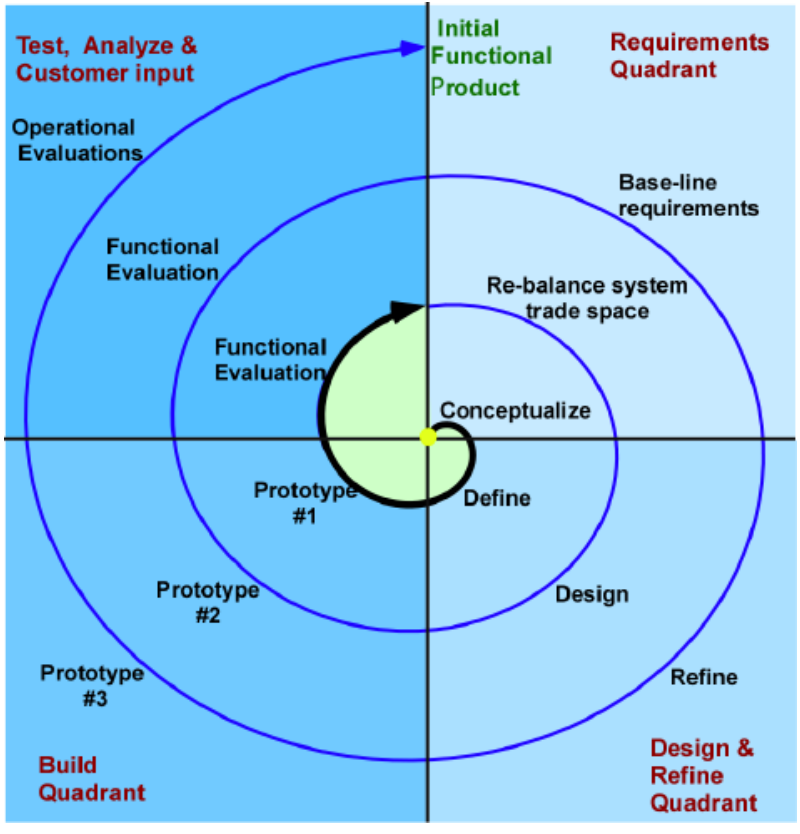


Spiraal = cyclisch

Test

Analyse

Systeem wordt stapje voor stapje verder uitgebreid…

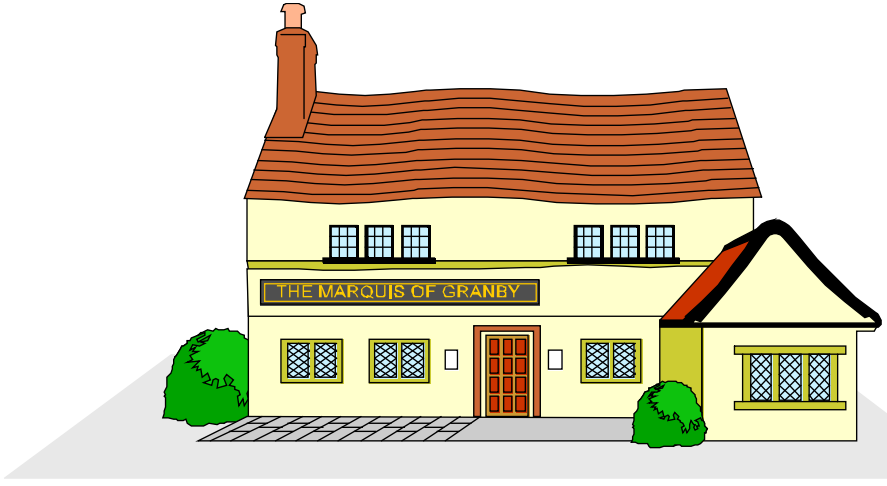


Typische duur van één cyclus: 3 weken!

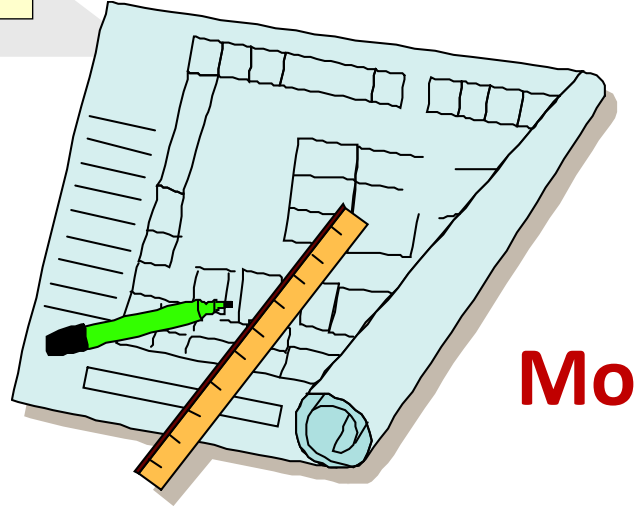
Implement

Design

Modelleren



Werkelijkheid



Model

Wat modelleren we?

- Analyse
  - Maak een model van de te automatiseren werkelijkheid of van het op te lossen probleem.
    - Domeinkennis → structuur.
    - Functionele eisen → gedrag.
- Ontwerp
  - Maak een model van de oplossing.
- Implementatie
  - Maak een model van het programma.

Bij OOA+OOD+OOP werken we steeds aan *hetzelfde model* (seamless development)

Unified Modeling Language

- Wikipedia (https://nl.wikipedia.org/wiki/Unified_Modeling_Language):
  - De Unified Modeling Language, afgekort UML, is een modelmatige taal om objectgeoriënteerde analyses en ontwerpen voor een informatiesysteem te kunnen maken.
  - UML is ontworpen door Grady Booch, James Rumbaugh en Ivar Jacobson in de jaren negentig en het is sinds 1997 een standaard.
  - Kenmerkend is dat de UML-modellen een grafische weergave zijn van bepaalde aspecten van het informatiesysteem.
  - Bekende bedrijven: IBM, Oracle, HP, TI

UML diagrammen

- Statische structuur van programma.
  - UML Klassediagram.
  - UML Objectdiagram.
- Dynamisch gedrag van programma.
  - UML Use-case-diagram.
  - UML Sequentiediagram.
  - UML Communicatiediagram.
  - UML Toestandsdiagram.
  - UML Activiteitsdiagram.

Unified Modeling Language

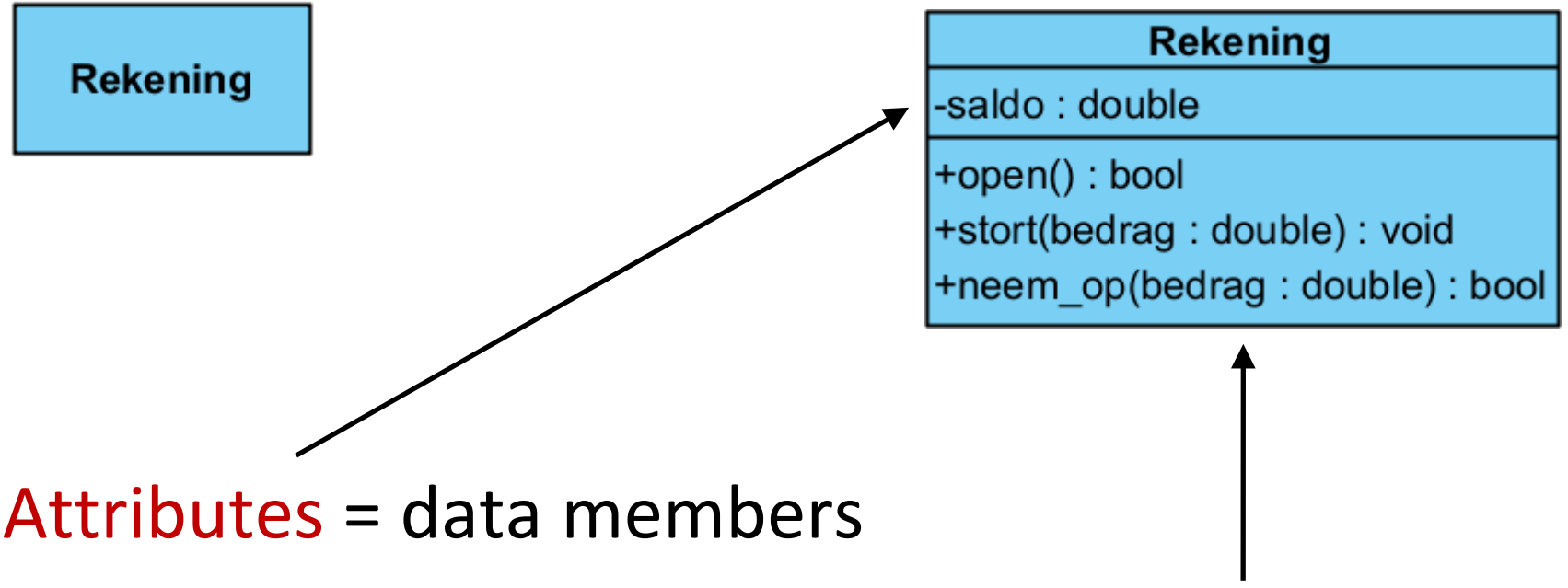
https://people.cs.kuleuven.be/%7Ejoost.vennekens/DN/OO-UML-cursus.pdf

- UML inleiding.
  - Hoofdstuk 1 en 2.
- UML Klasse- en objectdiagram.
  - Hoofstuk 3.
- UML Use-case-diagram.
  - Paragraaf 5.1.
- UML Sequentie- en communicatiediagram.
  - Paragraaf 5.2.
- UML Toestands- en Activiteitsdiagram.
  - Hoofdstuk 6.

Visual Paradigm for UML

- Tekenen van UML diagrammen.
- Omzetten UML naar C++ (of Java, C#, Python enz).
- Omzetten C++ (of Java, C#, Python) naar UML.
- https://www.visual-paradigm.com/
  - Beperkte versie gratis beschikbaar (geen code-generatie)
  - Volledige versie 30 dagen proefversie

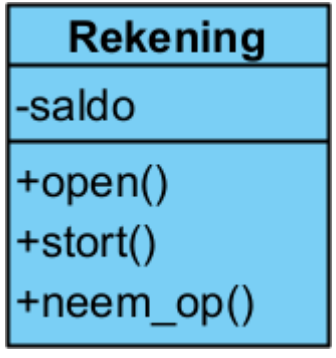
Klassediagram



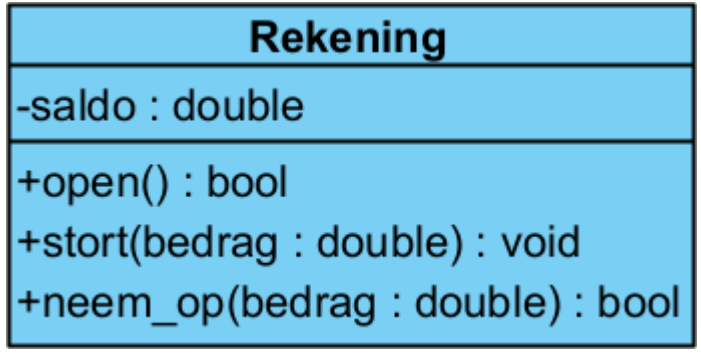
Attributes = data members

Operations = member functions = messages

Fases



Analyse

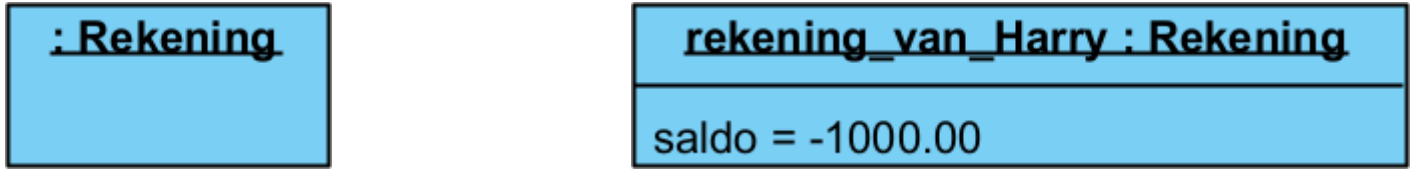


Ontwerp

Tijdens het ontwikkelen van het model voeg je steeds meer details toe.

Bij een *volgende ronde* van de spiraal kunnen we functionaliteit *toevoegen*, bijvoorbeeld: `+geefRente()`

Objectdiagram



- Object is een instantie van een Class
- Attributes = data members (met waarde)

Sommige instanties bestaan gedurende de gehele levensduur van het programma, anderen zijn *dynamisch* (ze verschijnen en verdwijnen)

Code generatie

Twee methoden in Visual Paradigm

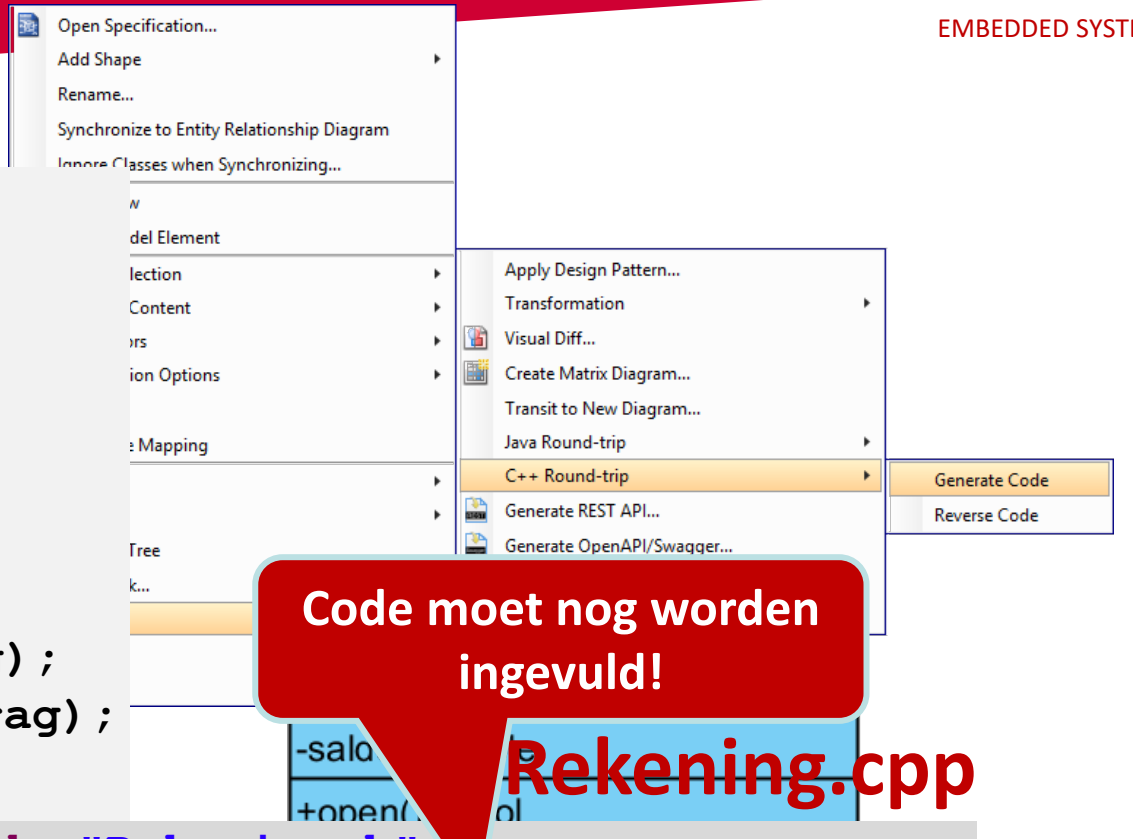
- Instant generator: produceert broncode op basis van het klassenmodel.

- Round-trip engineering: Inlezen van (bestaande) code om het model gesynchroniseerd te houden met de code.

Implementatie / Code generatie

Rekening.h

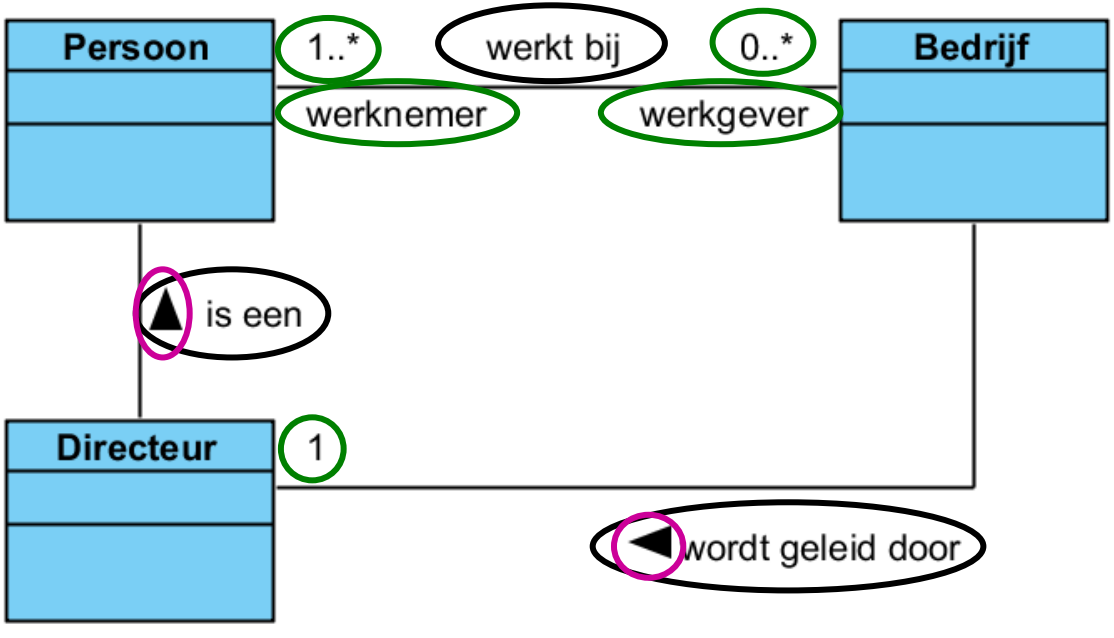
```
#ifndef REKENING_H
#define REKENING_H

class Rekening {
private:
    double saldo;
public:
    bool open();
    void stort(double bedrag);
    bool neem_op(double bedrag);
};

#endif
```



Rekening.cpp

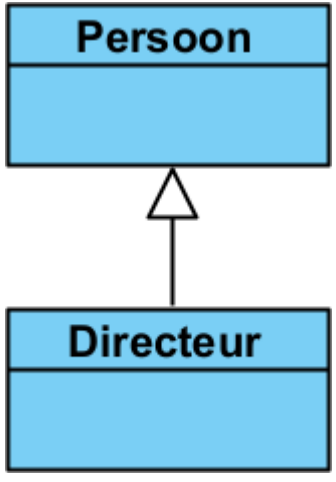
```
#include "Rekening.h"
bool Rekening::open() {}
void Rekening::stort(double bedrag) {}
bool Rekening::neem_op(double bedrag) {}
```

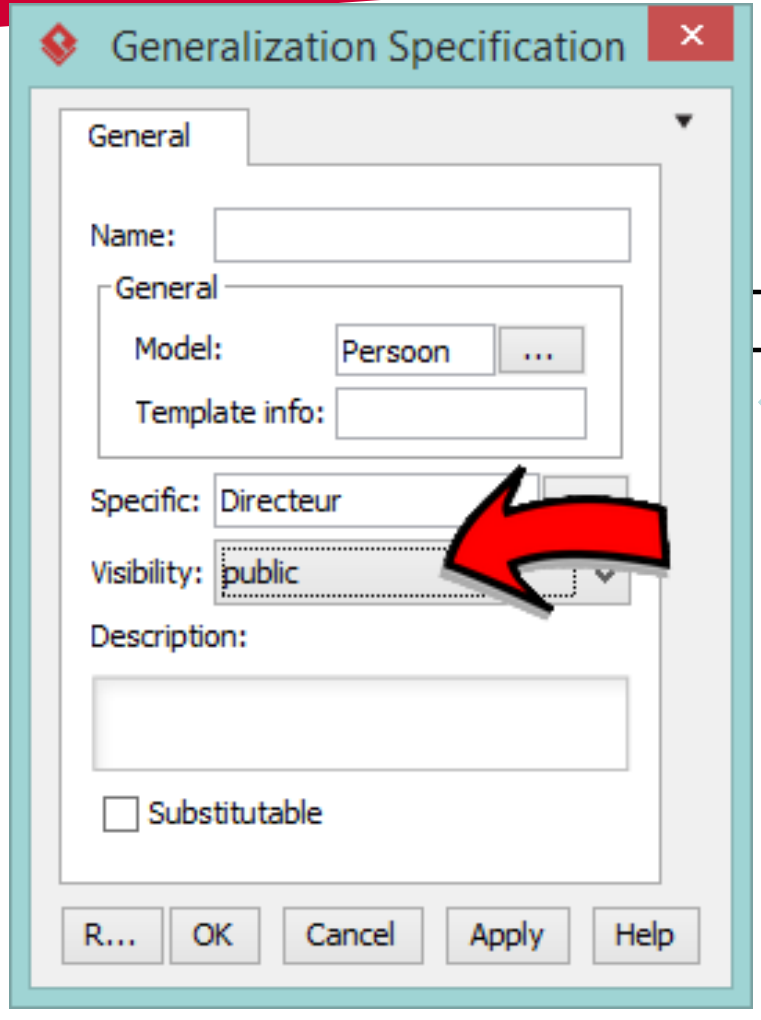
Associatie



Labels: a*ltijd* invullen!
Multipliciteit: *eventueel* invullen.
Rol: *eventueel* invullen.
Leesrichting: gebruiken om "*onnatuurlijke*" leesrichting aan te geven.

Overerving (... is een ...)

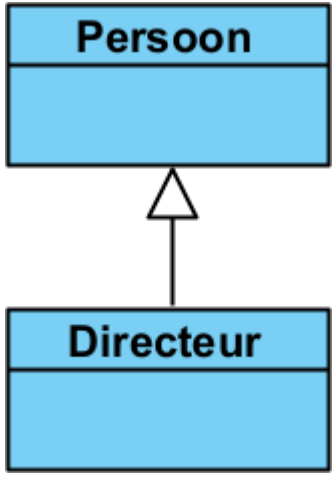





```
_H
_H

Persoon {
```

Niet goed!

Overerving (... is een ...)

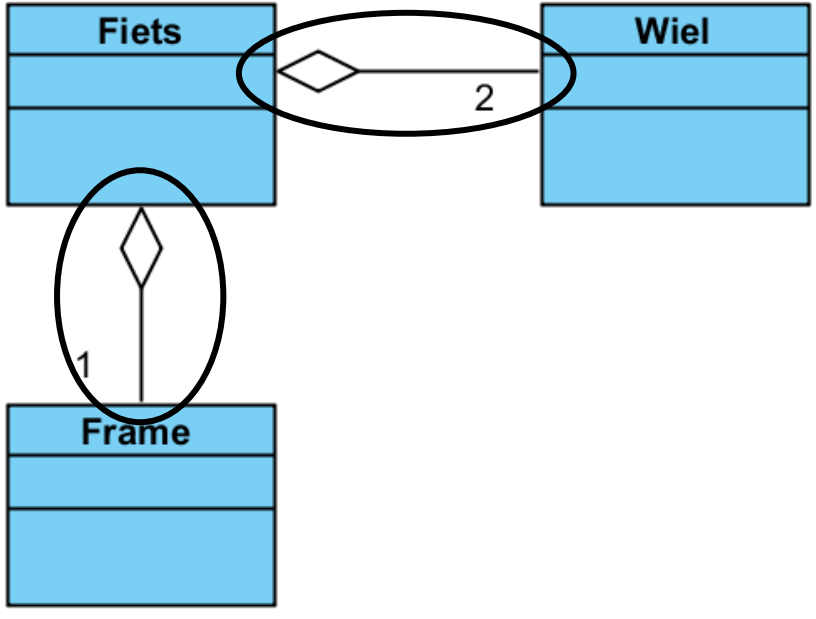


Directeur.h

Bijna goed!

```
#ifndef DIRECTEUR_H
#define DIRECTEUR_H
#include "Persoon.h"

class Directeur : public Persoon {
};

#endif
```

Aggregatie (... heeft een ...)



Fiets heeft:

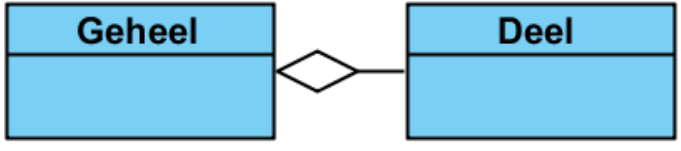
- 2 Wielen
- 1 Frame
- ...

Associatie

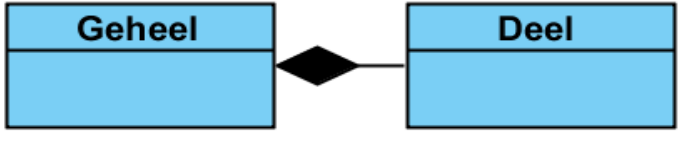
- Overerving
- Aggregatie
- Compositie

Beginnende UML modelleerders gebruiken vaak ten onrechte compositie!

Een compositie is een speciaal soort aggregatie.

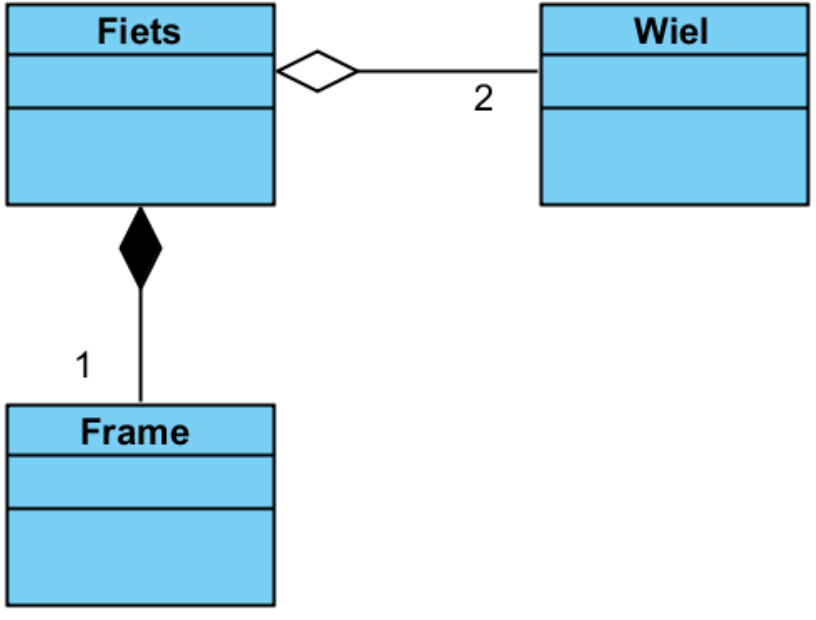


- Aggregatie
  - “**heeft een**” relatie.
  - vaag gedefinieerd



- Compositie
  - “**bevat een**” relatie.
  - deel behoort maar bij 1 geheel
  - levensduur deel <= levensduur geheel

Compositie (... bevat een ...)

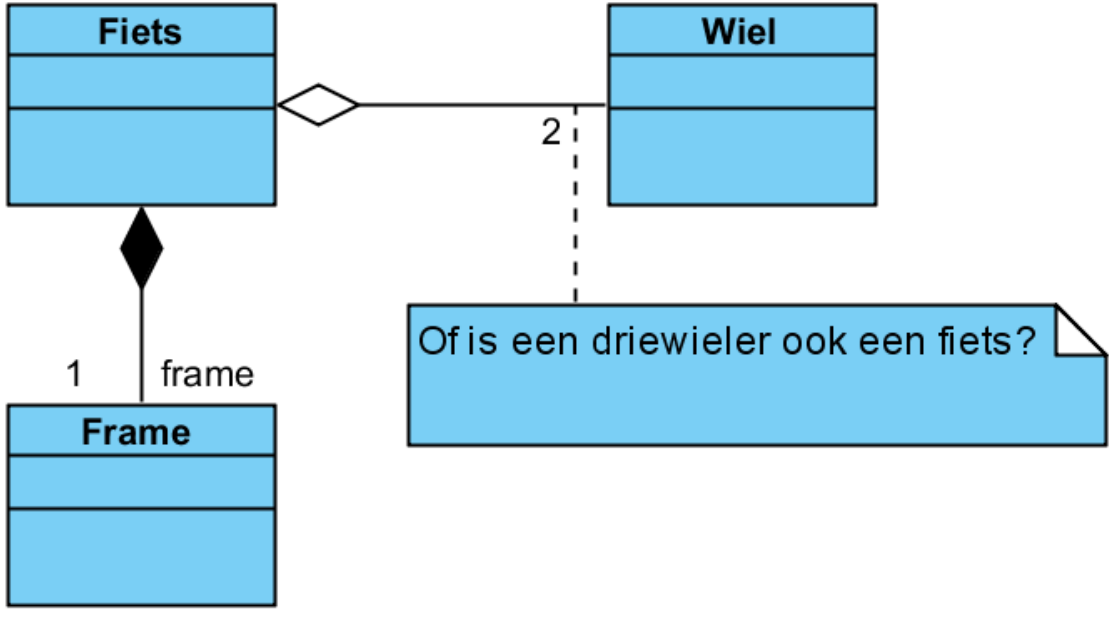


Welke aggregaties zijn composities?

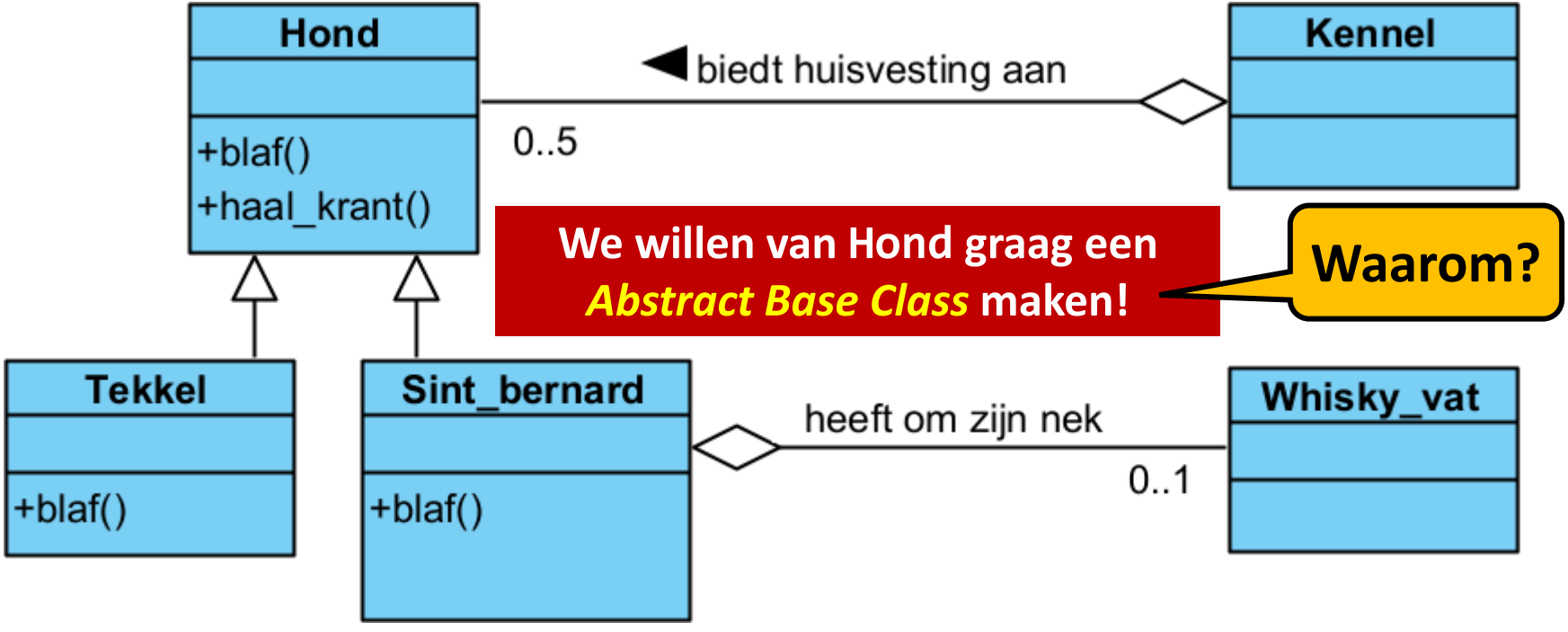
- Mens --- Nier
- Mens --- Hersenen
- PC --- Hard Disk
- PC --- CPU
- CPU --- Transistor
- Radio --- Transistor

Let op! Het goede antwoord is afhankelijk van de applicatie.

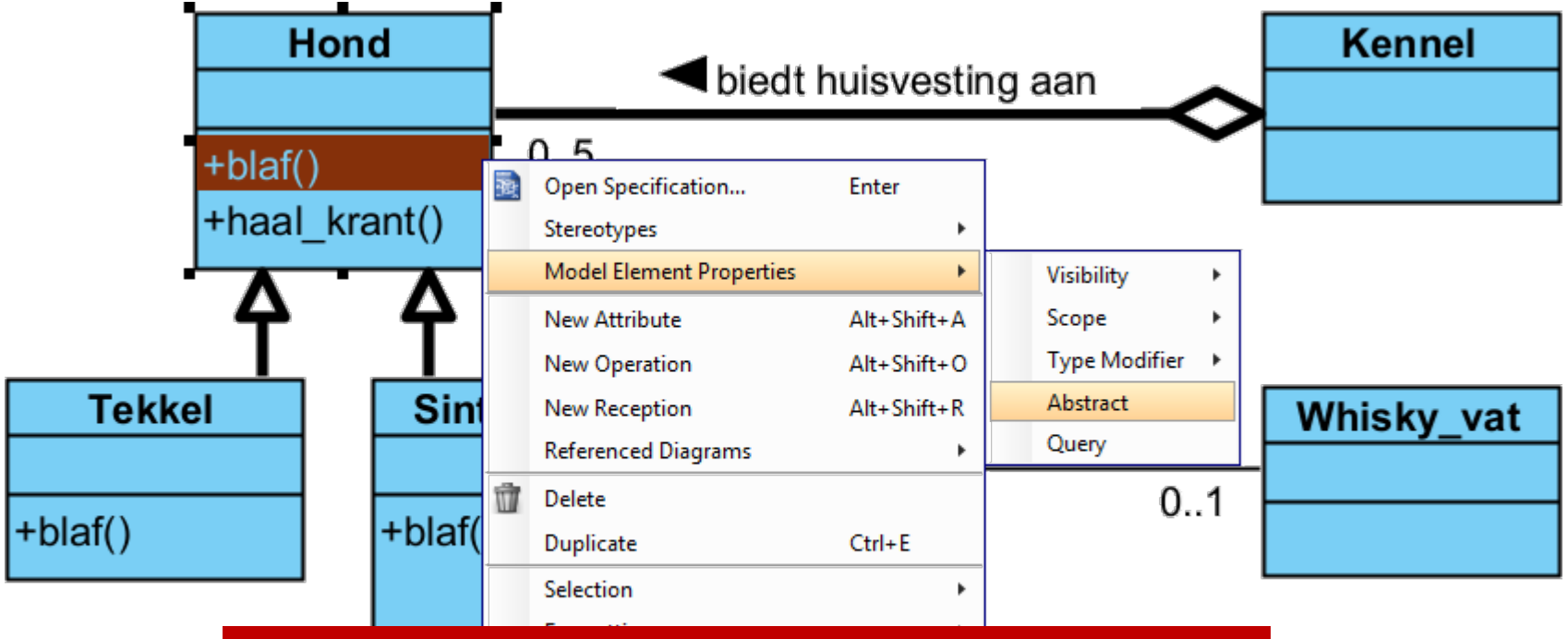
Commentaar



Voorbeeld

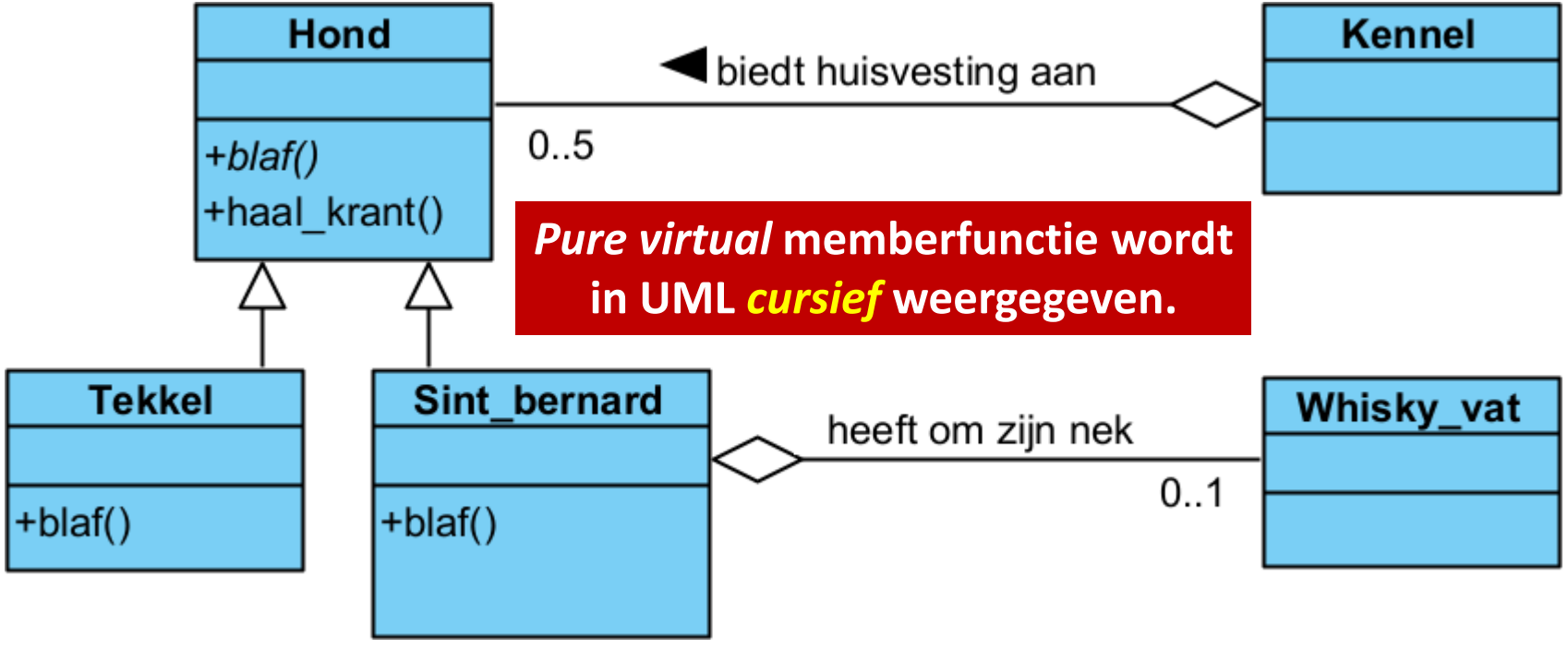


Pure virtual member function

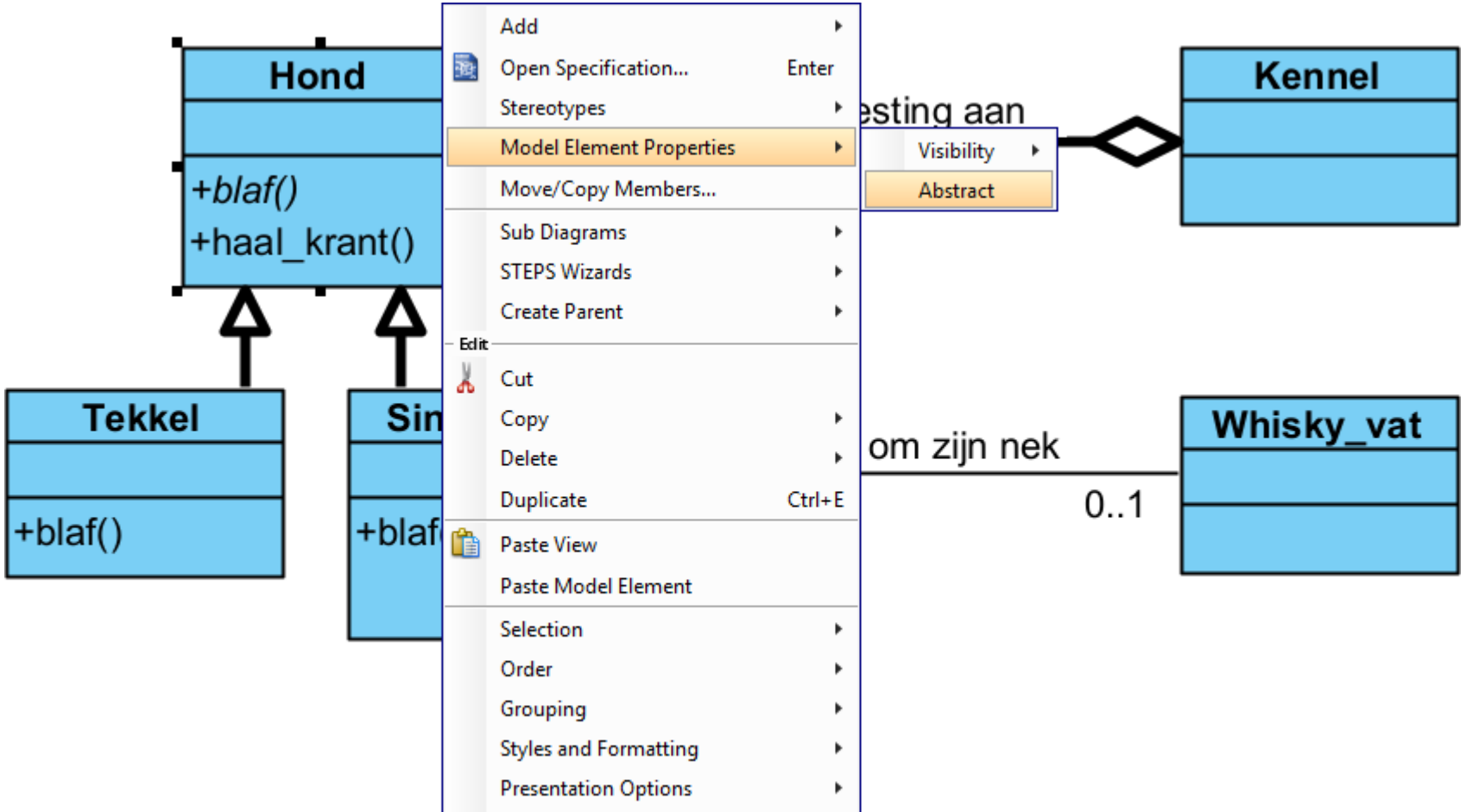


Abstract geeft aan dat het een *Pure virtual* memberfunctie wordt deze dus *moet* worden geïmplementeerd in een afgeleide klasse.

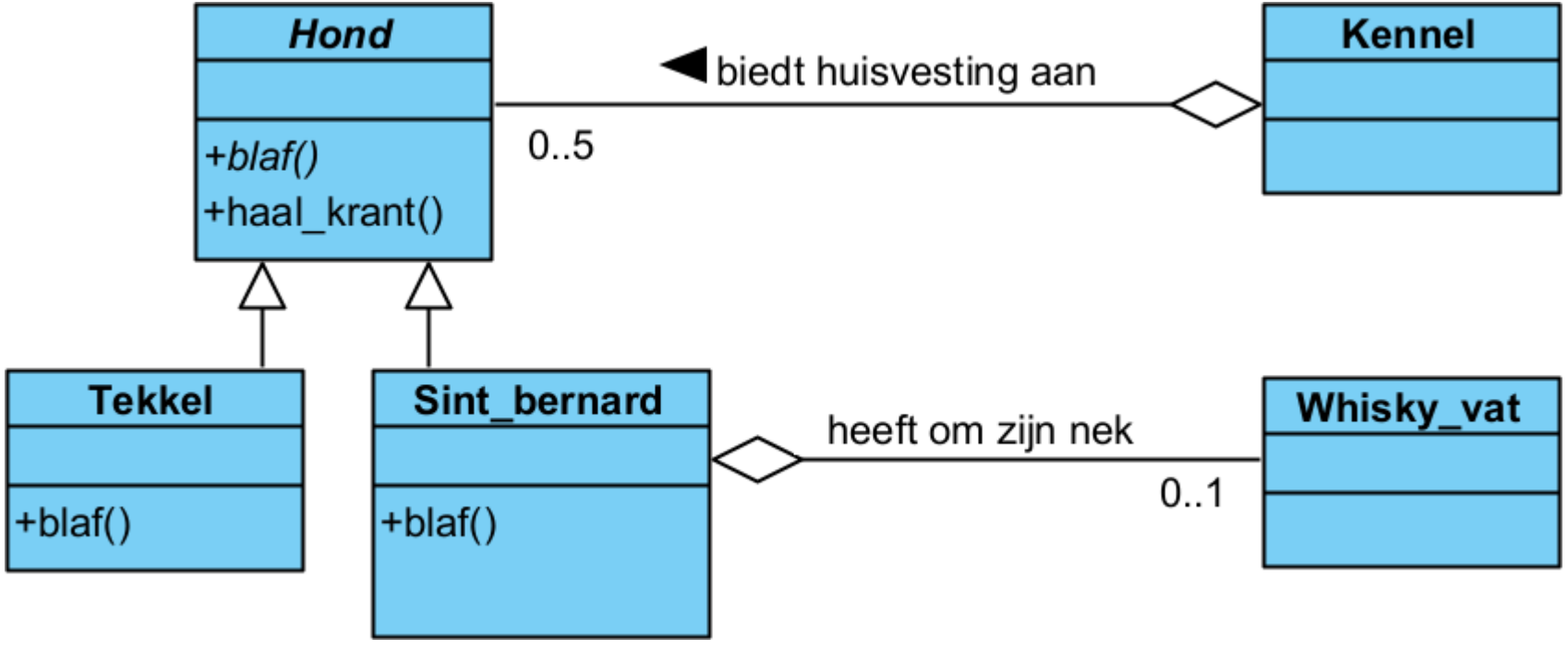
Pure virtual member function



Abstract Base Class

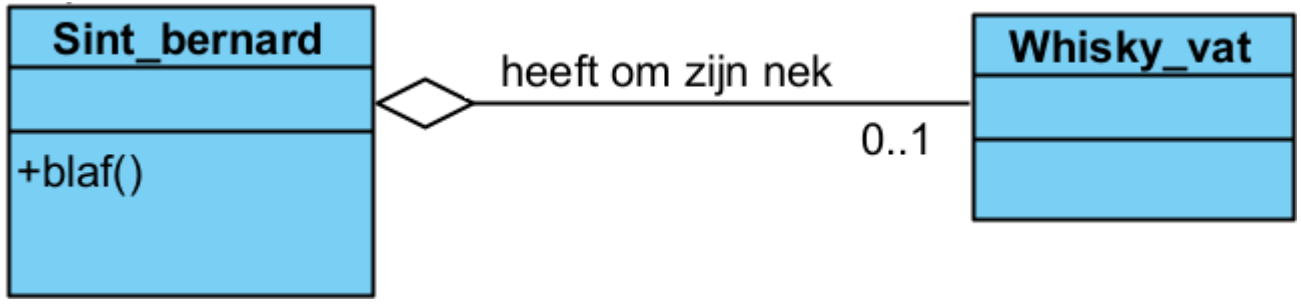


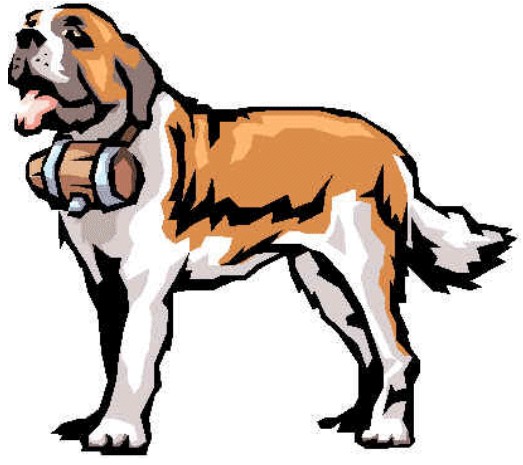
Abstract Base Class



Abstract Base Class **wordt ook** ***cursief*** **weergegeven.**

Aggregatie implementatie





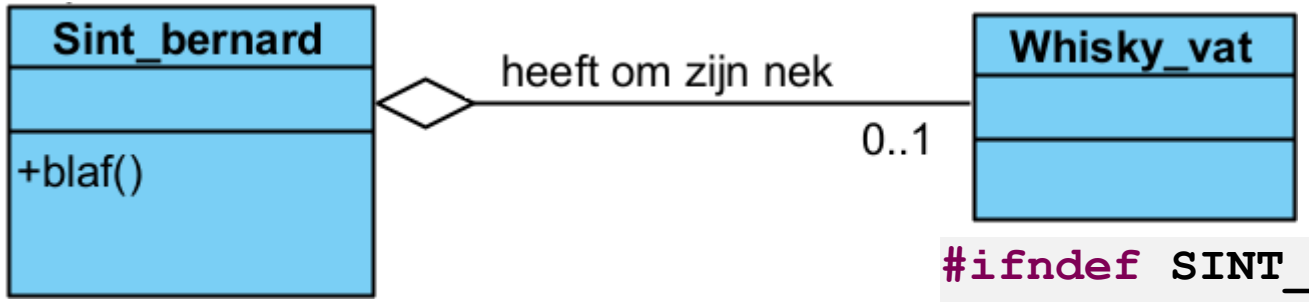
Hoe kunnen we een 0..1 aggregatie implementeren in class `SintBernard`?

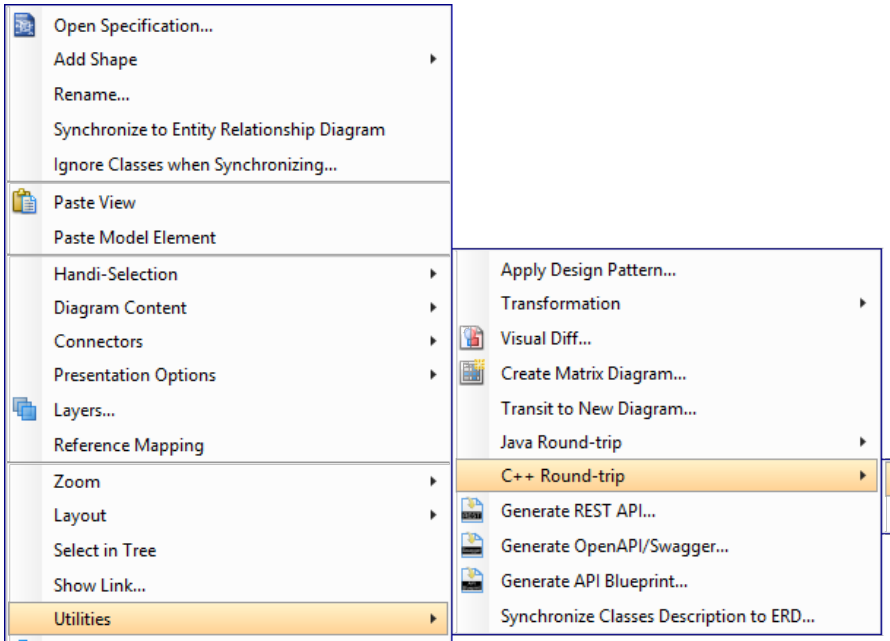
- `WhiskeyVat` data member?
- `WhiskeyVat&` data member?
- `WhiskeyVat*` data member?

Niet 0..1 maar 1.

Andere naam voor een `Whisky_vat` dat al bestaat (?)

Aggregatie implementatie

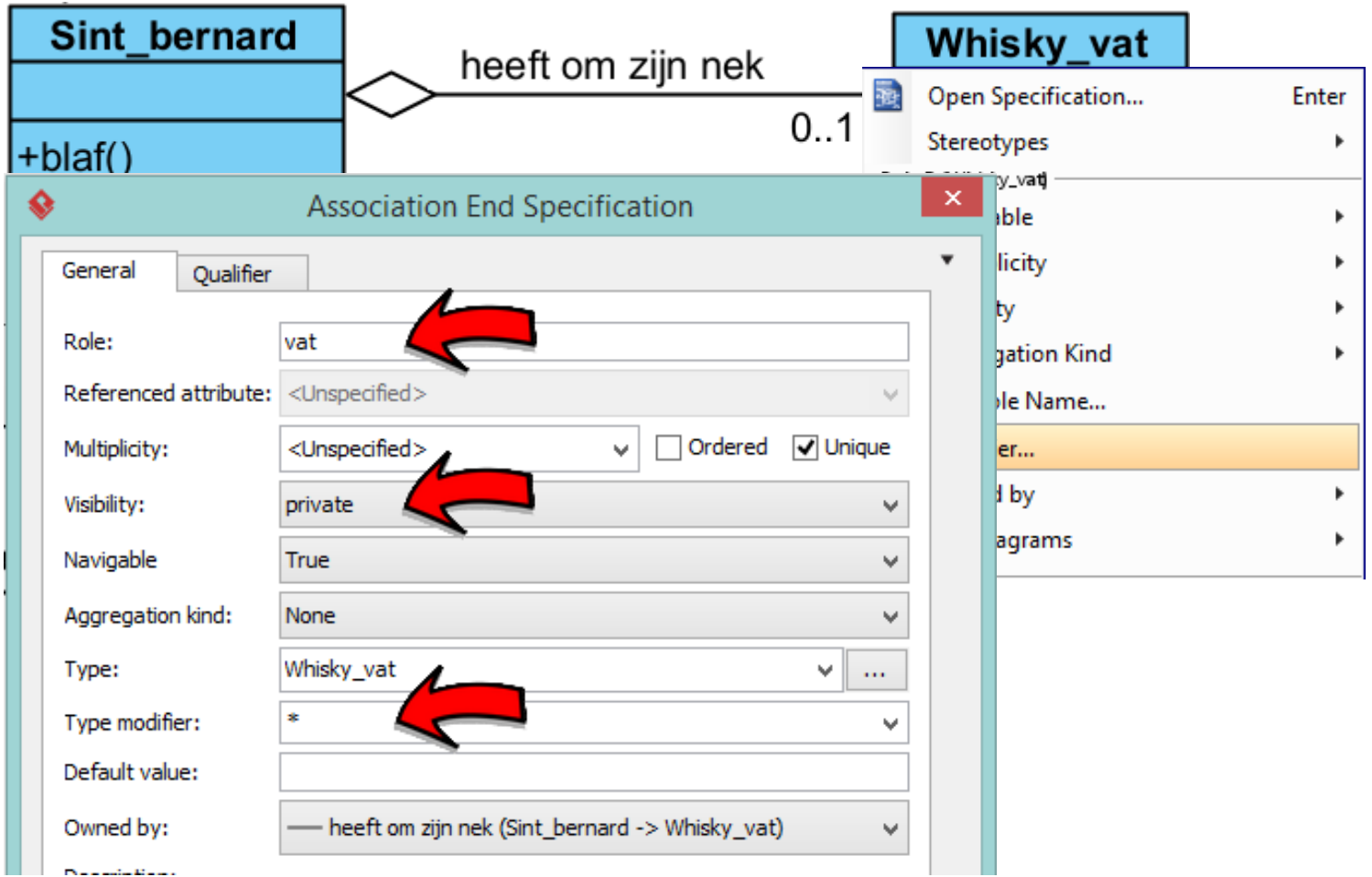





```
#ifndef SINT_BERNARD_H
#define SINT_BERNARD_H

class Sint_bernard : Hond {
public:
    void blaf();
};

#endif
```

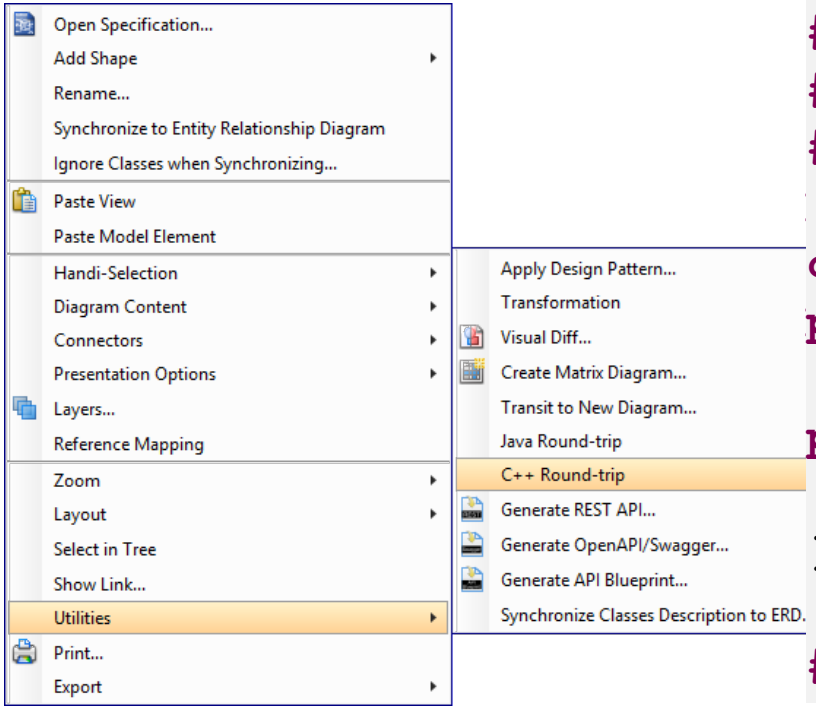
Waar is de Whisky?

Aggregatie implementatie



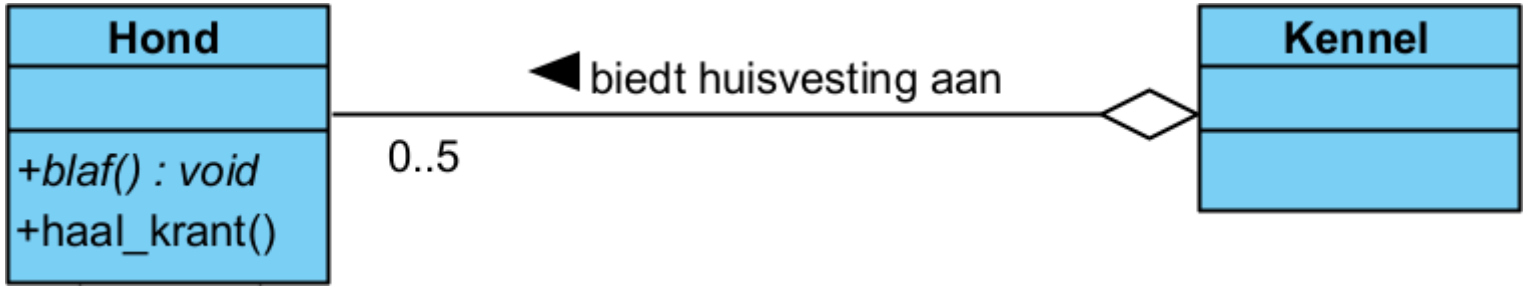
Aggregatie implementatie

Bijna goed!




```
#ifndef SINT_BERNARD_H
#define SINT_BERNARD_H
#include "Hond.h"
#include "Whisky_vat.h"

class Sint_bernard : public Hond {
public:
    void blaf();
private:
    Whisky_vat* vat;
};

#endif
```

Aggregatie implementatie



Hoe kunnen we een 0..5 aggregatie implementeren in class `Kennel`?

- `array` met `Hond*` -ers
- `array<Hond*, 5>`
- `vector<Hond*>`

Beter! Maar niet beschikbaar in Visual Paradigm

Goed! Wel beschikbaar in Visual Paradigm, ook bruikbaar voor 0..*

Aggregatie implementatie


```
#ifndef KENNEL_H
#define KENNEL_H
#include <vector>
#include "Hond.h"

class Kennel {
private:
    std::vector<Hond*> inwoners;
};

#endif
```

Volgende les…

Het gedrag van een systeem beschrijven met UML

- UML Use-case-diagram.
  - Paragraaf 5.1.
- UML Sequentie- en communicatiediagram.
  - Paragraaf 5.2.
- UML Toestands- en Activiteitsdiagram.
  - Hoofdstuk 6.

Eindopdracht 2

Eindopdracht 2 telt voor 50% mee en bestaat uit 3 delen:

- Het schrijven van een C++ programma voor de MSP430G2553. (60 punten, LD4)
- Het uitvoeren en beschrijven van de opdrachten van week 7.2. (20 punten, LD5)
- Modelleren van een stuk software uit je project met UML (20 punten, LD6)

Aan de slag!

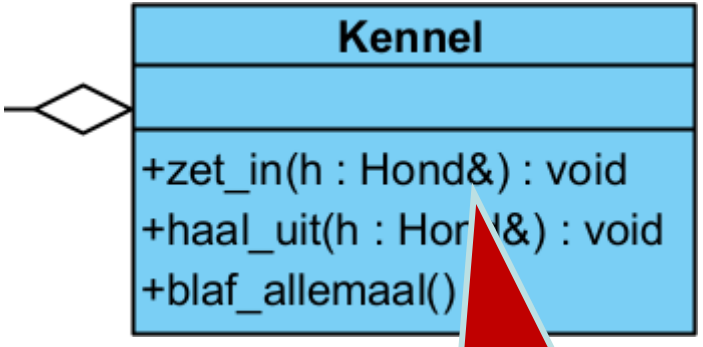
Aan de slag met Eindopdracht_2.pdf

Voorbeeld

Schrijf een testprogramma voor de hiervoor gemaakte class `Kennel`.

- Zet een `Sint_bernard` genaamd `boris` in de `Kennel` genaamd `k`.
- Zet een `Tekkel` genaamd `harry` in de `Kennel` `k`.
- Zet een `Sint_bernard` genaamd `felix` in de `Kennel` `k`.
- Laat alle honden in de `Kennel` `k` blaffen.
- Haal `harry` uit de `Kennel` `k`.
- Laat alle honden in de `Kennel` `k` blaffen.

Kennel in UML en headerfile



Waarom Hond& als parameter?

Headerfile kan automatisch gegenereerd worden als we de juiste opties in Visual Paradigm kiezen.

```
#ifndef KENNEL_H
#define KENNEL_H
#include <vector>
#include "Hond.h"

class Kennel {
private:
    std::vector<Hond*> inwoners;
public:
    void zet_in(Hond& h);
    void haal_uit(Hond& h);
    void blaf_allemaal();
};

#endif
```

Test Driven Development

Schrijf altijd eerst het testprogramma!

```
#include <iostream>
using namespace std;
#include "Sint_bernard.h"
#include "Tekkel.h"
#include "Kennel.h"

int main() {
    Sint_bernard boris;
    Kennel k;
    k.zet_in(boris);
    Tekkel harry;
    k.zet_in(harry);
    Sint_bernard felix;
    k.zet_in(felix);
    cout << "Alle honden in de kennel blaffen:\n";
    k.blaf_allemaal();
    k.haal_uit(harry);
    cout << "\nAlle honden in de kennel blaffen:\n";
    k.blaf_allemaal();
}
```

Gewenste uitvoer:

```
Alle honden in de kennel blaffen:
WOEF WOEF kef kef WOEF WOEF
Alle honden in de kennel blaffen:
WOEF WOEF WOEF WOEF
```

Kennel implementatie


```
#include <iostream>
#include <algorithm>
using namespace std;
#include "Hond.h"
#include "Kennel.h"

void Kennel::zet_in(Hond& h) {
    if (inwoners.size() < 5) inwoners.push_back(&h);
    else cout << "Kennel is al vol!\n";
}

void Kennel::haal_uit(Hond& h) {
    inwoners.erase(find(inwoners.begin(), inwoners.end(), &h));
}

void Kennel::blaf_allemaal() {
    for (auto hp: inwoners) {
        hp->blaf(); cout << " ";
    }
}
```