

EMS30 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

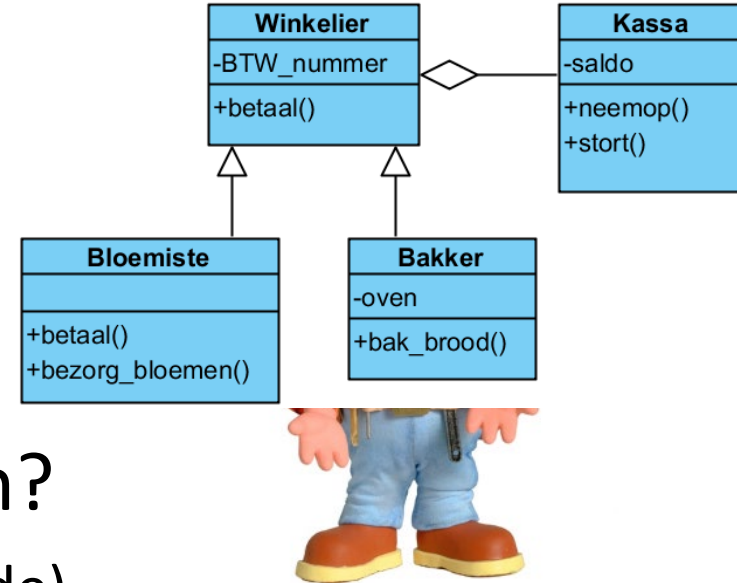
- Wat wil de klant?
- Analyse (wat zijn de eisen?).
- Hoe kun je het maken?
- Ontwerpen (gestructureerd of OO).
- Implementatie (schrijven van de code).
- Werkt het?
- Testen (verifiëren and valideren).

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?



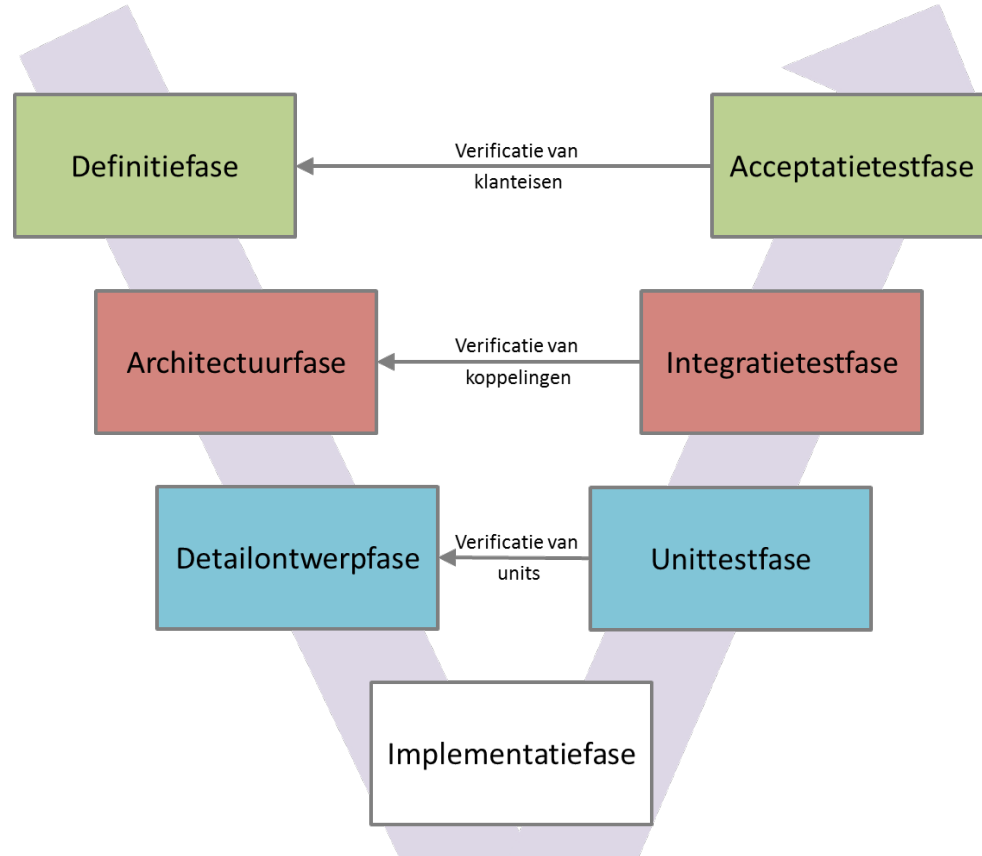
V-Model

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

Spiraal methode (Evolutionaire methode.)

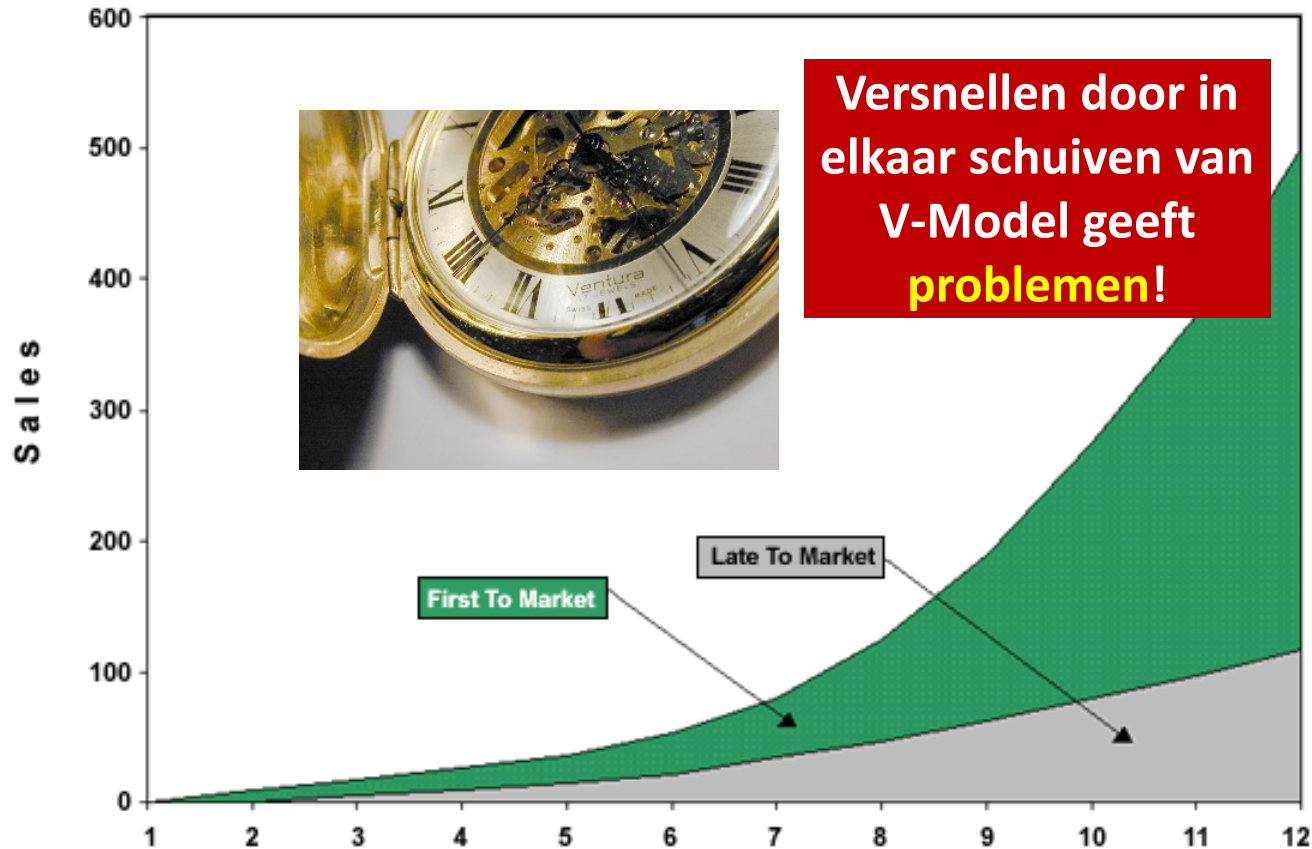
- Moderne methode.
- Geschikt voor projecten met veel risico (onzekerheid).
- Past goed bij OO aanpak.

V-Model

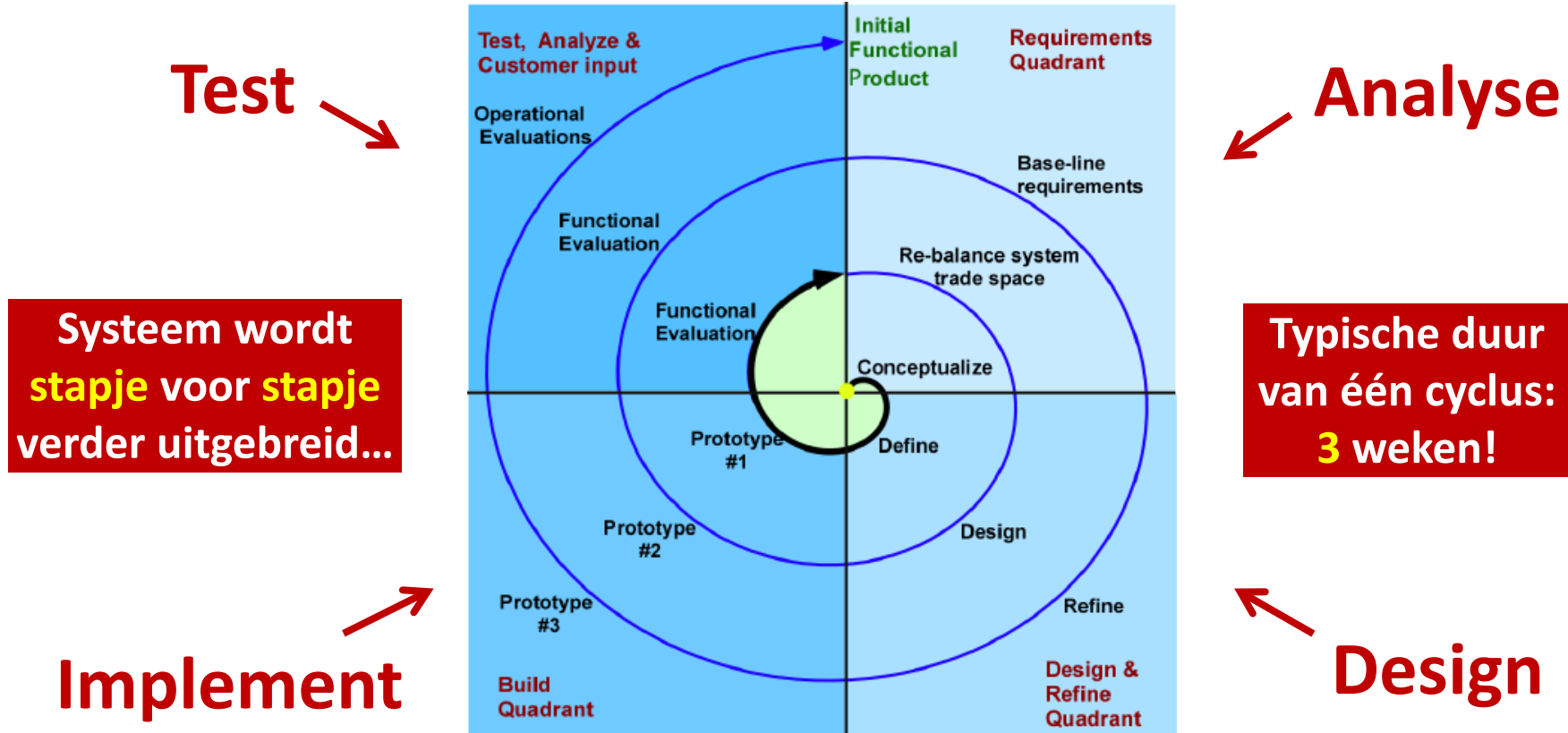


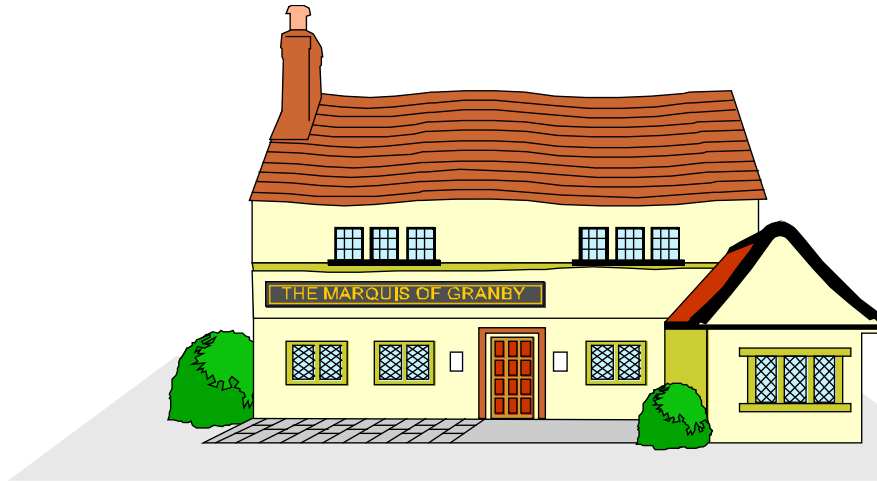
**Gehele systeem
in één keer....**

Effect Of Delayed Time To Market On Sales In The Presence Of Competition

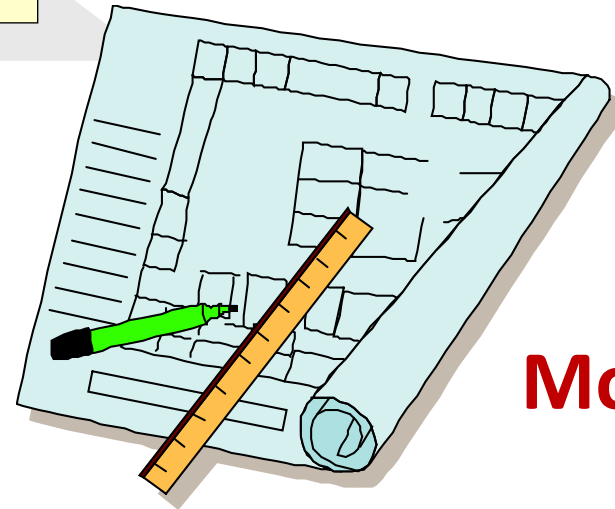


Spiraal = cyclisch





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)

- 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

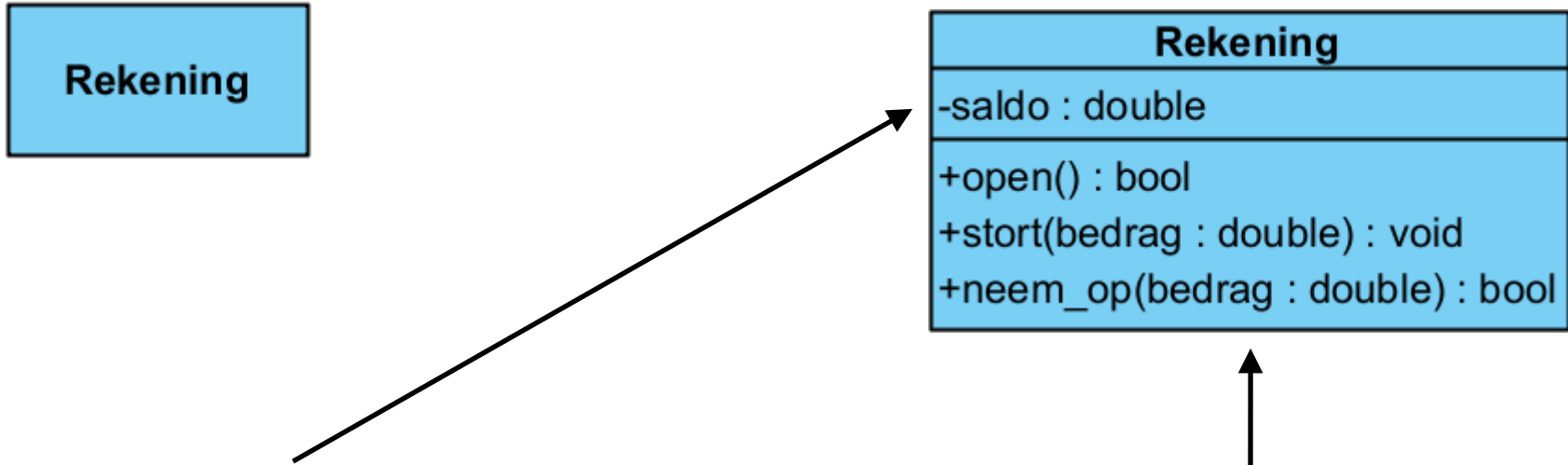
- **Statische structuur** van programma.
 - UML Klassediagram.
 - UML Objectdiagram.
- **Dynamisch gedrag** van programma.
 - UML Use-case-diagram.
 - UML Sequentiediagram.
 - UML Communicatiediagram.
 - UML Toestanddiagram.
 - UML Activiteitsdiagram.

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

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

- 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

Rekening
-saldo
+open() +stort() +neem_op()

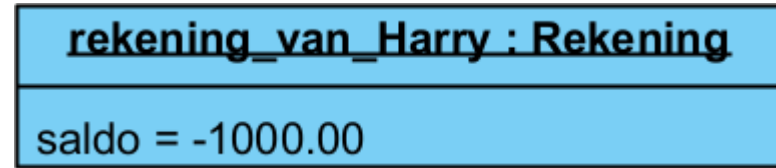
Analyse

Rekening
-saldo : double
+open() : bool +stort(bedrag : double) : void +neem_op(bedrag : double) : bool

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 ()



- 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)

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

EMBEDDED SYSTEMS

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
```

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



Code moet nog worden ingevuld!

Rekening.cpp

Associatie



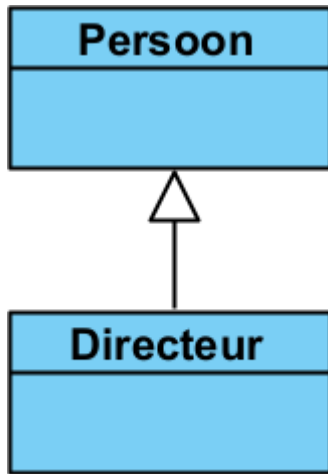
Labels: *altijd* invullen!

Multipliciteit: *eventueel* invullen.

Rol: *eventueel* invullen.

Leesrichting: gebruiken om “*onnatuurlijke*” leesrichting aan te geven.

Overerving (... is een ...)



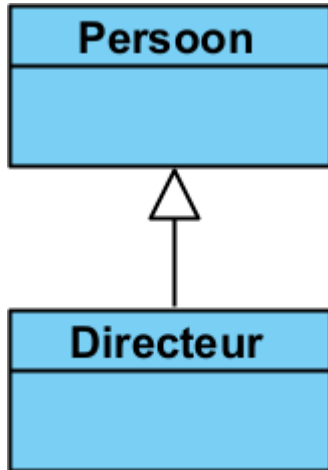
A dialog box titled 'Generalization Specification' with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- General**: A dropdown menu.
- Name**: An empty text input field.
- General**: A sub-dialog box containing:
 - Model**: A dropdown menu with 'Persoon' selected and an ellipsis button.
 - Template info**: An empty text input field.
- Specific**: A dropdown menu with 'Directeur' selected.
- Visibility**: A dropdown menu with 'public' selected.
- Description**: An empty text input field.
- Substitutable**

At the bottom, there are five buttons: 'R...', 'OK', 'Cancel', 'Apply', and 'Help'. A large red arrow points from the 'Specific' dropdown to the 'Visibility' dropdown.

A code snippet showing a class declaration: `Person {`. A red speech bubble with the text 'Niet goed!' (Not good!) points to the code. A red arrow from the dialog box points to the 'Visibility' dropdown in the dialog box.

Overerving (... is een ...)



Directeur.h

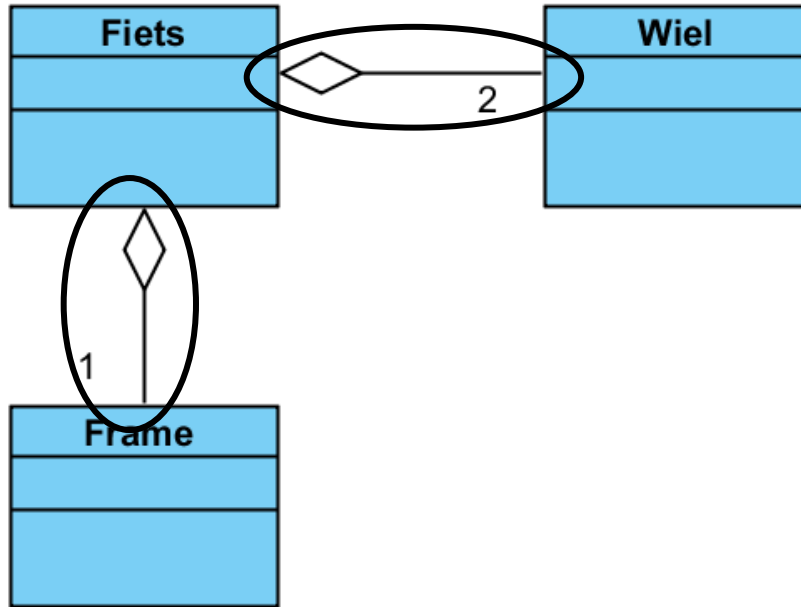
```
#ifndef DIRECTEUR_H
#define DIRECTEUR_H
#include "Persoon.h"

class Directeur : public Persoon {
};

#endif
```

Bijna goed!

Aggregatie (... heeft een ...)



Fiets heeft:

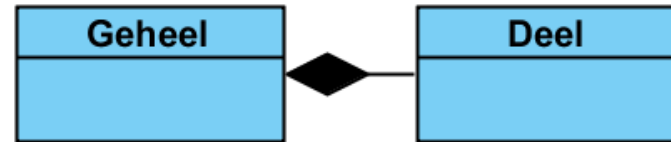
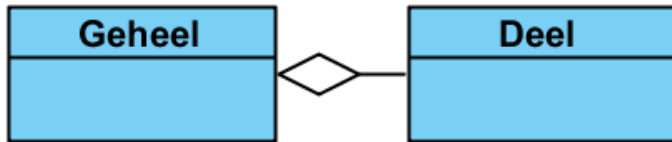
- 2 Wielen
- 1 Frame
- ...

- Overerving
- Aggregatie
- **Compositie**



Beginnende UML modelleerders gebruiken vaak ten onrechte compositie!

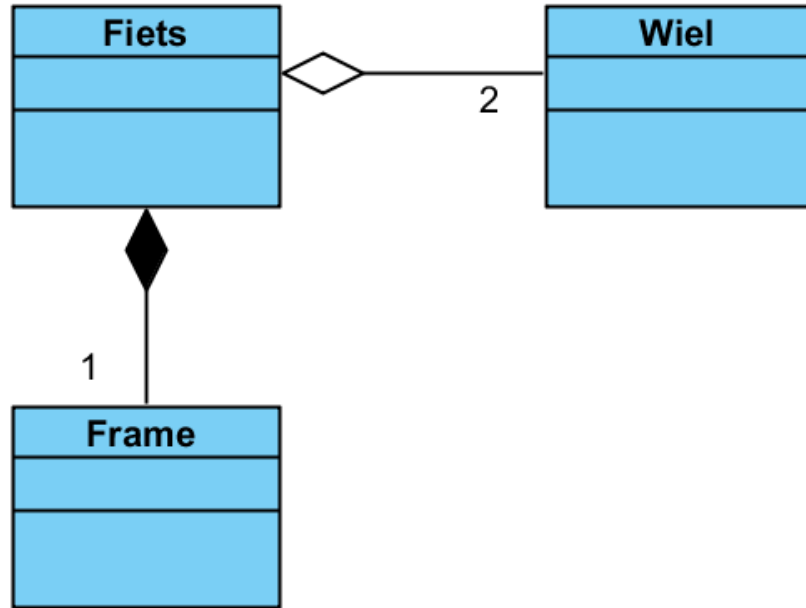
Een compositie is een **speciaal soort** aggregatie.



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

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

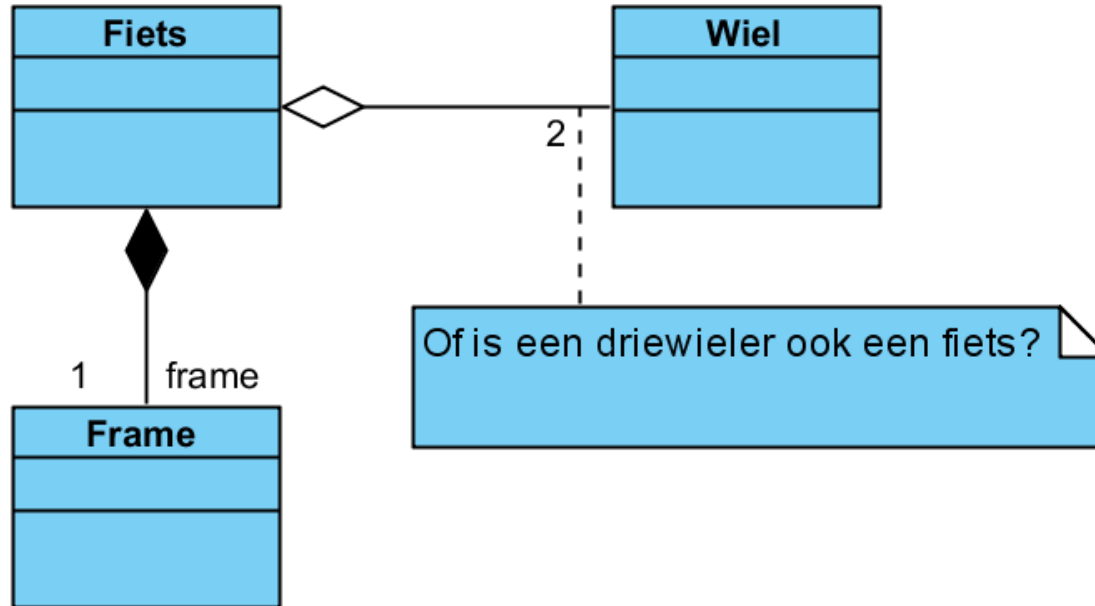
Compositie (... bevat een ...)



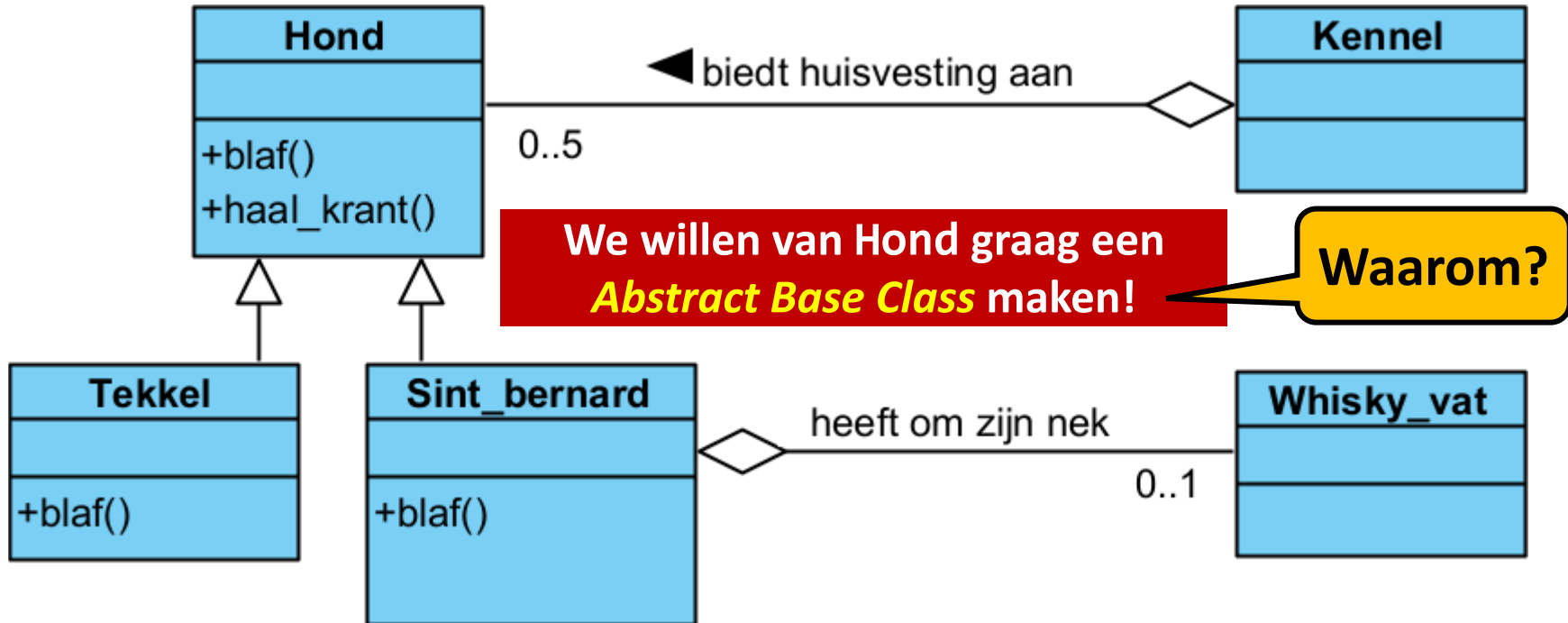
Welke **aggregaties** zijn **composities**?

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

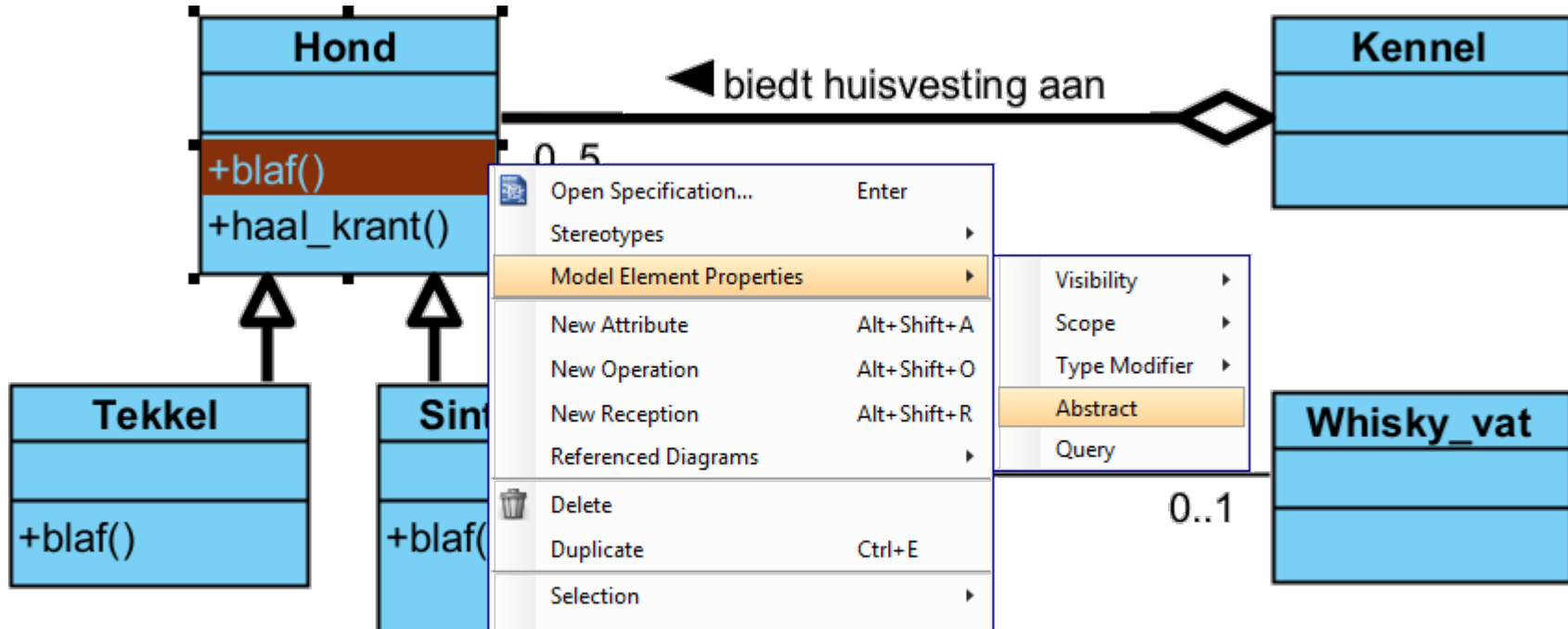
Let op! Het goede antwoord is afhankelijk van de applicatie.



Voorbeeld

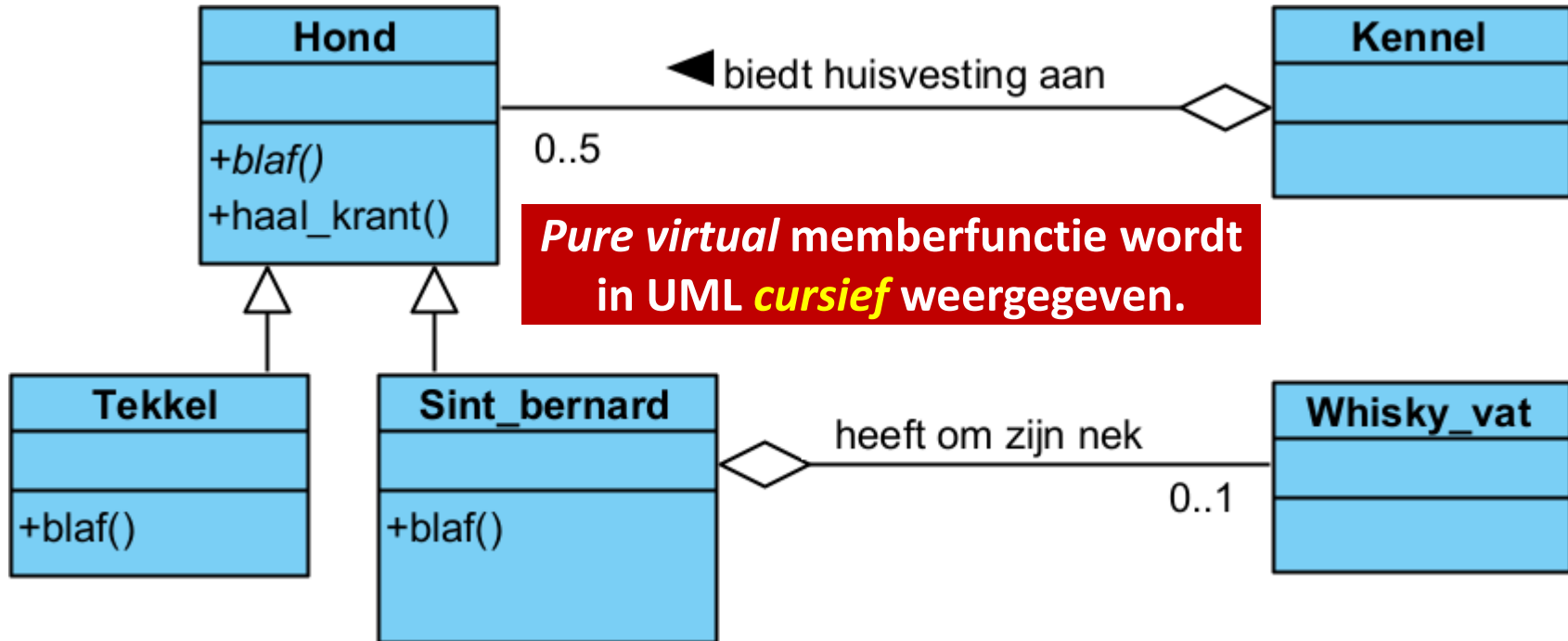


Pure virtual member function

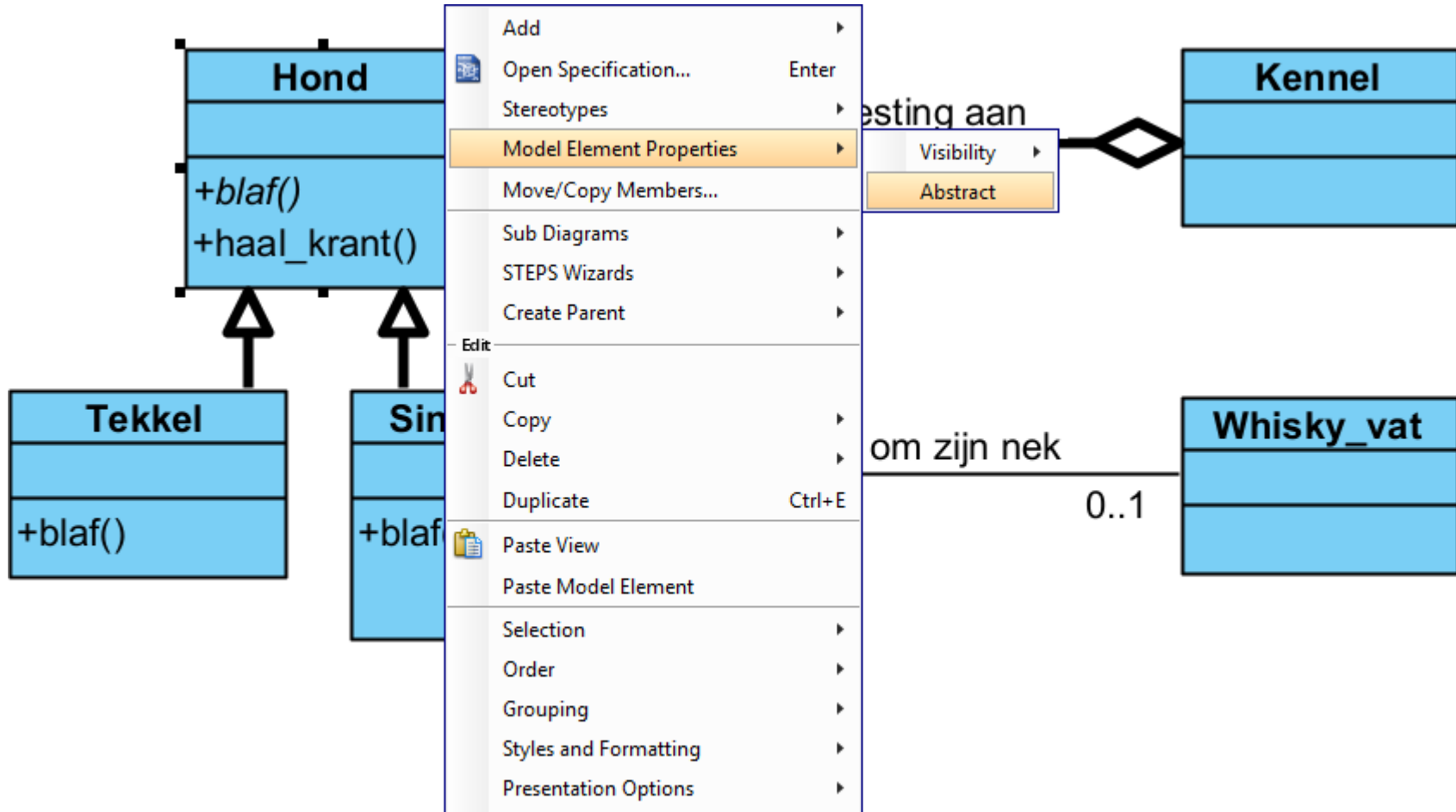


Abstract geeft aan dat het een **Pure virtual** memberfunctie wordt deze dus **moet** worden geïmplementeerd in een afgeleide klasse.

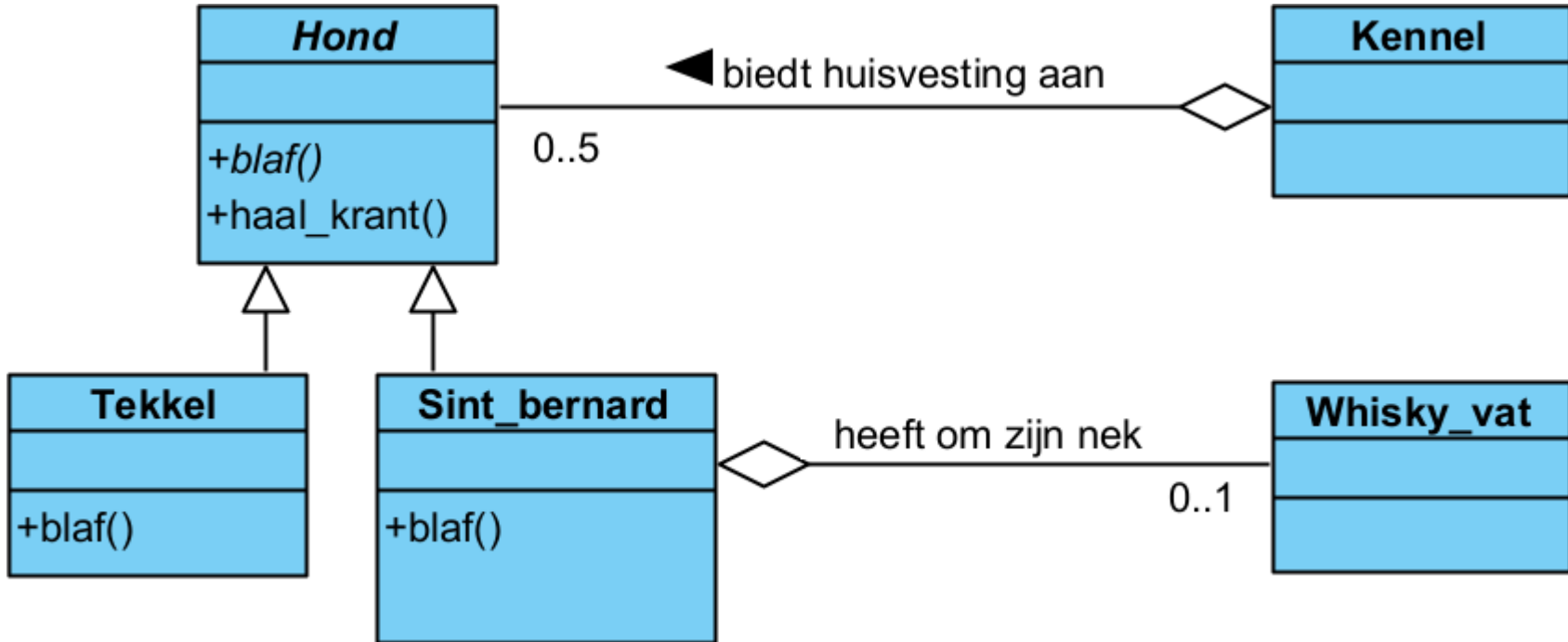
Pure virtual member function



Abstract Base Class

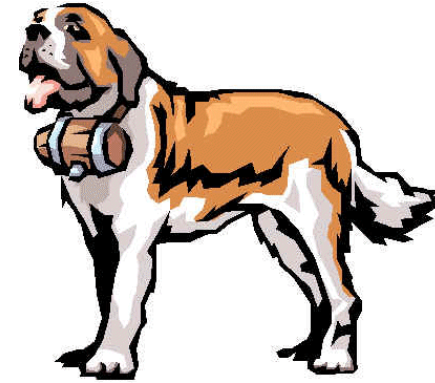
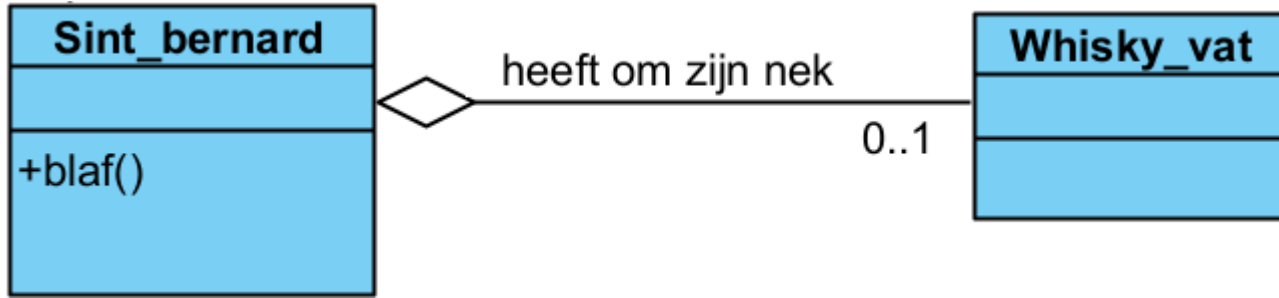


Abstract Base Class



Abstract Base Class wordt ook *cursief* weergegeven.

Aggregatie implementatie



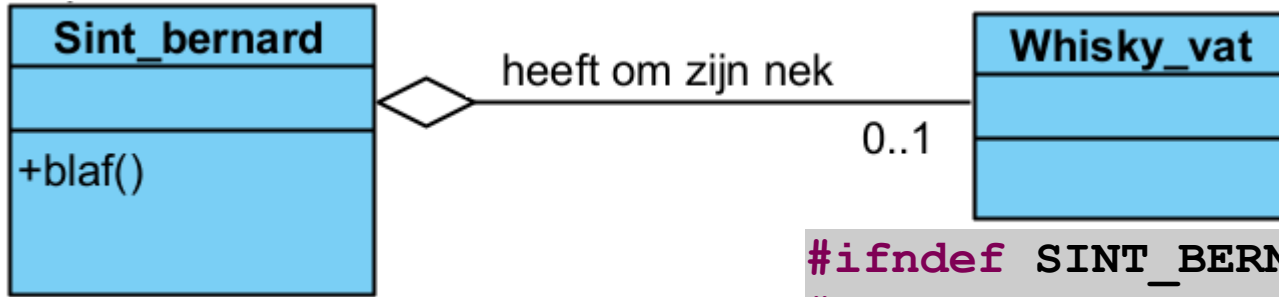
Hoe kunnen we een **0..1** aggregatie implementeren in class SintBernard?

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

Niet **0..1** maar **1**.

Andere naam voor een **Whisky_vat** data member al bestaat (?)

Aggregatie implementatie



- Open Specification...
- Add Shape
- Rename...
- Synchronize to Entity Relationship Diagram
- Ignore Classes when Synchronizing...
- Paste View
- Paste Model Element
- Handi-Selection
- Diagram Content
- Connectors
- Presentation Options
- Layers...
- Reference Mapping
- Zoom
- Layout
- Select in Tree
- Show Link...
- Utilities

- Apply Design Pattern...
- Transformation
- Visual Diff...
- Create Matrix Diagram...
- Transit to New Diagram...
- Java Round-trip
- C++ Round-trip
 - Generate Code
 - Reverse Code
- Generate REST API...
- Generate OpenAPI/Swagger...
- Generate API Blueprint...
- Synchronize Classes Description to ERD...

```
#ifndef SINT_BERNARD_H
#define SINT_BERNARD_H

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

#endif
```

Waar is de Whisky?

Aggregatie implementatie

The diagram shows two classes: **Sint_bernard** and **Whisky_vat**. **Sint_bernard** has a method `+blaf()`. An aggregation relationship, indicated by a hollow diamond on the **Sint_bernard** side, is labeled "heeft om zijn nek" and has a multiplicity of `0..1` at the **Whisky_vat** end. A context menu is open over **Whisky_vat**, and the "Association End Specification" dialog is shown with the following settings:

Field	Value
Role	vat
Referenced attribute	<Unspecified>
Multiplicity	<Unspecified>
Visibility	private
Navigable	True
Aggregation kind	None
Type	Whisky_vat
Type modifier	*
Owned by	— heeft om zijn nek (Sint_bernard -> Whisky_vat)

Aggregatie implementatie



Bijna goed!

- Open Specification...
- Add Shape
- Rename...
- Synchronize to Entity Relationship Diagram
- Ignore Classes when Synchronizing...
- Paste View
- Paste Model Element
- Handi-Selection
- Diagram Content
- Connectors
- Presentation Options
- Layers...
- Reference Mapping
- Zoom
- Layout
- Select in Tree
- Show Link...
- Utilities

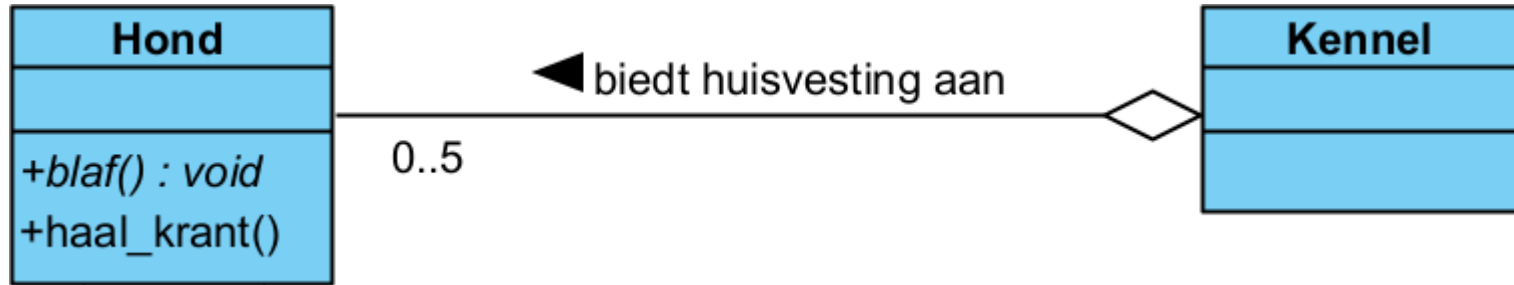
- Apply Design Pattern...
- Transformation
- Visual Diff...
- Create Matrix Diagram...
- Transit to New Diagram...
- Java Round-trip
- C++ Round-trip
- Generate REST API...
- Generate OpenAPI/Swagger...
- Generate API Blueprint...
- Synchronize Classes Description to ERD.

```
#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



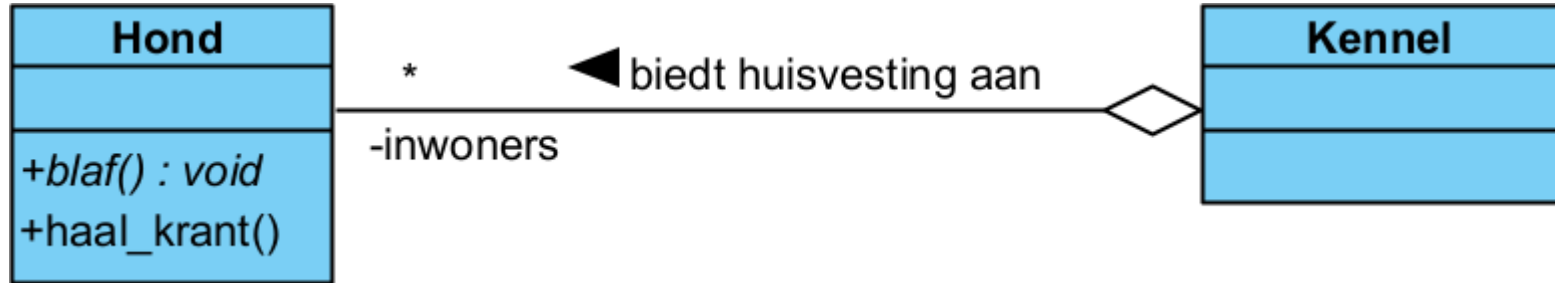
Hoe kunnen we een **0..5** aggregatie implementeren in class Kennel?

- 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



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

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

#endif
```


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 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!

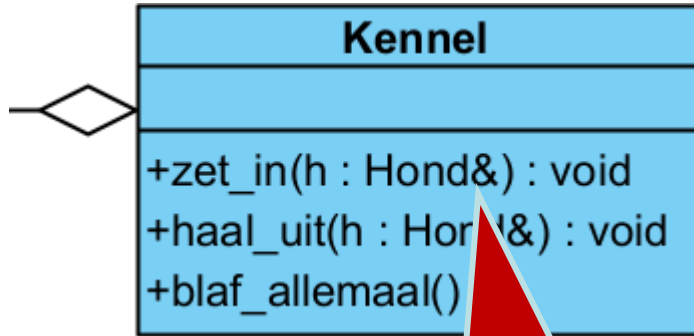
Aan de slag met [Eindopdracht_2.pdf](#)



Schrijf een testprogramma voor de hiervoor gemaakte class `Kenne1`.

- Zet een `Sint_bernard` genaamd `boris` in de `Kenne1` genaamd `k`.
- Zet een `Tekke1` genaamd `harry` in de `Kenne1` `k`.
- Zet een `Sint_bernard` genaamd `felix` in de `Kenne1` `k`.
- Laat alle honden in de `Kenne1` `k` blaffen.
- Haal `harry` uit de `Kenne1` `k`.
- Laat alle honden in de `Kenne1` `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
```

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 << " ";
    }
}
```