Design patterns

W. Oele

1 juli 2014

Onderwerpen

- U.M.L.
- Ontwerp patronen:
  - adapter en facade
  - bridge
  - abstract factory
  - decorator
  - observer
  - template method
  - ...

Leermiddelen

- boek: Design patterns explained (Shalloway/Trott)

Toetsing

- Vier uitgebreide practicum opdrachten
- Uitwerken in groepjes van 4
- Eindcijfer is gemiddelde van de opdrachten

Ontwerpen

Tot nu toe werd gegeven:

Ontwerpen

Tot nu toe werd gegeven:

- het vraagstuk in natuurlijke taal

Ontwerpen

Tot nu toe werd gegeven:

- het vraagstuk in natuurlijke taal
- aanwijzingen voor het schrijven van het programma:
  - welke classes
  - fields, constructoren en methodes
  - algehele technische werking van een programma

Ontwerpen

In werkelijkheid:

Ontwerpen

In werkelijkheid:

- vraagstuk doorgaans alleen in natuurlijke taal gegeven
- al het overige ontbreekt

Ontwerpen

In werkelijkheid:

- vraagstuk doorgaans alleen in natuurlijke taal gegeven
- al het overige ontbreekt

Vraag: Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

Ontwerpen

Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

Ontwerpen

Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

- welke classes en interfaces te schrijven?

Ontwerpen

Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

- welke classes en interfaces te schrijven?
- hoe werken deze samen?

Ontwerpen

Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

- welke classes en interfaces te schrijven?
- hoe werken deze samen?
- welke fields, methodes en constructoren bevatten ze?

Ontwerpen

Hoe een gegeven vraagstuk te vertalen naar een werkend programma?

- welke classes en interfaces te schrijven?
- hoe werken deze samen?
- welke fields, methodes en constructoren bevatten ze?

Bovenstaande vragen betreffen het *ontwerpen* van een programma.

Een klassiek model...

Stappen:

Requirements ↓	eisen waaraan een programma moet voldoen
Technisch ontwerp ↓	classes en hun inhoud
Implementatie ↓	het daadwerkelijke coderen
Testen ↓	
Onderhoud	aanpassen aan nieuwere standaards, uitbreiding van functionaliteit

Requirements

Requirements:

- vaak niet compleet

Requirements

Requirements:

- vaak niet compleet
- met enige regelmaat fout

Requirements

Requirements:

- vaak niet compleet
- met enige regelmaat fout
- onduidelijk/vaag of zelfs misleidend

Requirements

Requirements:

- vaak niet compleet
- met enige regelmaat fout
- onduidelijk/vaag of zelfs misleidend
- bevatten soms impliciete (en niet altijd juiste) aannames

Requirements

Requirements:

- vaak niet compleet
- met enige regelmaat fout
- onduidelijk/vaag of zelfs misleidend
- bevatten soms impliciete (en niet altijd juiste) aannames

Derhalve: behoefte aan ontwerptechnieken die rekening houden met het bovenstaande...

Ontwerpen

Model:

- probleem analyseren
- probleem oplossen

Probleem analyseren

- Niet nadenken in termen van oplossingen
- Probleem beschrijven in het jargon van het probleem
- Decompositie: groot probleem opdelen in kleinere, behapbare, stukken
- Niet denken in termen van classes, methods, etc.
- Denken in termen van *entiteiten*:
- Wie is waarvoor verantwoordelijk?

Ontwerpen

Enkele algemene richtlijnen:

- Classes/objecten een duidelijk afgebakende verantwoordelijkheid geven
- Encapsulatie: verbergen van implementatie voor buitenwereld

Ontwerpen: cohesion vs. coupling

High cohesion:

- nauwe verbondenheid van functies/methodes/onderdelen

Loose coupling:

- losse koppeling van onderdelen

Voorbeeld: de autoradio

Duidelijk afgebakende functies en verantwoordelijkheden:

- muziek afspelen
- radio zenders ontvangen

Voorbeeld: de autoradio

Duidelijk afgebakende functies en verantwoordelijkheden:

- muziek afspelen
- radio zenders ontvangen

High cohesion:

- radio bestaat uit honderden onderdelen die allemaal nodig zijn
- onderdelen werken nauw samen om een autoradio te zijn in een groter geheel

Voorbeeld: de autoradio

Duidelijk afgebakende functies en verantwoordelijkheden:

- muziek afspelen
- radio zenders ontvangen

High cohesion:

- radio bestaat uit honderden onderdelen die allemaal nodig zijn
- onderdelen werken nauw samen om een autoradio te zijn in een groter geheel

Encapsulatie:

- interne werking van radio onbekend voor de buitenwereld

Voorbeeld: de autoradio

Loose coupling:

- weinig connecties met andere onderdelen van de auto

Gevolg:

- makkelijk te vervangen
- geen onbegrijpelijke afhankelijkheden
- auto makkelijker te onderhouden

Unified Modelling Language

- visuele taal
- eigen regels
- eigen syntax
- de facto standaard voor o.o.p.

Unified Modelling Language

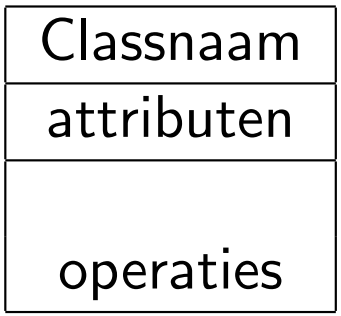
- visuele taal
- eigen regels
- eigen syntax
- de facto standaard voor o.o.p.

U.M.L. is niet:

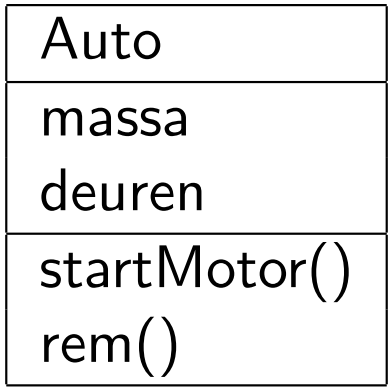
- een programmeertaal (u.m.l. produceert een raamwerk, geen algoritmen)
- een modeleringstool
- een ontwikkel methode
- uitontwikkeld

De class

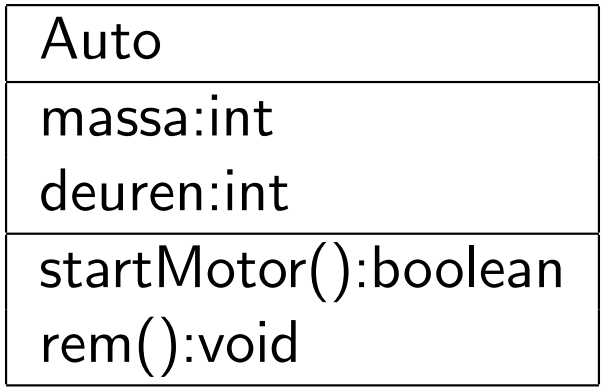
Een klasse representeert men met een rechthoek:



Voorbeeld



Voorbeeld met typen



Attributen

- attributen vormen de fields in een class
- kunnen van elk denkbaar type zijn:
  - int
  - float
  - etc.
  - pointers naar andere classes

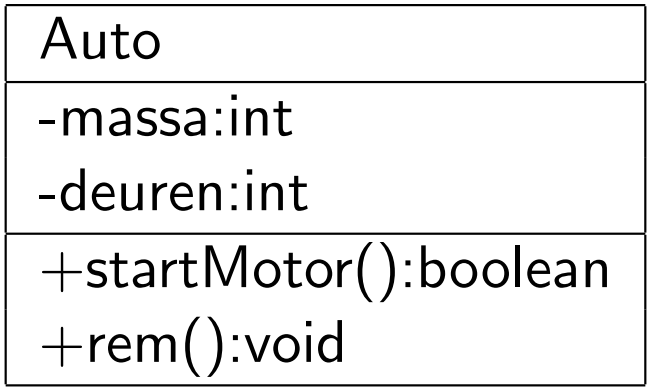
Operaties

- vormen de methodes/functies in een class
- kunnen parameters hebben
- kunnen parameters teruggeven

Modifiers

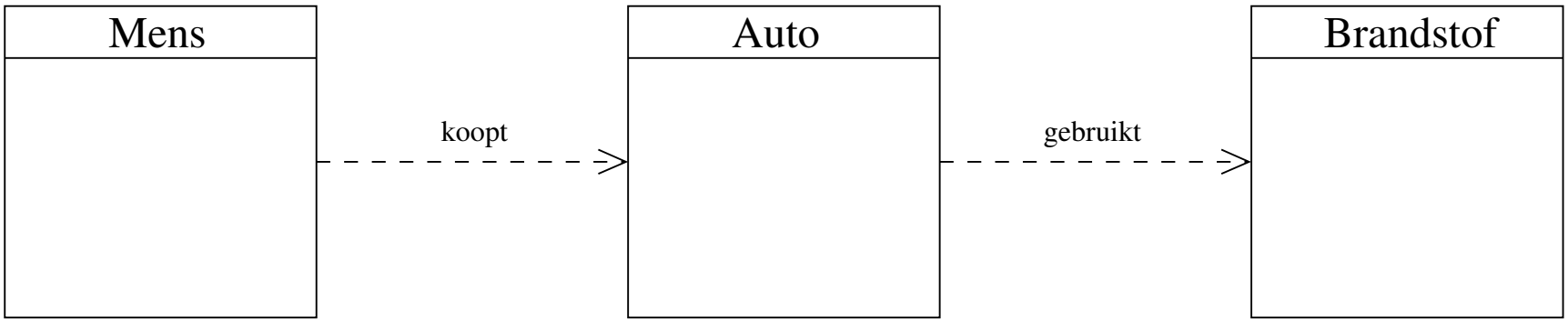
- - : private access
- + : public access
- # : protected access
- ~: beschikbaar in dezelfde package

Voorbeeld



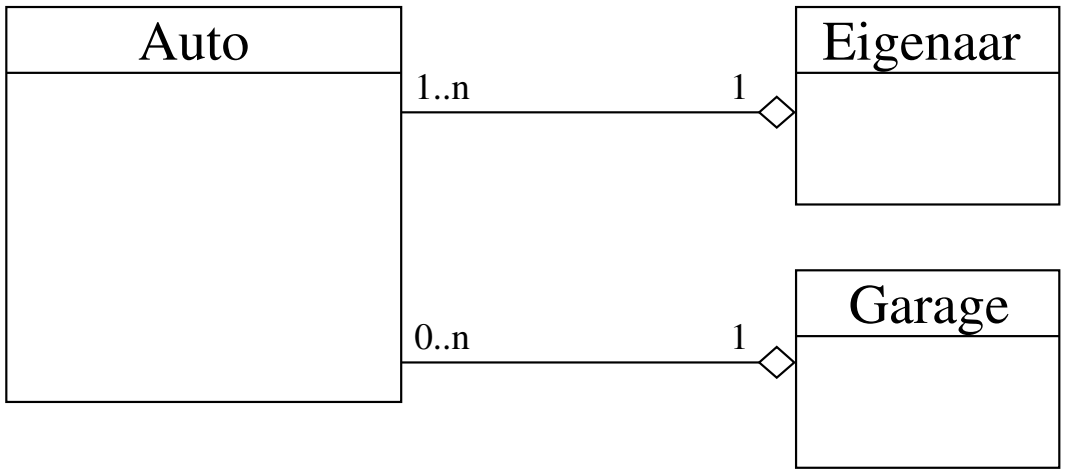
Associatie

- Twee objecten kunnen iets met elkaar te maken hebben. Dit wordt genoteerd middels een stippellijn:
- Het soort associatie kan men er in tekst bijzetten.



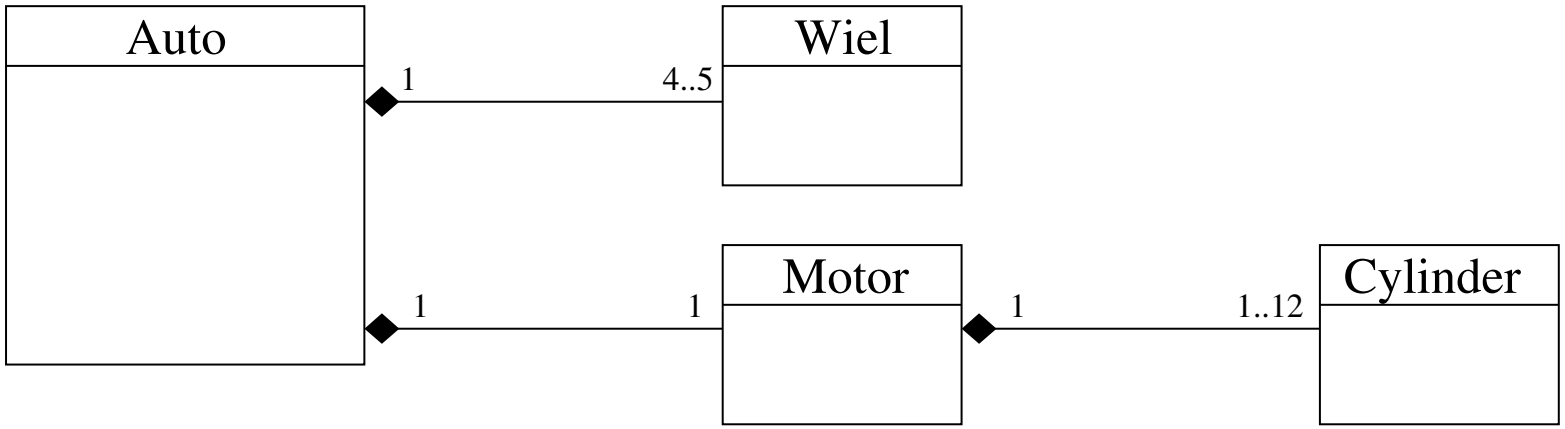
Aggregatie

De ene klasse kan als attribuut een andere klasse hebben.



- Een eigenaar heeft 1 of meerdere auto' s.
- Een auto heeft één eigenaar.

Compositie



Bond minicar 1949
Pullman 1905

Compositie vs. aggregatie

Compositie:

- dichte diamanten
- het ene object kan niet bestaan zonder het andere.
- het ene obect is een onderdeel van het andere
- voorbeeld: huis$\leftrightarrow$muur

Compositie vs. aggregatie

Compositie:

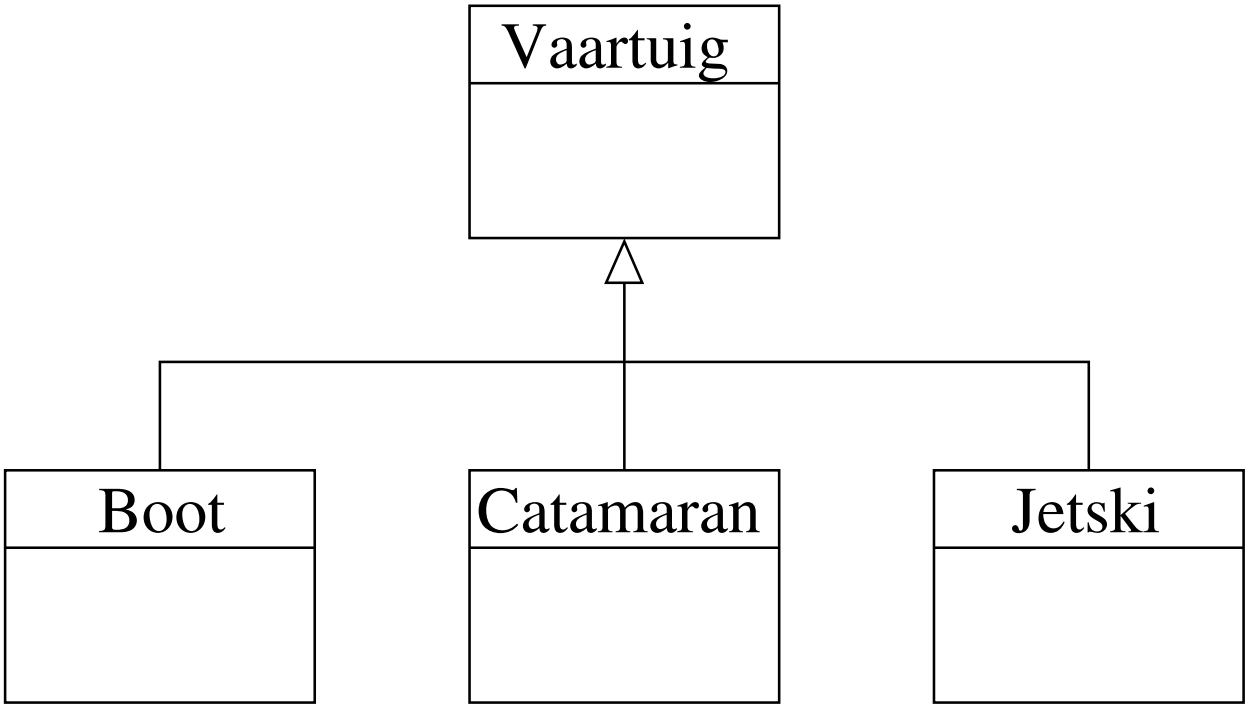
- dichte diamanten
- het ene object kan niet bestaan zonder het andere.
- het ene obect is een onderdeel van het andere
- voorbeeld: huis↔muur

Aggregatie:

- open diamanten
- het ene object kan bestaan zonder het andere
- voorbeeld: vereniging↔leden

Subtyping

Class deriving van een concrete parent class.



Subtyping

Class deriving van een abstracte parent class.