Methods of Development

College 3
“Object-georiënteerd programmeren”

Arjan Scherpenisse
arjan.scherpenisse@kmt.hku.nl
@acscherp

Deze week

- Tussenstand opdracht 2
- Pseudocode
  - Herhaling vorige week
  - Functies
- Object-oriëntatie

Tussenstand opdracht

- 4 mensen hebben een game ingeleverd
- Heel aantal mensen heeft ook pseudocode gestuurd

Literatuurlijst

- Software studies *(Matthew Fuller)*
- Code Complete
  (2nd Ed) by Steve McConnell
- Software engineering *(Ian Sommerville)*
- The Pragmatic Programmer *(Andrew Hunt & David Thomas)*

Pseudocode

- “Pseudocode wordt gebruikt om *algoritmen* vast te leggen op een door mensen leesbare manier met behoud van de stappen.” (wikipedia)

Pseudocode

- Statements
- If-then-else
- Loops
- Variabelen
- Condities
- Functies

Pseudocode herhaling

- Statements, al dan niet *geparametrizeerd*
  - “Verwarm de oven voor”
  - “Verwarm de oven voor op X graden”
- Vertakkingen (branches)
  - IF … THEN … (ELSE …)
- Herhalingen (loops)
  - REPEAT … UNTIL …
- Condities: “checks” waar / niet waar

Vertakkingen (branches)

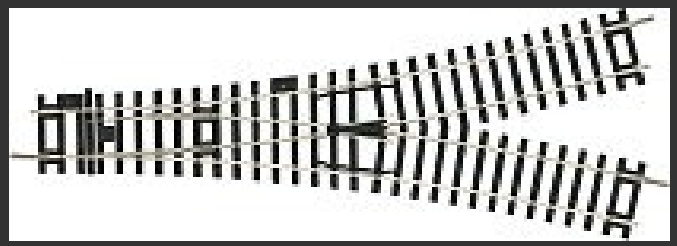
- Oorzaak → gevolg: “Als dit, dan dat”
- IF iets, THEN doewat, (ELSE doewatanders)

Expressie

Statement(s)

Statement(s)

- IF oventemperatuur te hoog
  THEN stop met voorverwarmen

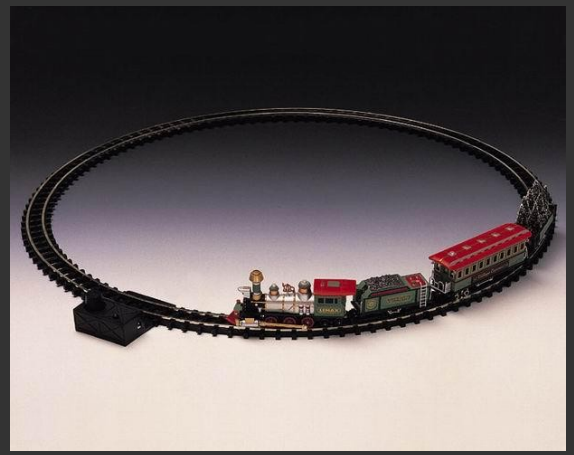


Herhalingen (Loops)

- “Doe hetzelfde, totdat er iets geldig is”
- REPEAT doewat UNTIL iets *(geldig is)*

Expressie

Statement(s)



Variabelen

- Een “vakje” waar iets in kan worden bewaard

```
SET gewensteTemperatuur TO 220
activeer oven
REPEAT
    SET huidigeTemperatuur TO de huidige oventemperatuur
UNTIL huidigeTemperatuur >= gewensteTemperatuur
```

~~Expressies~~ Condities

- "huidigeTemperatuur GROTER DAN gewensteTemperatuur"
  → is een *conditie*
- Expressies zijn algemener dan condities
- Niet perse waar/niet waar, maar ook bv.
  - *Maximum levens* - *Levens verbruikt*
  - lowercase(player name)
- **Condities** zijn expressies die waar/niet waar (true/false) als resultaat hebben

Logische blokken

- Door code te “indenten” krijg je inzicht in welke stukken bij elkaar horen

```
IF
WekkerTijd kleiner dan 9 uur
THEN
Wekker op snooze
REPEAT
Slapen
UNTIL
WekkerTijd 9 uur
END
Sta eindelijk op
```

```
IF  WekkerTijd kleiner dan 9 uur THEN
    Wekker op snooze
    REPEAT
        Slapen
    UNTIL
    WekkerTijd 9 uur
END
Sta eindelijk op
```

2 ”geneste” niveaus

- “Verwarm de oven op X graden”
- Dat verwarmen zelf heeft ook weer een stukje pseudocode...!
- ...*sub*-pseudocode?
- X is hier een *parameter*

Functies

- Statement: voorverwarmen(220)
- Implementatie:

```
FUNCTION voorverwarmen(gewensteTemperatuur)
  activeerOven()
  REPEAT
    SET temperatuur TO getOvenTemperatuur()
  UNTIL temperatuur >= gewensteTemperatuur
  RETURN temperatuur
ENDFUNCTION
```

Da's ook weer een functie!

Yup.

Functies

- In sommige talen heten dat *subroutines*
- Don't Repeat Yourself! (D.R.Y.)
- Maken het herbruiken van code makkelijker
- Verstoppen de onnodige details
- Functienaam is vaak een werkwoord: het doet iets

Anatomie van een functie

Verwarm oven voor tot een temperatuur. Returnt
de huidige temperatuur van de oven.

Functiedeclaratie of prototype

```
FUNCTION voorverwarmen(gewensteTemperatuur)

    activeerOven()
    REPEAT
      SET temperatuur TO getOvenTemperatuur()
    UNTIL temperatuur >= gewensteTemperatuur
    RETURN temperatuur

ENDFUNCTION
```

Functie-body

Return-statement

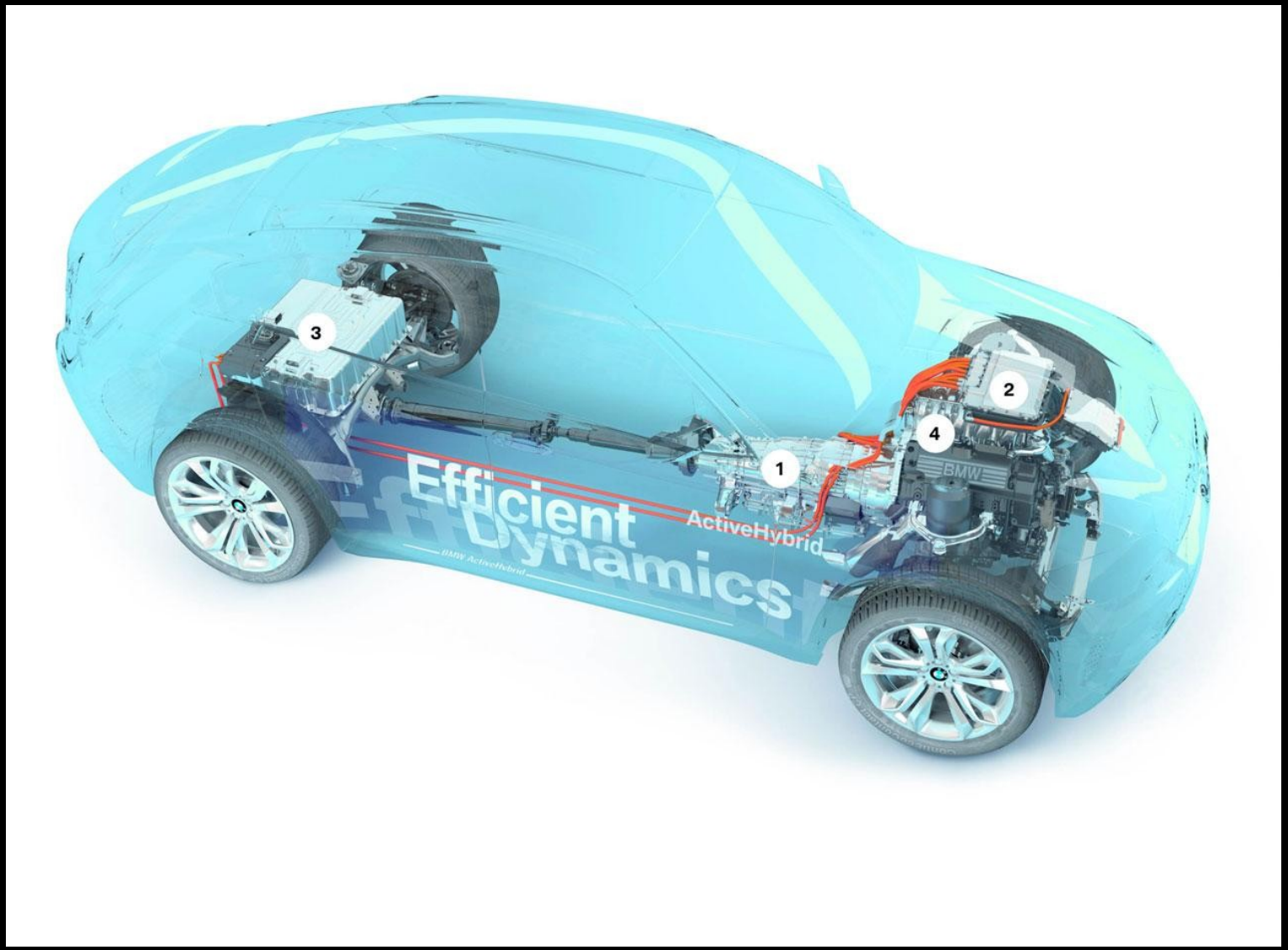
Onderdelen v.e. functie

- Declaratie / prototype
  - Hoe heet de functie
  - Welke parameters gaan er in
- Functie body
  - De *implementatie*: hier gebeurt het werk
- Return statement
  - Om uit de functie te springen met een value


SCRATCH

Objecten

- Logische groepering van functionaliteit
  - Eigen data (attributen)
  - Eigen *methodes* (functies)
- Begrippen:
  - Interface
  - Encapsulatie / data hiding
  - Overerving


3
2
4
1
Efficient
Dynamics
ActiveHybrid
BMW

Encapsulatie

- Omdat je niet alles van een object hoeft te weten
- De interface van de class bepaalt het gedrag
- Voordelen:
  - Je interface documenteert zichzelf
  - Later makkelijker aan te passen (refactoring)
  - Data op 1 plek


MENTAT
OPTIONS
Credits
699
Ordos Trooper.
TROOPER
DMG
ATTACK
MOVE
RETREAT
GUARD

De Interface van een object

Class TROOPER

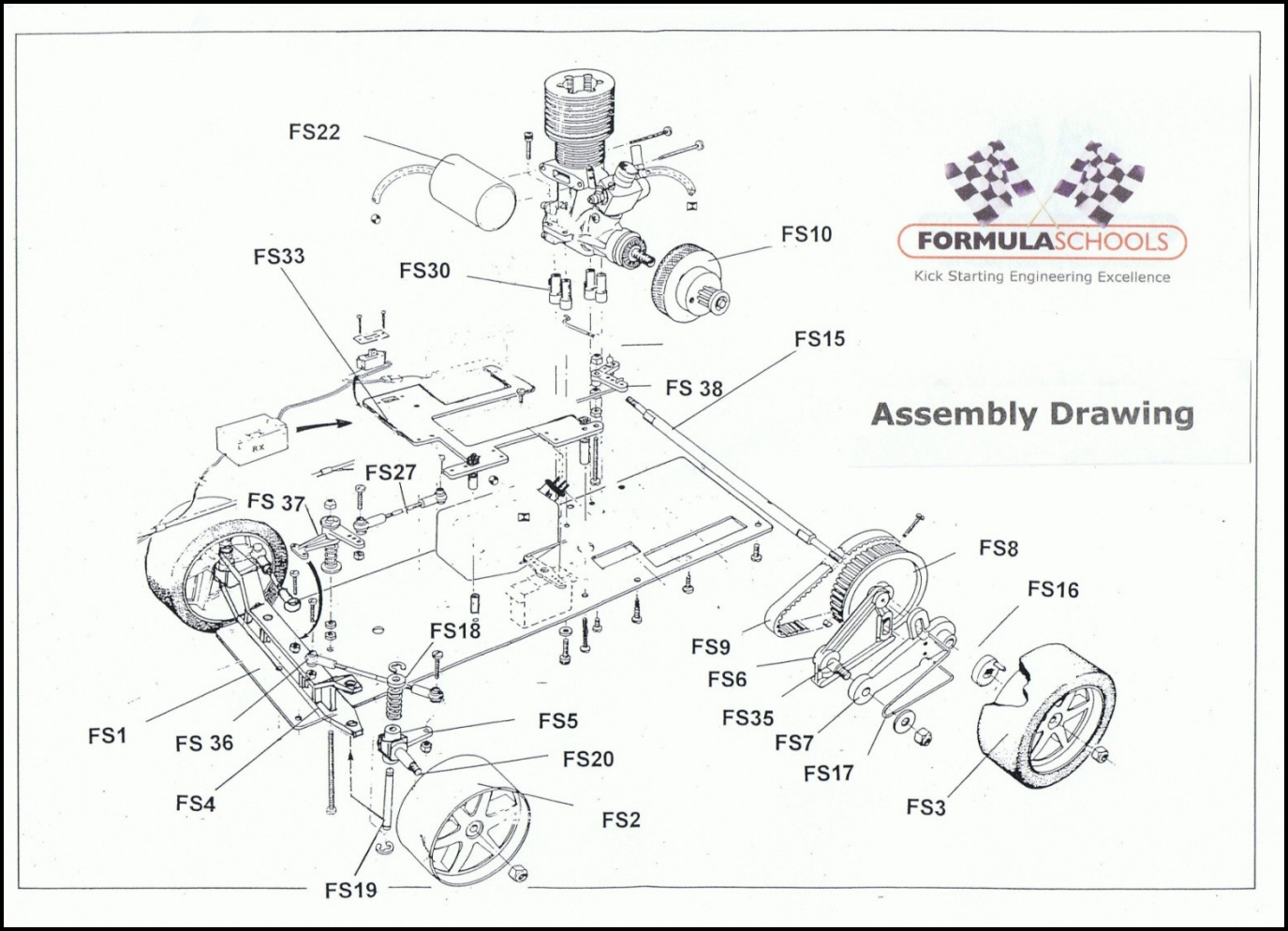
Attributen:

- naam
- prijs
- positie
- health

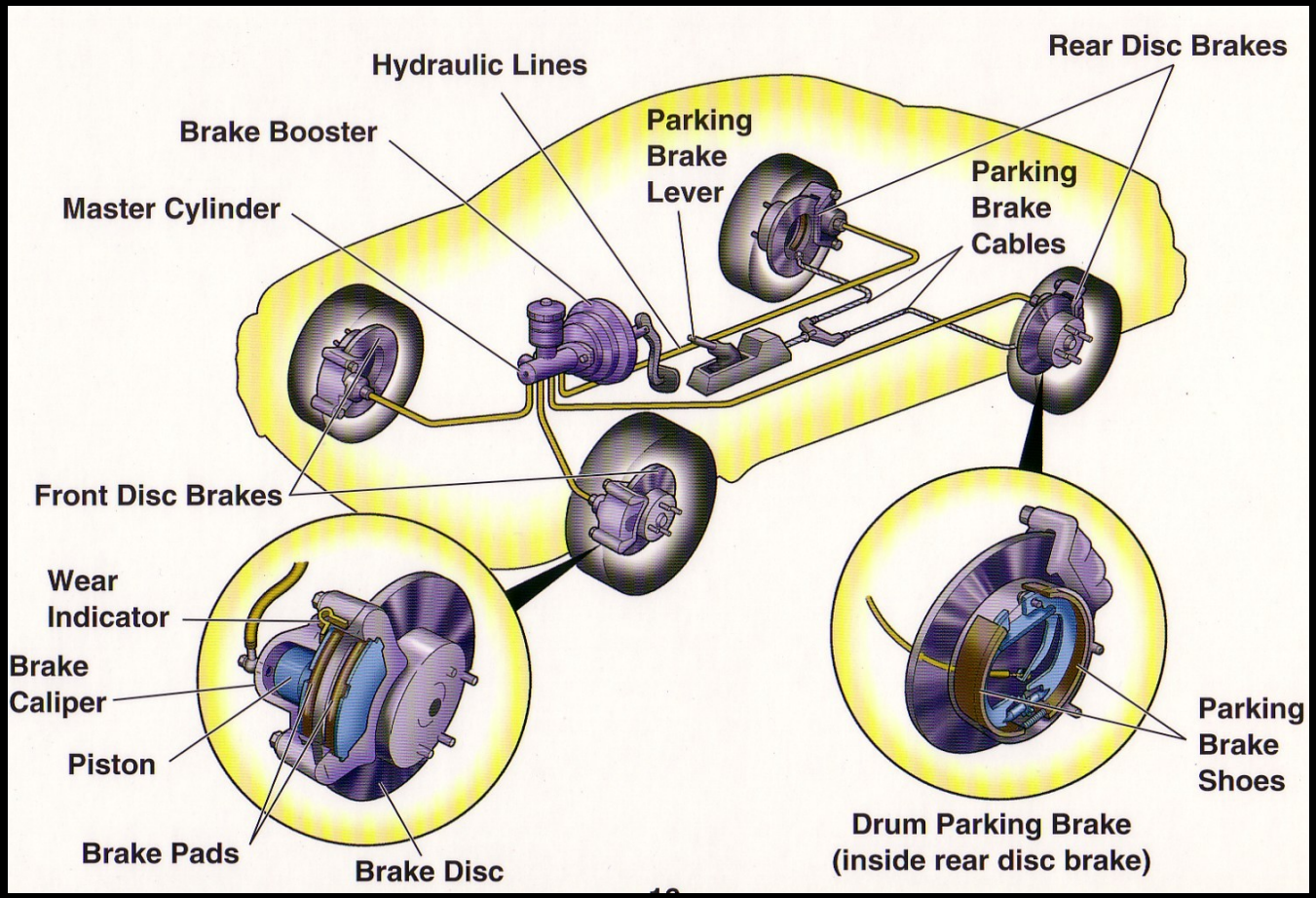
Methoden:

+ attack()
+ move()
+ retreat()
+ guard()

Attributen


FS22
FS10
FS33
FS30
FS15
FS 38
FS27
FS 37
FS8
FS16
FS18
FS9
FS6
FS35
FS1
FS 36
FS5
FS20
FS7
FS17
FS4
FS2
FS3
FS19
FORMULASCHOOLS
Kick Starting Engineering Excellence
Assembly Drawing

Methoden


Hydraulic Lines
Rear Disc Brakes
Brake Booster
Parking Brake Lever
Master Cylinder
Parking Brake Cables
Front Disc Brakes
Wear Indicator
Brake Caliper
Piston
Brake Pads
Brake Disc
Parking Brake Shoes
Drum Parking Brake (inside rear disc brake)

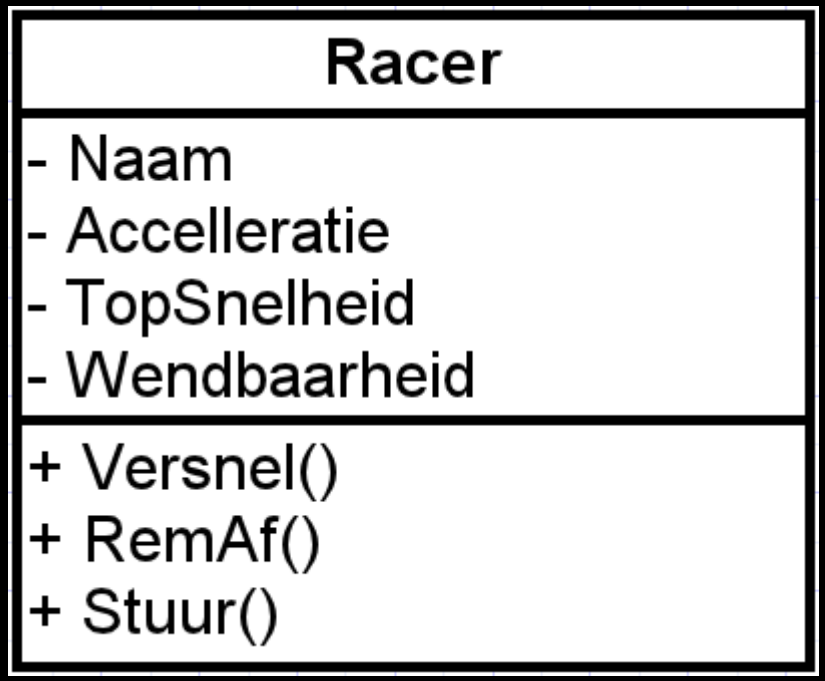
Een object in pseudocode

CLASS Racer

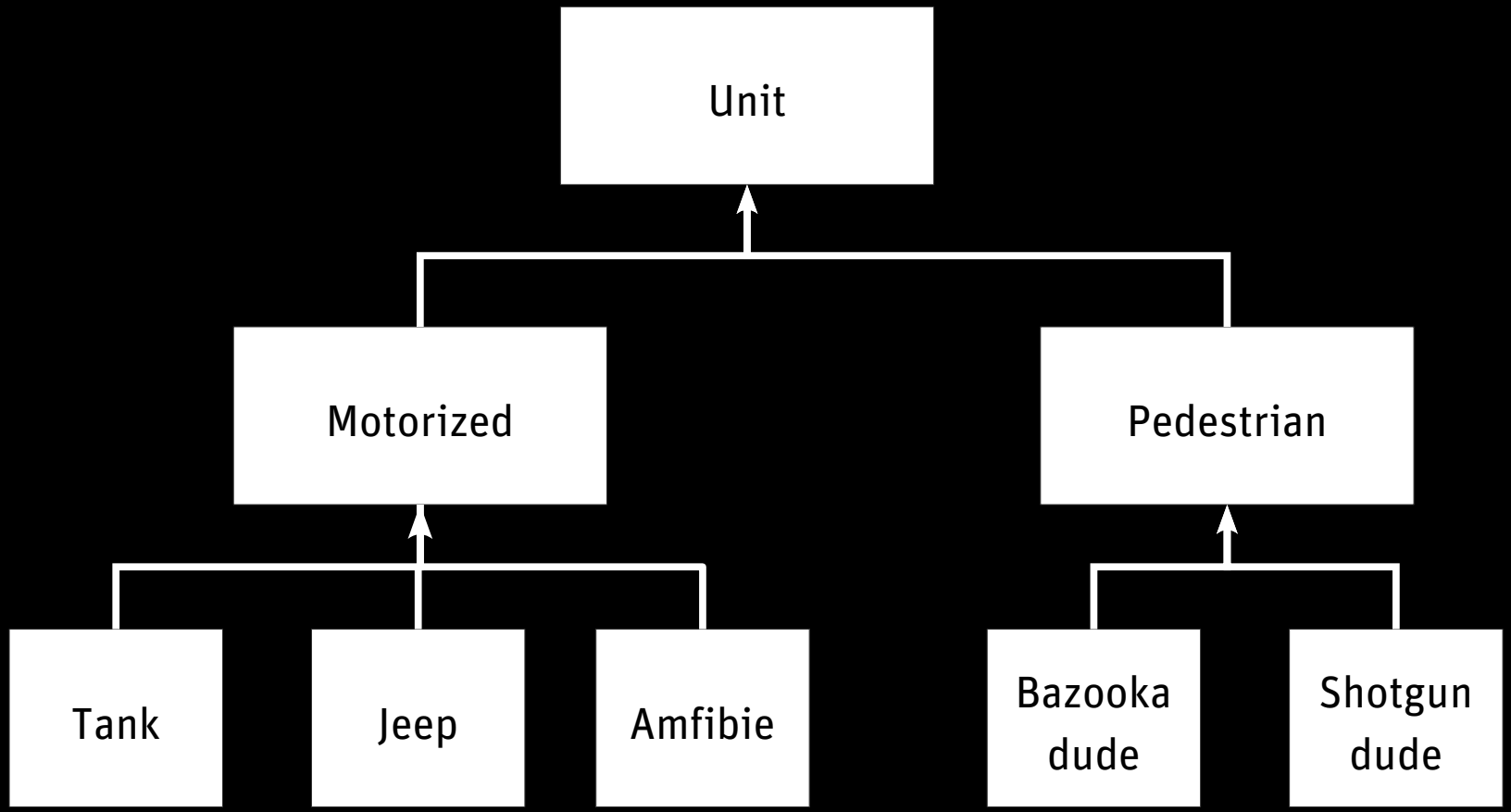
ATTRIBUTES
- Naam
- Acceleratie
- TopSnelheid
- Wendbaarheid

METHODS
- Versnel()
- Rem()
- Stuur()

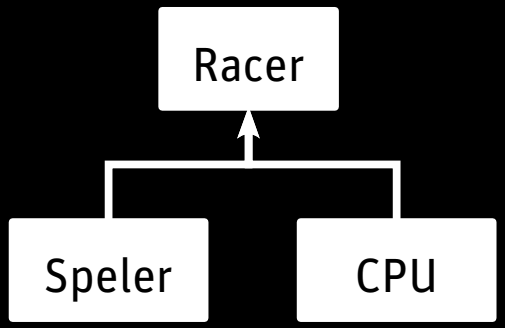
...en in UML



Overerving (inheritance)



CLASS Racer
ATTRIBUTES
Naam
Acceleratie
TopSnelheid
Wendbaarheid
METHODS
Versnel()
Rem()
Stuur()

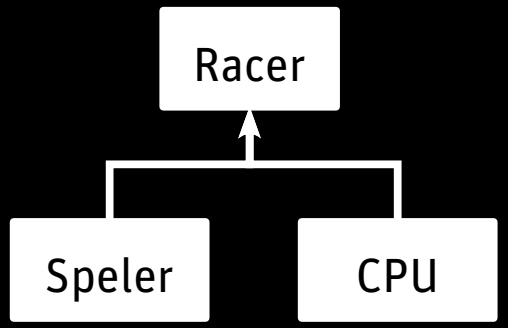


CLASS Speler **EXTENDS** Racer
ATTRIBUTES
ControllerNummer
PowerUp
AantalMuntjes

CLASS CPU **EXTENDS** Racer
ATTRIBUTES
AI_Patroon

Overerving

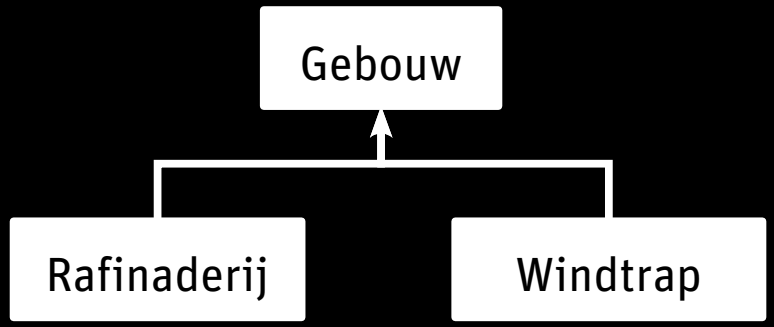
- Omdat classes vaak dingen met elkaar gemeen hebben
- Code delen tussen classes
- Subclasses en superclasses
  - Ook wel: parent class vs. child class
- Naamgeving afh. van hoe je het bekijkt
  - CPU heeft Racer als superclass (en speler ook)
  - Racer heeft CPU en Speler als subclasses



CLASS Gebouw
ATTRIBUTES
Naam
Prijs
HitPoints
Stroomkosten
METHODS
Bouw()
Repareer()

CLASS Windtrap **EXTENDS** Gebouw
ATTRIBUTES
StroomCapaciteit
METHODS
WinStroom()

CLASS Rafinaderij **EXTENDS** Gebouw
ATTRIBUTES
SpecieCapaciteit
AantalHarvesters
METHODS
VerzamelSpecie()



“Class” vs. “Object” ???

- Een class is een definitie
  - Er is maar 1 definitie
- Een object is een instantie van een class
  - Er kunnen meerdere instanties zijn
    (meerdere dezelfde units, allemaal met verschillende health, positie e.d.)
  - $mijnUnit = new Trooper(“Piet”)

object

class


000
000
000
000

Oefenopdracht

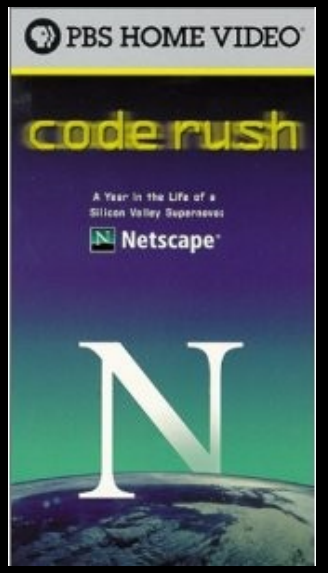
- Bedenk hoe het class diagram van je favoriete game eruit zou zien. Gebruik niet meer dan 5 classes, en minimaal 1 maal inheritance. Beschrijf de attributen en methodes van de classes.

  Opdracht niet verplicht, maar wel nuttig
  → tentamenstof! Mailen == feedback.

Zijn we er nog?

Popcornmateriaal

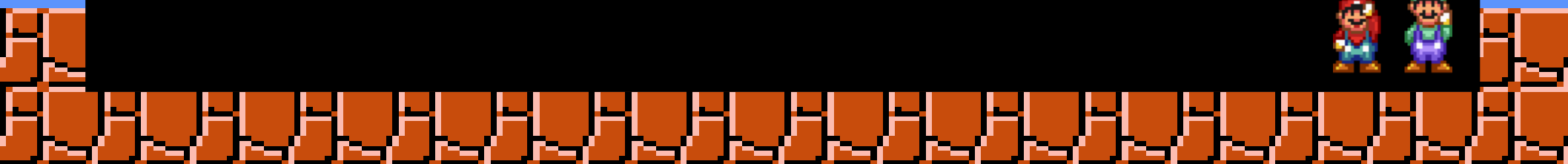
- Minecraft documentaire
- Code Rush (over opensourcen van Netscape)

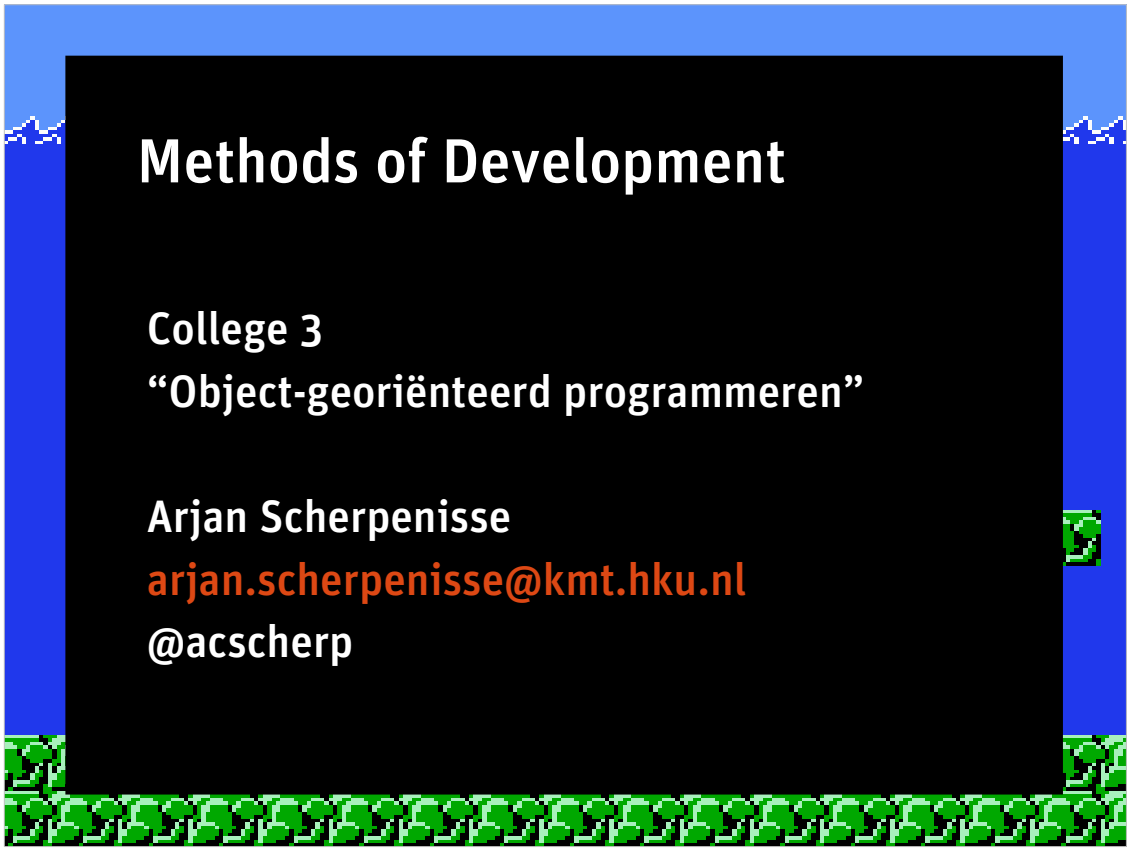


Komende colleges

- ~~College 1: waar hebben we het over~~
- ~~College 2: imperatief programmeren~~
- ~~College 3: object-orientatie~~
- College 4: vervolg OO, design patterns, Q&A
- Tentamen (begin december)

Tot volgende week!




Methods of Development
College 3
“Object-georiënteerd programmeren”
Arjan Scherpenisse
arjan.scherpenisse@kmt.hku.nl
@acscherp


Deze week
Tussenstand opdracht 2
Pseudocode
Herhaling vorige week
Functies
Object-oriëntatie


Tussenstand opdracht
4 mensen hebben een game ingeleverd
Heel aantal mensen heeft ook pseudocode gestuurd

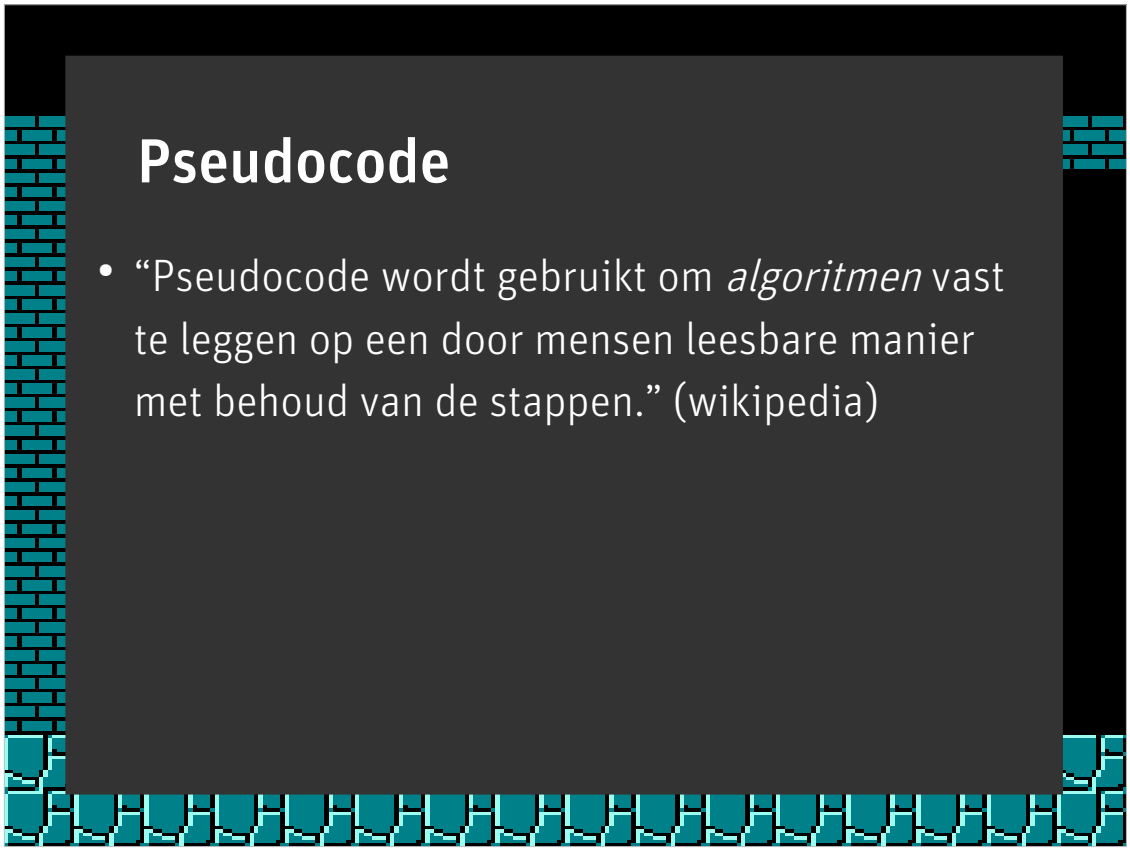
Deze lijst is optioneel, en bevatten geen tentamenstof.
Maar het zijn wel leuke werken om door te bladeren.

“Software studies” is een verzameling artikelen over het schrijven en de werking software vanuit een cultureel / nieuwe media standpunt.

Code Complete is *het* standaardwerk voor alle programmeurs: je wordt er een betere programmeur van. Van harte aan te raden!

Software Engineering gaat over het complete productieproces van software: van specificaties tot quality assurance.

The Pragmatic programmer gaat over hoe je als programmeur meer effectief kunt zijn en hoe je solide software van hoge kwaliteit maakt. Het gaat in op alle aspecten, zowel technische als niet-technische. Een aanrader.



http://nl.wikipedia.org/wiki/Pseudocode

En een “algorithme” is een bepaalde manier om een vastgesteld doel te bereiken: een stappenplan, een manier van aanpak.

Bij het schrijven van pseudocode is het het allerbelangrijks om een goed *niveau van abstractie* te vinden: niet te gedetailleerd, maar ook niet te globaal.

Als je je pseudocode goed schrijft, kan je deze later, wanneer je echt gaat programmeren, **1-op-1 overnemen** in de comments van je echte programma

Pseudocode

- Statements
- If-then-else
- Loops
- Variabelen
- Condities
- Functies

Pseudocode herhaling

- Statements, al dan niet *geparametrizeerd*
  - “Verwarm de oven voor”
  - “Verwarm de oven voor op X graden”
- Vertakkingen (branches)
  - IF … THEN … (ELSE …)
- Herhalingen (loops)
  - REPEAT … UNTIL …
- Condities: “checks” waar / niet waar

Vertakkingen (branches)

- Oorzaak → gevolg: “Als dit, dan dat”
- IF iets, THEN doewat, (ELSE doewatanders)

Expressie

Statement(s)

Statement(s)

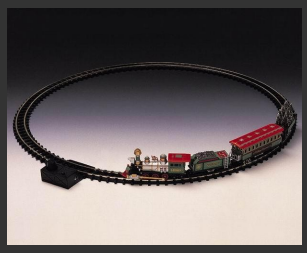
- IF oventemperatuur te hoog
  THEN stop met voorverwarmen

Herhalingen (Loops)

- “Doe hetzelfde, totdat er iets geldig is”
- REPEAT doewat UNTIL iets *(geldig is)*

Statement(s)

Expressie



Variabelen

- Een “vakje” waar iets in kan worden bewaard

```
SET gewensteTemperatuur TO 220
activeer oven
REPEAT
    SET huidigeTemperatuur TO de huidige oventemperatuur
UNTIL huidigeTemperatuur >= gewensteTemperatuur
```

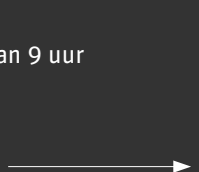
~~Expressies~~ Condities

- “huidigeTemperatuur GROTER DAN gewensteTemperatuur”
  → is een *conditie*
- Expressies zijn algemener dan condities
- Niet perse waar/niet waar, maar ook bv.
  - *Maximum levens - Levens verbruikt*
  - lowercase(player name)
- **Condities** zijn expressies die waar/niet waar (true/false) als resultaat hebben

Logische blokken

- Door code te “indenten” krijg je inzicht in welke stukken bij elkaar horen

```
IF
WekkerTijd kleiner dan 9 uur
THEN
Wekker op snooze
REPEAT
Slapen
UNTIL
WekkerTijd 9 uur
END
Sta eindelijk op
```



```
IF  WekkerTijd kleiner dan 9 uur THEN
    Wekker op snooze
    REPEAT
        Slapen
    UNTIL
    WekkerTijd 9 uur
END
Sta eindelijk op
```

2 ”geneste” niveaus


• “Verwarm de oven op X graden”
• Dat verwarmen zelf heeft ook weer een stukje pseudocode...!
• ...*sub*-pseudocode?
• X is hier een *parameter*

Functies

- Statement: voorverwarmen(220)
- Implementatie:

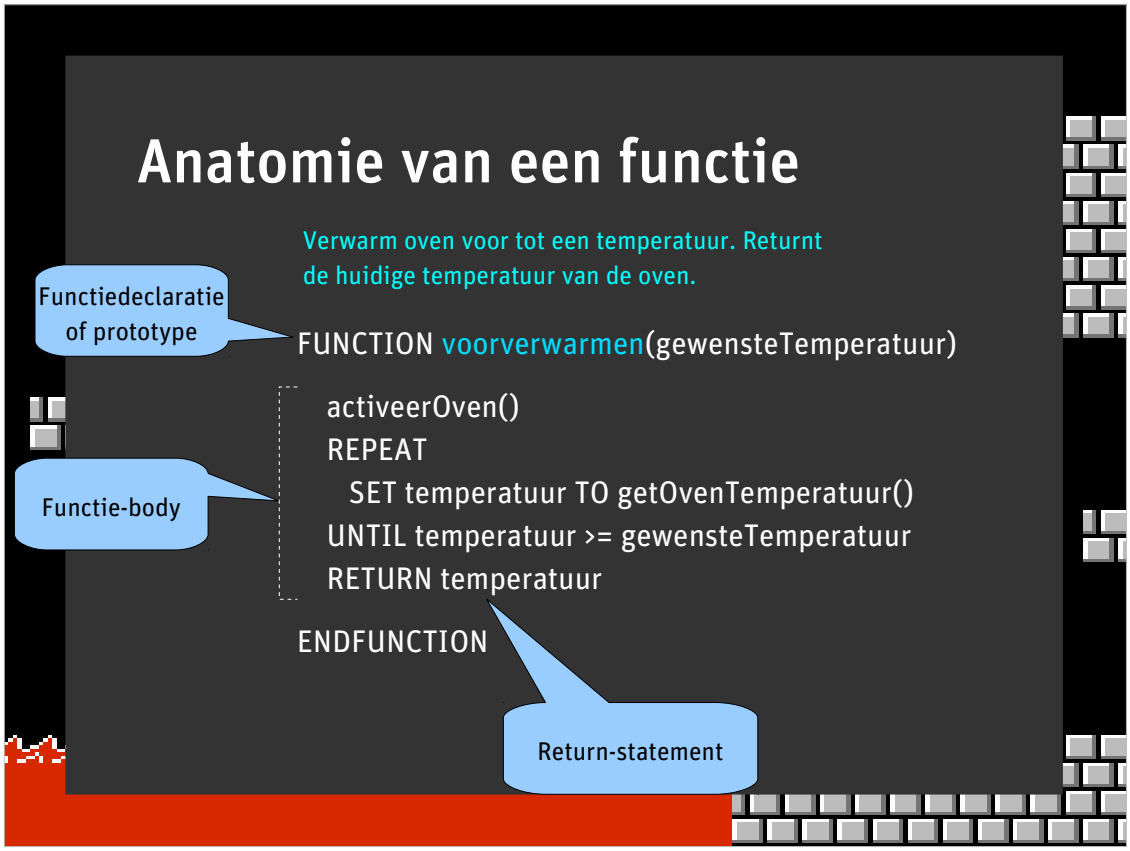
```
FUNCTION voorverwarmen(gewensteTemperatuur)
  activeerOven()
  REPEAT
    SET temperatuur TO getOvenTemperatuur()
  UNTIL temperatuur >= gewensteTemperatuur
  RETURN temperatuur
ENDFUNCTION
```

Da's ook weer een functie!

Yup.

Functies

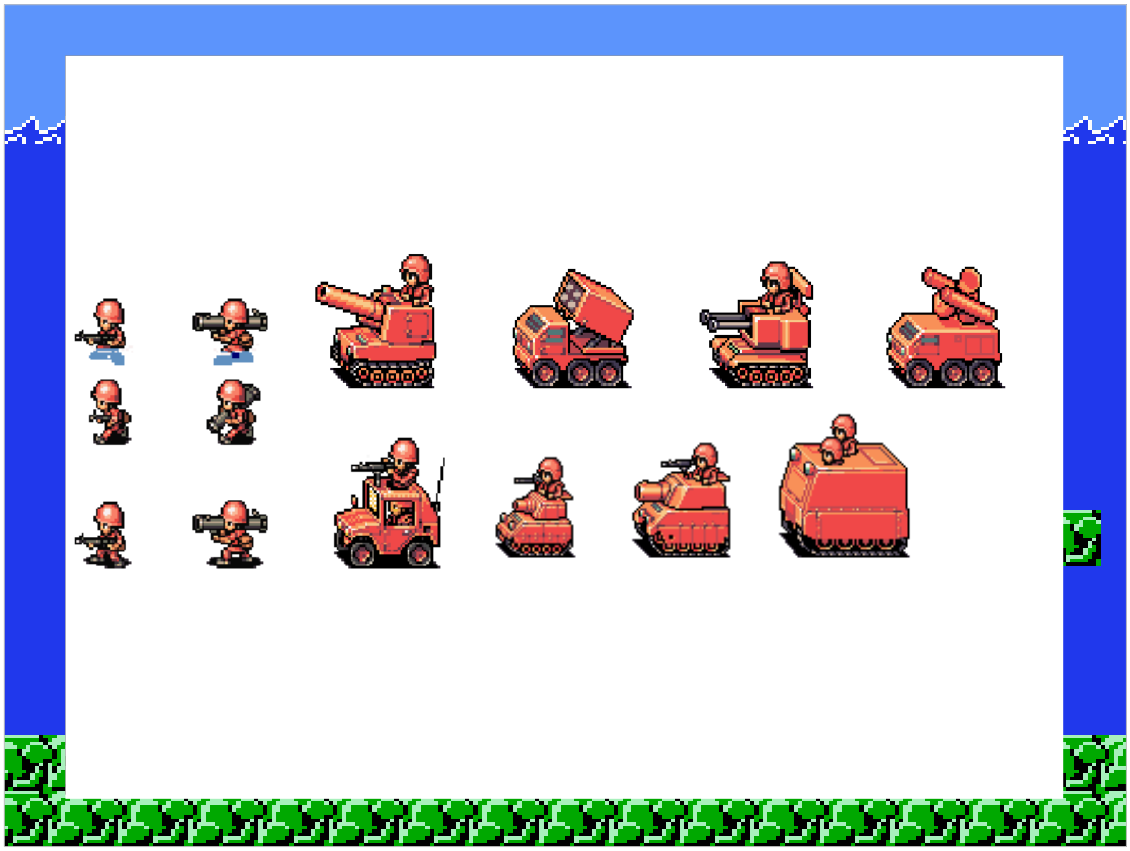
- In sommige talen heten dat *subroutines*
- Don't Repeat Yourself! (D.R.Y.)
- Maken het herbruiken van code makkelijker
- Verstoppen de onnodige details
- Functienaam is vaak een werkwoord: het doet iets


Anatomie van een functie
Verwarm oven voor tot een temperatuur. Returnt de huidige temperatuur van de oven.
Functiedeclaratie of prototype
FUNCTION voorverwarmen(gewensteTemperatuur)
activeerOven()
REPEAT
SET temperatuur TO getOvenTemperatuur()
UNTIL temperatuur >= gewensteTemperatuur
RETURN temperatuur
ENDFUNCTION
Functie-body
Return-statement

Onderdelen v.e. functie

- Declaratie / prototype
  - Hoe heet de functie
  - Welke parameters gaan er in
- Functie body
  - De *implementatie*: hier gebeurt het werk
- Return statement
  - Om uit de functie te springen met een value

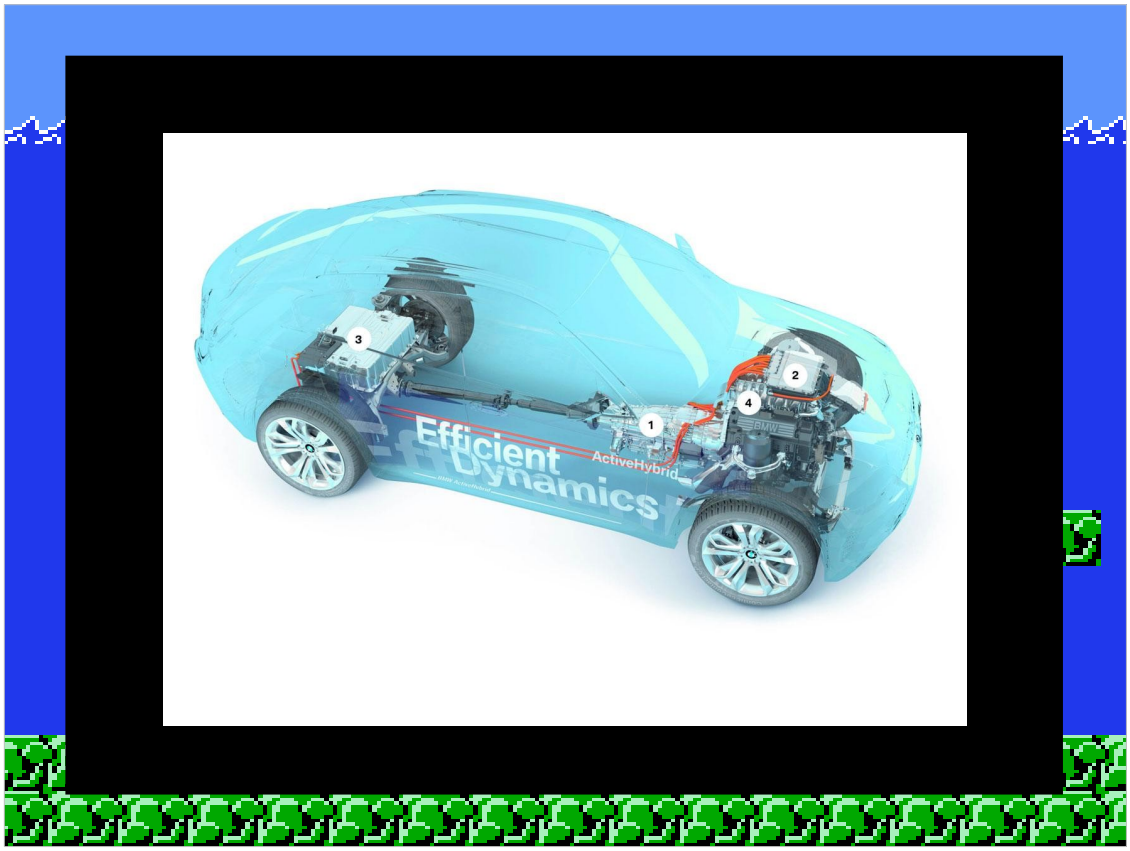
Laten we even kijken naar de tussenstand van de ingeeverde opdrachten.



We gaan het over objecten hebben.

Objecten

- Logische groepering van functionaliteit
  - Eigen data (attributen)
  - Eigen *methodes* (functies)
- Begrippen:
  - Interface
  - Encapsulatie / data hiding
  - Overerving



Als je een object gebruikt, hoef je je geen zorgen te maken over hoe het precies werkt. Als je maar snapt hoe je het moet gebruiken.

Encapsulatie

- Omdat je niet alles van een object hoeft te weten
- De interface van de class bepaalt het gedrag
- Voordelen:
  - Je interface documenteert zichzelf
  - Later makkelijker aan te passen (refactoring)
  - Data op 1 plek

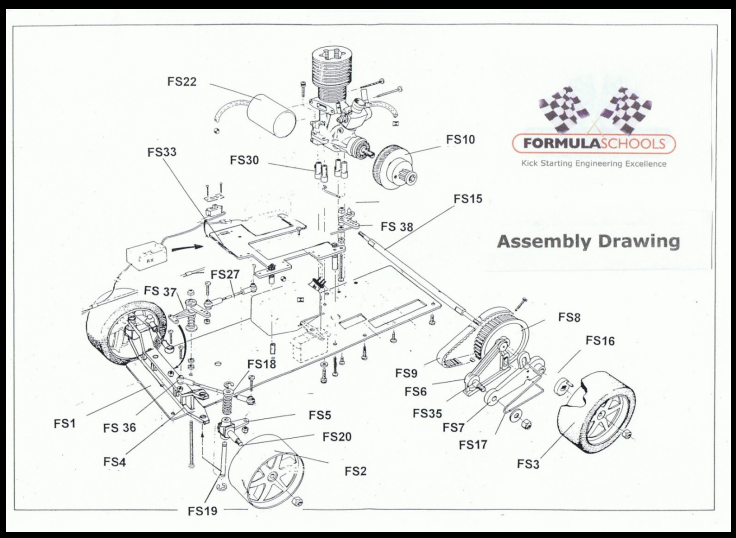
Dune II

Er is een trooper geselecteerd. Dat is een unit van het type “voetvolk”. Zoals uit de interface duidelijk wordt kan je 4 dingen doen met de unit.

Objecten hebben een publieke “interface”. Dat is de manier waarop je het object kan gebruiken.

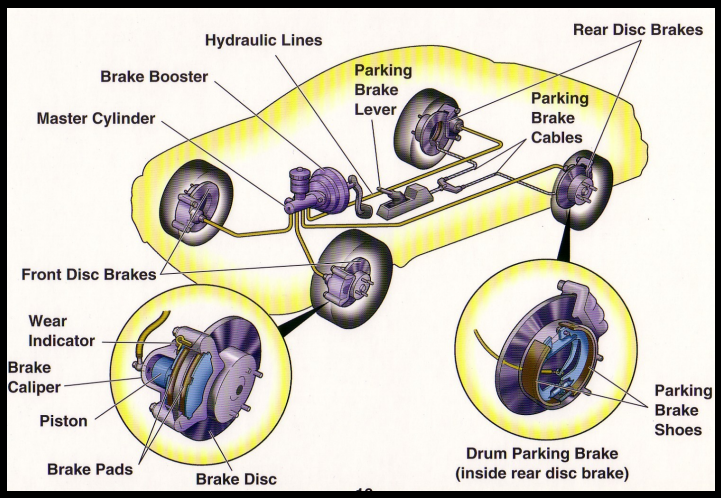
Met zijn interface geeft het object aan wat het kan en hoe het gebruikt dient te worden.

Attributen



Attributen zijn de onderdelen van een object. Het is de data waar het object mee redeneert.

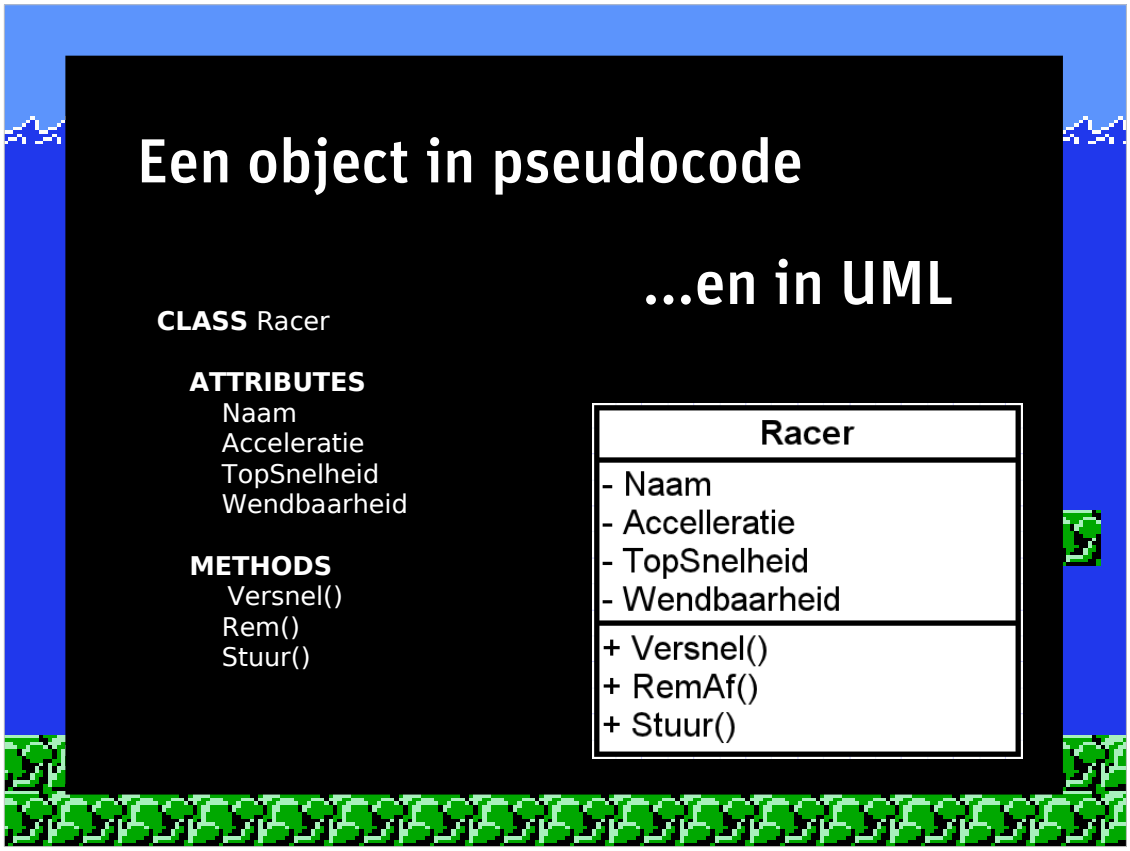
Methoden



Methoden kun je zien als werkwoorden. Het zijn acties die je op een object kunt aanroepen.

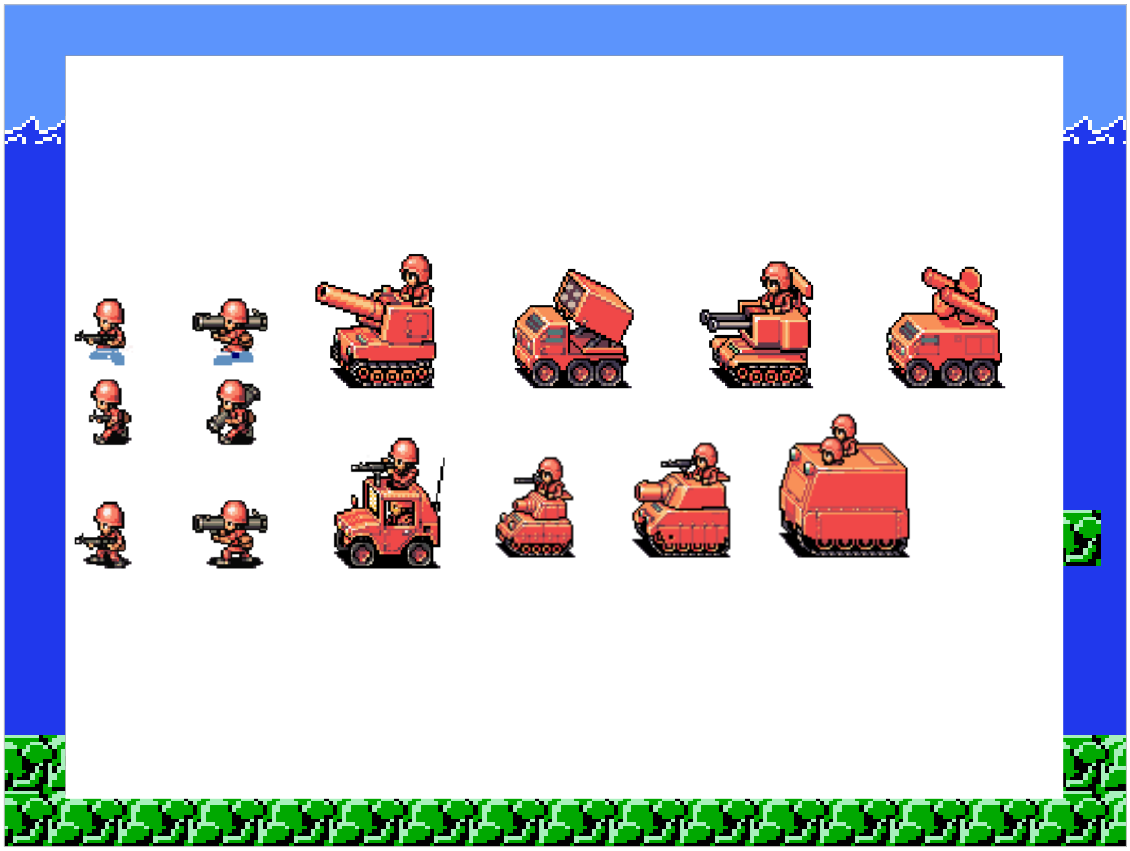
Eigenlijk zijn methoden gewoon functies. Ze bevatten een reeks instructies die uitgevoerd kunnen worden.

Het verschil is dat ze altijd uitgevoerd worden “op” een object. Een methode is een functie op het object.

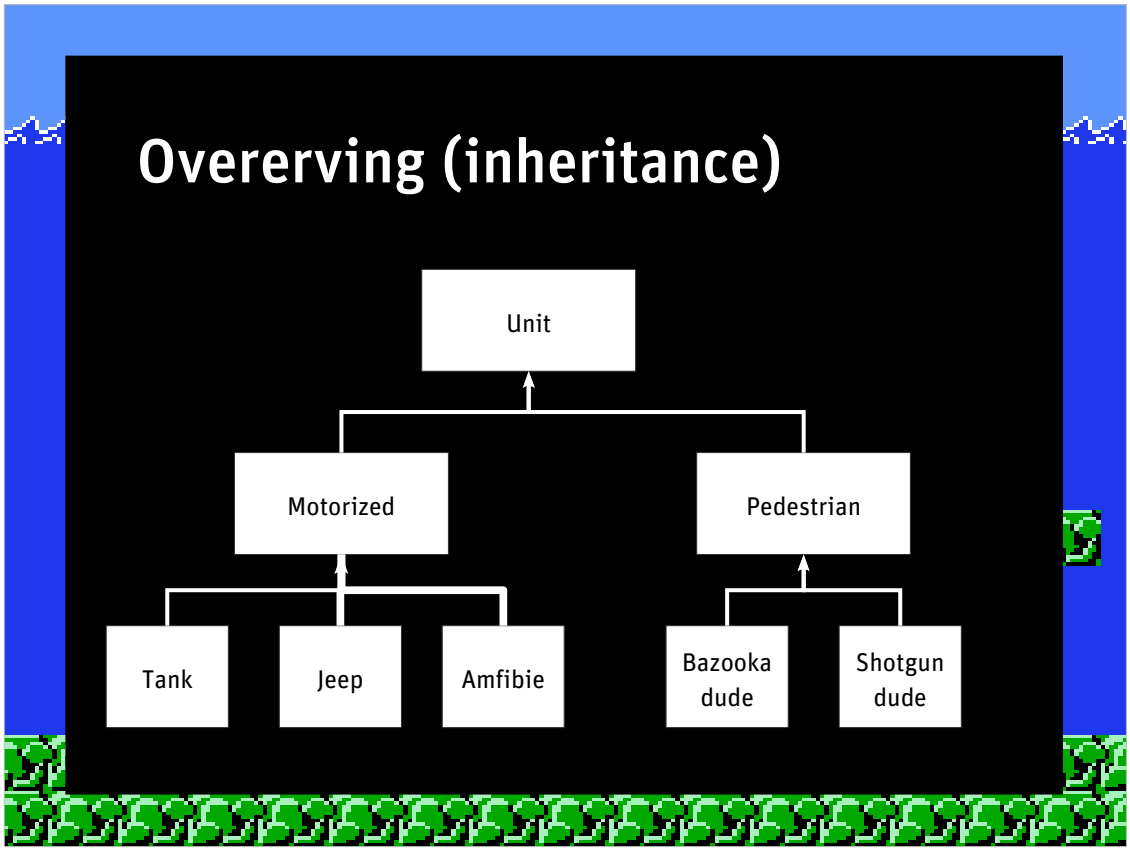


UML: Universal Modelling Language

Een gangbare schema-taal waarmee software wordt gemodelleerd.



En een class is niet alleen.
Er zijn vaak veel classes die op elkaar lijken, die dingen met elkaar gemeen hebben. Daar kunnen we iets voor verzinnen.



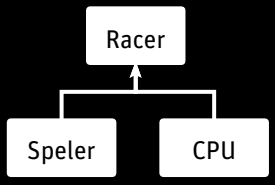
Classes kunnen eigenschappen van elkaar overnemen.

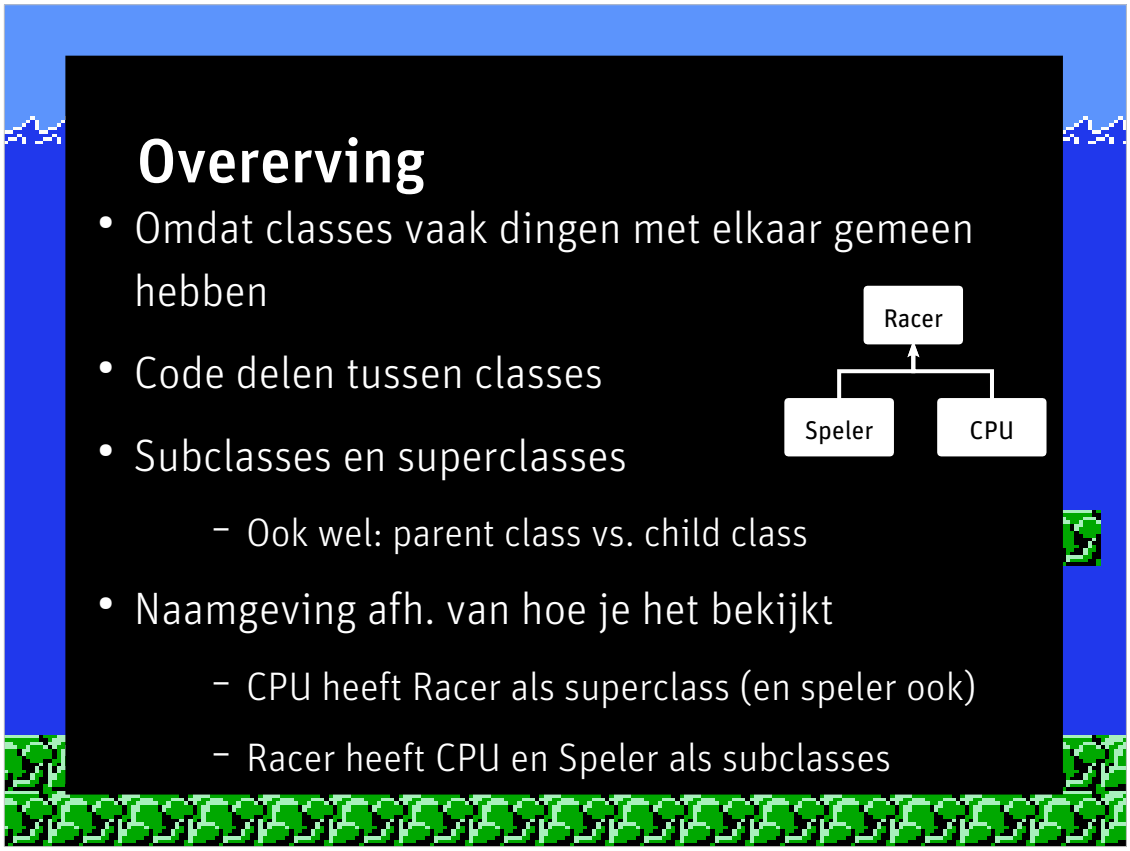
Ze “delen” eigenschappen. Alle units in een RTS hebben bv. Een positie, maar ze kunnen niet allemaal schieten.

CLASS Racer
ATTRIBUTES
Naam
Acceleratie
TopSnelheid
Wendbaarheid
METHODS
Versnel()
Rem()
Stuur()

CLASS Speler **EXTENDS** Racer
ATTRIBUTES
ControllerNummer
PowerUp
AantalMuntjes

CLASS CPU **EXTENDS** Racer
ATTRIBUTES
AI_Patroon





Je doet aan overerving van classes omdat er vaak dingen in je project zijn die op elkaar lijken, maar toch niet helemaal hetzelfde zijn.

Probeer die dingen te benoemen en kijk of je overeenkomende eigenschappen en methoden kunt vinden. En ook verschillen: Wat heeft de ene class wel wat de andere class niet heeft.

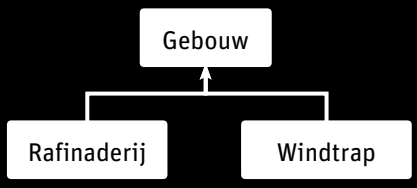
Waar zitten de overeenkomsten? Die gaan in de superclass

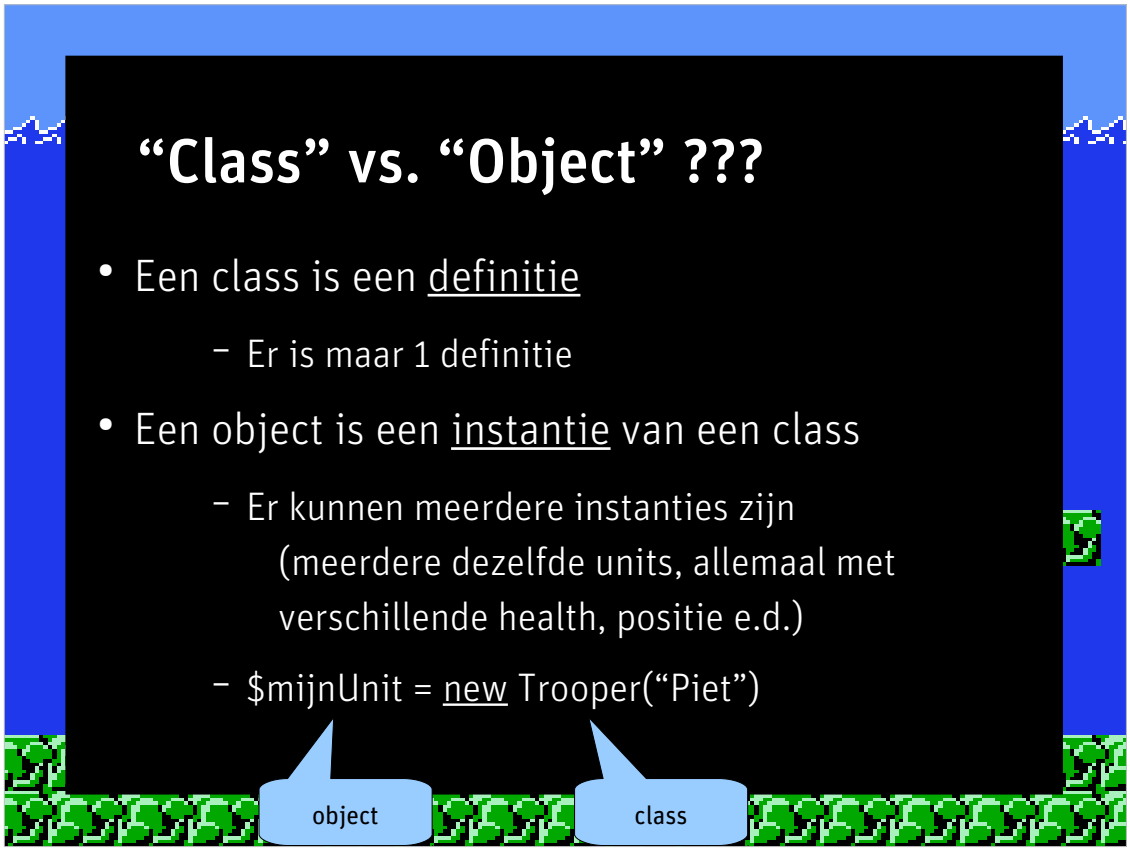
Waar zitten de verschillen? Die gaan in de subclasses.

CLASS Gebouw
ATTRIBUTES
Naam
Prijs
HitPoints
Stroomkosten
METHODS
Bouw()
Repareer()

CLASS Windtrap **EXTENDS** Gebouw
ATTRIBUTES
StroomCapaciteit
METHODS
WinStroom()

CLASS Rafinaderij **EXTENDS** Gebouw
ATTRIBUTES
SpecieCapaciteit
AantalHarvesters
METHODS
VerzamelSpecie()

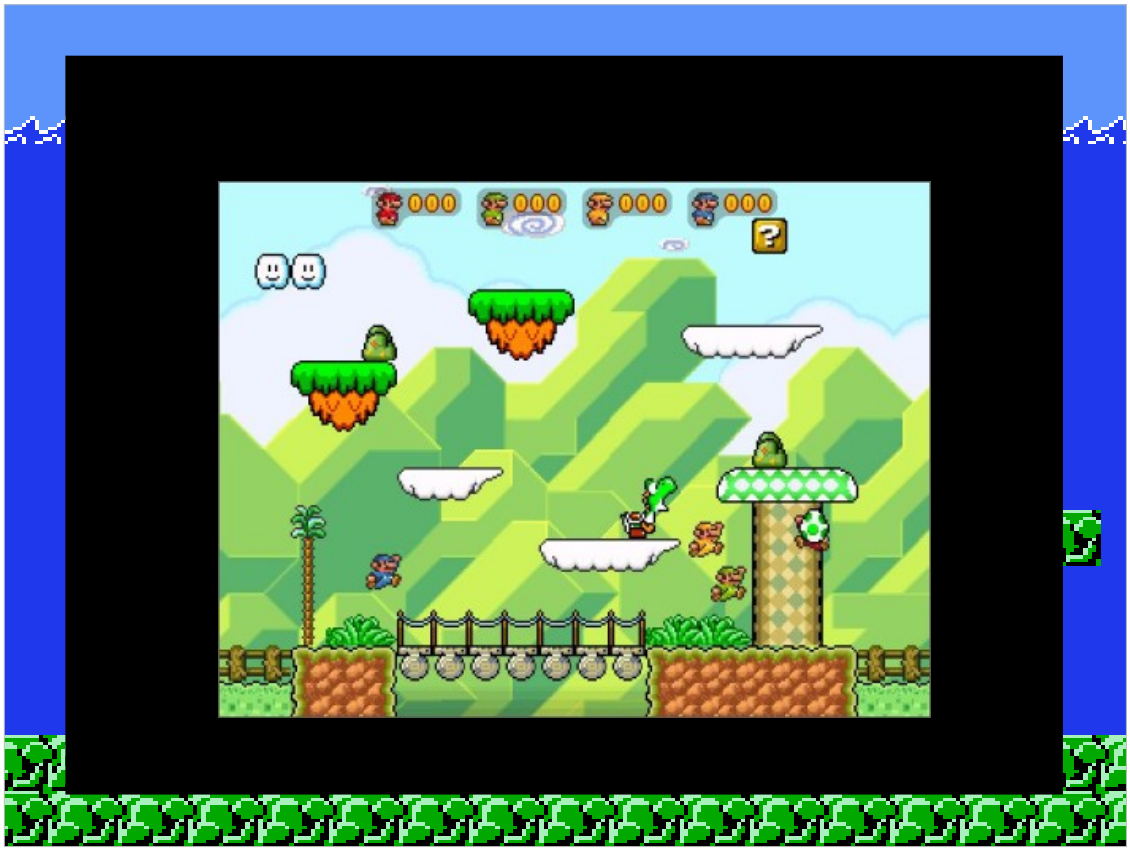




Als je in een RTS game een overzicht hebt van je units die je kunt laten bouwen, dan kijk je eigenlijk naar een lijstje van **classes**.

Als je een unit laat bouwen in de game, dan wordt hij **geinstantieerd** tot een **object**, met zijn eigen naam, health, positie, e.d.

Je kunt het ook vergelijken met de ideeënleer van plato.
http://nl.wikipedia.org/wiki/Idee%C3%ABnleer_%28Plato%29
Classes zijn ideeën, onvergankelijk en ontastbaar; objecten zijn de voorwerpen: vergankelijk, tastbaar, het zijn "afbeeldingen" van de classes.



Hoe zou je Super Mario Wars opdelen in een class hierarchy?

Oefenopdracht

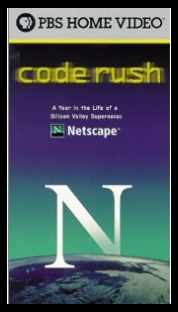
- Bedenk hoe het class diagram van je favoriete game eruit zou zien. Gebruik niet meer dan 5 classes, en minimaal 1 maal inheritance. Beschrijf de attributen en methodes van de classes.

  Opdracht niet verplicht, maar wel nuttig
  → tentamenstof! Mailen == feedback.

Zijn we er nog?

Popcornmateriaal

- Minecraft documentaire
- Code Rush (over opensourcen van Netscape)



Komende colleges

- ~~College 1: waar hebben we het over~~
- ~~College 2: imperatief programmeren~~
- ~~College 3: object orientatie~~
- College 4: vervolg OO, design patterns, Q&A
- Tentamen (begin december)


Tot volgende week!