Bachelorscriptie 2011

Ilian van der Velden

Leren programmeren

in het Voortgezet Onderwijs

Abstract

De Enigma methode is een lesmethode voor het vak informatica op de middelbare school. In dit onderzoek zal een module over programmeren uit de Enigma methode geanalyseerd worden aan de hand van verschillende vakdidactische standaarden. Tevens zullen er aanbevelingen worden gegeven om deze methode te verbeteren. Een andere leermethode voor programmeren is de methode Objects First. De programmeeromgeving Greenfoot maakt gebruik van deze methode. Met behulp van enkele experimenten worden opdrachten die gebruikt kunnen worden bij Greenfoot verbeterd.

Inhoudsopgave

Inleiding 3
Hoofdstuk 1: Theoretisch kader 4
1.1 Leren programmeren 4
1.2 Analyse van lesstof 4
1.3 Objects First 6
1.4 Leerdoelen 6
Hoofdstuk 2: Analyse 7
2.1 AC graph 7
2.2 Analyse met PCK: Loops 9
2.3 Vergelijking met Instruct 11
2.4 Voorbeelden 13
2.5 Oriëntatie 13
2.6 Ordening van leerinhoud 13
Hoofdstuk 3: Evaluatie 14
3.1 Voorbeelden 14
3.2 Oriëntatie 14
3.3 Ordening van leerinhoud 15
Hoofdstuk 4: Leren programmeren met Greenfoot 16
4.1 Eerste experiment 16
4.2 Tweede experiment 17
4.3 Afronding 18
Conclusie 19
Bronnen 20
Appendix A: opdracht 1 21
Appendix B: opdracht 2 26
Appendix C: Opdrachtenhandleiding 29

Inleiding

Het programmeren van een computerprogramma is vaak iets dat door mensen wordt bestempeld als onbegrijpelijk en ingewikkeld. Toch wordt het elk jaar weer aan vele scholieren en studenten uitgelegd, en leren zij computerprogramma's schrijven. Hoe kun je dit nu eigenlijk het beste aan iemand leren? Welke moeilijkheden komen naar voren bij het leren van programmeren?
Een van de lesmethodes die op dit moment op de middelbare school wordt gebruikt is de Enigma Methode. In dit onderzoek zal een analyse worden gemaakt van deze methode aan de hand van verschillende wetenschappelijke inzichten. Tevens wordt er een vergelijking gemaakt met een andere lesmethode.
Daarna wordt er een evaluatie gemaakt van de Enigma methode. In deze evaluatie wordt er een waardeoordeel gekoppeld aan de punten die in de analyse aan de orde zijn gekomen.
In een aanvullend hoofdstuk wordt gekeken naar een relatief nieuwe manier van leren programmeren: Objects First. De programmeeromgeving Greenfoot maakt gebruik van deze programmeermethode. Er worden een aantal experimenten gedaan met een opdracht in Greenfoot om deze opdracht te verbeteren. Tevens wordt er een handleiding bij de opdracht geschreven om deze gemakkelijker bruikbaar te maken.

Hoofdstuk 1: Theoretisch kader

1.1 Leren programmeren

In de wetenschappelijke wereld zijn verschillende onderzoekers bezig geweest met het onderzoeken van hoe men iemand programmeren leert. Het “leren programmeren” is iets dat vaak als moeilijk wordt bestempeld.
Het onderzoek van McCracken uit 2001[1] gaat hier verder op in. In dit onderzoek wordt gekeken naar de moeilijkheden die eerstejaars informatica-studenten hebben met het leren van programmeren. Er werd na het volgen van een cursus programmeren verschillende opdrachten gegeven, en deze opdrachten werden gemiddeld gezien behoorlijk slecht gemaakt. In het artikel wordt tevens een framework opgezet om op een algemene manier de resultaten van een opdracht te beoordelen, door middel van een DoC-score (Degree of Closeness). Met deze score wordt aangegeven hoe dichtbij de oplossing was om ook daadwerkelijk te werken. Een score van 5 is het hoogst, waarbij de student in principe een werkend programma zou moeten hebben. Een score van 1 geeft aan dat een student geen idee had hoe het probleem te benaderen was.
Het artikel van Nico van Diepen geeft ook verschillende redenen waarom programmeren moeilijk is. [2] Hij schrijft dat vooral het abstraheren en generaliseren van de programmacode lastig is.

1.2 Analyse van lesstof

1.2.1 Anchor Graphs

In het artikel van Mead uit 2006[3] wordt een methode beschreven om lesstof inzichtelijker te maken. Er wordt gebruik gemaakt van Anchor Concepts. Dit zijn belangrijke concepten in de lesstof die centraal staan in het begrijpen van de totale lesstof. Bij het uitleggen van nieuwe Anchor Concepts is het volgens het onderzoek belangrijk dat deze worden gekoppeld aan Foundational Anchor Concepts. Dit zijn concepten die door een leerling al worden begrepen of die kunnen worden uitgelegd aan de hand van bekende begrippen. Aangezien de Anchor Concepts aan elkaar gekoppeld worden kan een conceptueel schema worden opgesteld aan de hand van de begrippen uit lesstof. Mead geeft enkele Anchor Graphs bij bepaalde lesstof als voorbeeld.

1.2.2 PCK

De PCK methode is ontwikkeld door Schulman (1986). PCK staat voor Pedagogical Content Knowledge en gaat over de kennis van het onderwijzen en het leren van een onderwerp. Er komen vier belangrijke aspecten aan bod:

- Waarom is dit onderwerp belangrijk om te leren?
- Wat moet er precies geleerd worden?
- Hoe wordt de voorkennis van de studenten geschat, wat zijn misschien moeilijkheden of verwarrende onderdelen van de lesstof?
- Hoe moet de lesstof overgebracht worden? (welke methoden)

In het onderzoek van Saeli wordt de CoRe-methode gebruikt om de PCK te achterhalen. Deze methode is ontwikkeld door Australische onderzoekers (Loughran, Gunstone, Berry, Milroy & Mulhall, 2000; Loughran et al, 2001; Mulhall, Berry & Loughran, 2003; Loughran et al, 2004). De CoRe methode bestaat uit 8 vragen waaruit de PCK is samen te stellen.

In het artikel van Saeli uit 2011 (nog ongepubliceerd) is gebruik gemaakt van de CoRe methode om de PCK te achterhalen van programmeeronderwijs op de middelbare school. Er is dus gekeken hoe leraren en lerarenopleiders denken over bepaalde termen uit de informatica. Er wordt bijvoorbeeld onderzocht hoe leraren bepaalde begrippen willen overbrengen en hoe zij de kennis van de studenten over deze begrippen schatten. De onbewerkte resultaten van het onderzoek staan in het technical report van Jochems et al.[4]

1.2.3 Volgorde van leerinhoud

Een andere manier om een lesmethode te analyseren is door te kijken naar de volgorde van leerinhoud. Op de website van de Katholieke Universiteit Leuven[5] is hier een uitgebreide uitleg van. Zij onderscheiden drie ordeningsprincipes:

- lineair
- conceptueel
- elaborerend

Bij de lineaire ordening staat de opeenvolging van leerinhoud centraal. De verschillende onderdelen moeten dan in een “welbepaalde volgorde” worden behandeld, waarbij ook de overgangen belangrijk zijn.
Bij de conceptuele structurering gaat het om concepten die een bepaalde relatie tot elkaar hebben. De website schrijft: “*men kan vertrekken van veelomvattende begrippen om later tot verbijzonderingen te komen, of men kan de omgekeerde weg volgen: van het elementaire naar de complexe structuur*”. In deze volgorde staat het vastknopen van nieuwe begrippen aan al bekende begrippen centraal. Dit principe wordt ook genoemd in het artikel van Mead.
De elaborerende ordening wordt ook wel het “zoomlensprincipe” genoemd. Bij deze ordening wordt in het begin de gehele leerinhoud verkort weergegeven. Daarna wordt er per deel ingezoomd en wordt er gedetailleerd gekeken naar elk deel.

1.3 Objects First

Er bestaat al enige tijd een nieuwe leermethode voor programmeren, namelijk Objects First. In deze methode staat de context, de wereld centraal. Vanaf het begin wordt er gebruik gemaakt van objecten en inheritance, en is object-georiënteerd programmeren belangrijk. Aan de hand van de objecten worden dan later traditionele programmeerstructuren uitgelegd. Het is gebruikelijk om bij deze methode in het begin gebruik te maken van een programmeeromgeving die zich richt op het visuele aspect van programmeren. Er zijn verschillende onderzoeken naar Objects First gedaan. In het onderzoek van Cooper, Dann en Pausch[6] wordt getoond hoe leerlingen omgaan met deze methode. De auteurs van dit artikel beweren dat dit de beste manier is om studenten kennis te laten maken met programmeren. Het artikel van Bennedsen en Schulte[7] richt zich op de vraag hoe docenten de methode in de praktijk toepassen en hoe de docenten naar deze methode kijken. De auteurs schrijven dat er al enige jaren een hevig debat is welke lesmethode het meest geschikt is voor het leren programmeren. Uit hun onderzoek blijkt dat er drie categorieën zijn die centraal staan bij Objects First, namelijk: het gebruik van objecten, het aanmaken van klassen en het concept van object-georiënteerd programmeren.

1.4 Leerdoelen

In de taxonomie van Benjamin Bloom wordt onderscheid gemaakt tussen 3 categorieën van leerdoelen: cognitieve, affectieve en motorische doelen. Cognitieve doelen refereren aan doelen zoals kennis hebben van een bepaald concept of een concept begrijpen. Affectieve doelen hebben te maken met emoties, of een bepaald concept wordt gewaardeerd. Motorische doelen zeggen iets over of een persoon een bepaalde handeling kan uitvoeren.

Hoofdstuk 2: Analyse

In dit hoofdstuk zal er een analyse worden gemaakt van het introducerende programmeerhoofdstuk van de Enigma methode.

2.1 AC graph

In het artikel van J. Mead [3] wordt een AC graph gegeven die weergeeft op welke manier er een introducerende programmeercursus vormgegeven kan worden.
In deze AC graph staan de nieuwe begrippen/concepten rond omlijnd, en de bekende begrippen waaraan deze nieuwe begrippen gekoppeld hebben een vierkante omlijning. Een "*variable*" wordt bijvoorbeeld gekoppeld aan de begrippen *address*, *memory* en *data*.

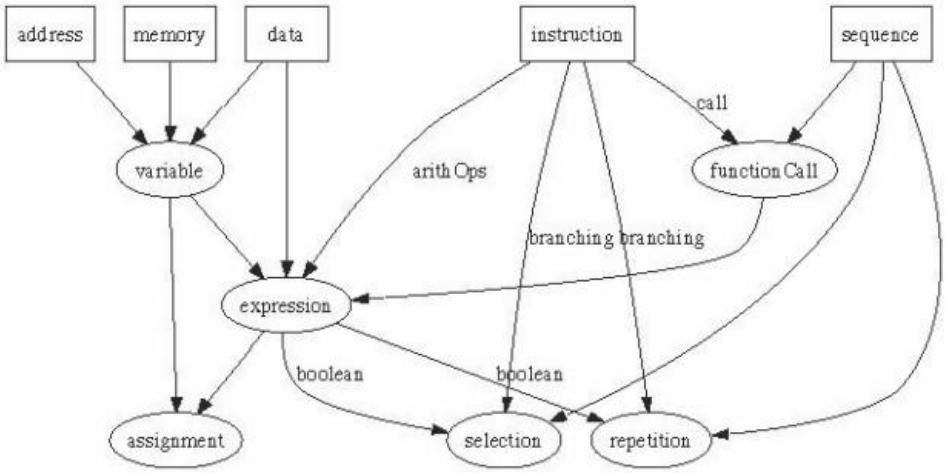


Figure 1: AC Graph for Algorithmics

Bij de opbouw van deze graaf heeft de machine-oriëntatie centraal gestaan. De foundational concepts in de graaf (aangegeven door de vierkanten) zijn allemaal concepten uit de machine-oriëntatie.
Het is van belang dat de concepten waaraan de nieuwe concepten worden gekoppeld bekend zijn bij de leerling. Sommige concepten, zoals bijvoorbeeld een *assignment* worden uitgelegd aan de hand van andere nieuwe concepten. In dat geval is het uiteraard nodig dat eerst deze andere concepten worden uitgelegd.
In het artikel van Mead wordt de suggestie gedaan om voor het begin van een cursus over algoritmen eerst een simpele von Neumann machine uit te leggen. De foundational concepts kunnen in dat geval aan die machine gekoppeld worden.

2.1.1 Enigma

Het inleidende programmeeronderwijs in de Enigma methode is in de eerste vier hoofdstukken geplaatst.
In het eerste hoofdstuk wordt kennis gemaakt met verschillende bestandsformaten en de manier waarop deze worden opgeslagen (binair).
In het tweede en derde hoofdstuk worden de bewerkingen op de binaire en hexadecimale getallen behandeld en de hardware die bij de computer hoort.
In het vierde hoofdstuk worden de eerste programma's geschreven. Er wordt eerst kort ingegaan op Programma Structuur Diagrammen (PSD's) en daarna wordt er in Java geprogrammeerd.
Met PSD's worden sequentie, keuze en herhaling uitgelegd, en wordt er gebruik gemaakt van variabelen. Daarna wordt met Java in de onderstaande volgorde bepaalde onderwerpen behandeld:

3 Java-programma's nader bekeken
3.1 Toekenning en expressie
3.2 Namen in Java
3.3 Declaratie van variabelen
3.4 Het RenteApplet
3.5 Toekenning door operaties en operanden
3.6 Herhaling met while
3.7 Methode-aanroep
3.8 Conversie
3.9 Statements
4 Herhaling en keuze
4.1 Het RenteApplet anders geschreven
4.2 Declaratie en toekenning
4.3 Het for-statement
4.4 Decimale getallen omzetten naar binaire getallen
4.5 Het do-while-statement
4.6 Het if-else-statement
5 Variabelen en operatoren
5.1 Gehele getallen
5.2 Reële getallen
5.3 Operatoren
5.4 Random getallen
5.5 Conversie van numerieke variabelen
5.6 Type boolean
5.7 Lokale variabelen en instance variabelen
6 Strings
6.1 Het PalindroomApplet
6.2 Het gebruik van methoden
6.3 Concatenatie van strings
6.4 Conversie van en naar strings

Wat in het hoofdstuk opvalt, is dat de foundational concepts uit de AC graph bijna niet aan de orde komen. Er wordt bij de uitleg van een nieuw concept zoals herhaling vooral geleund op het functionele ("wat doet het") en op voorbeelden. Er is weinig koppeling met de achterliggende hardware.

2.2 Analyse met PCK: Loops

In deze paragraaf zal een onderdeel van het hoofdstuk van Enigma worden geanalyseerd aan de hand van de resultaten van het onderzoek van Jochems et al.[4]. Er is gekozen voor het onderdeel “Loops” omdat deze het beste aansloot bij de onderwerpen die in het hoofdstuk worden behandeld.
De technical report van Jochems et al. [4] beschrijft het volgende over loops:

> *There are three main goals: the role of the initialization, which means what happens before the loop; the role of the condition; the role of the variables of the iterative step; recognition of the group of instruction to repeat; the possible role of the counter, often loops have it whether explicit or implicit; and then aspects relative to the termination of the loop, regarding pragmatic issues and not theoretical.*

In het onderzoek van Saeli komt het volgende overzicht naar voren:

Big Idea: Control structure, focus on Loops

What about the students:
• **What else you might know about loops (that you don't intend students to know yet)?** -knowledge about recursion and invariants;| -more concise and efficient solutions; -knowledge of implementations of loops in different programming languages • **What are difficulties/ limitations connected with the teaching of loops?** -identifying: what is before loops; identifying the group of instruction to repeat; and after loops. -condition of loops (true/false or non trivial conditions); -explicit/implicit counters; zero iteration; limit cases (first and last iteration). -generalization of the problems; -to synthesize and to imagine the expected result of a loop; -difficulties in changing a term (e.g. modifying the initialization of a variable). • **What do you think students need to know in order for them to learn loops?** -Variables; -assignments; -conditions; -basic types, usually integer are used as counters in loops such as “for”. • **What are your specific ways of ascertaining students understanding or confusion around loops?** -simulation with pencil and paper; -explaining in words what does happen in the algorithm used and reasons to use certain variables; -a computational equivalence of iterative structures; -giving almost identical loops and asking students to identify the loop that actually solves a specific problem; -correcting her/his own program by asking to execute it in a debugging context; -giving a loop with a mistake and asking to spot it; -measuring students’ enjoyment in creating new and more complicated loops.

Er worden 6 moeilijkheden/beperkingen genoemd ten opzichte van het lesgeven over loops. Deze punten komen niet terug in de uitleg in de methode van Enigma. Er wordt geen aandacht besteed aan het handmatig bekijken wat er precies gebeurt in een while-loop, ook niet in de opdrachten in het verwerkingsboek. Randgevallen zoals een loop die nul keer wordt uitgevoerd of een oneindige loop komen niet voor in de uitleg.

In het artikel worden ook 4 onderwerpen genoemd waarvan wordt gedacht dat studenten deze eerst moeten begrijpen voordat Loops kunnen worden uitgelegd. (variabelen, assignments, condities en basistypen)
Als we naar het hoofdstuk van Enigma kijken zien we dat Loops voor het eerst uitgelegd worden in paragraaf 3.6. In 3.3 wordt de werking van variabelen en types uitgelegd. Assignments worden in 3.5 uitgelegd.
Alleen het onderwerp “conditie” of “voorwaarde” is nog niet uitgelegd voordat leerlingen in aanraking komen met loops. De werking van de voorwaarde wordt echter uitgelegd in de paragraaf over loops, er wordt namelijk geschreven dat dit een expressie is met type boolean.

Als laatste worden er 7 punten genoemd waarmee kan worden gekeken of leerlingen het onderwerp begrijpen of nog moeite ermee hebben. Er worden verschillende suggesties gegeven, waaronder:

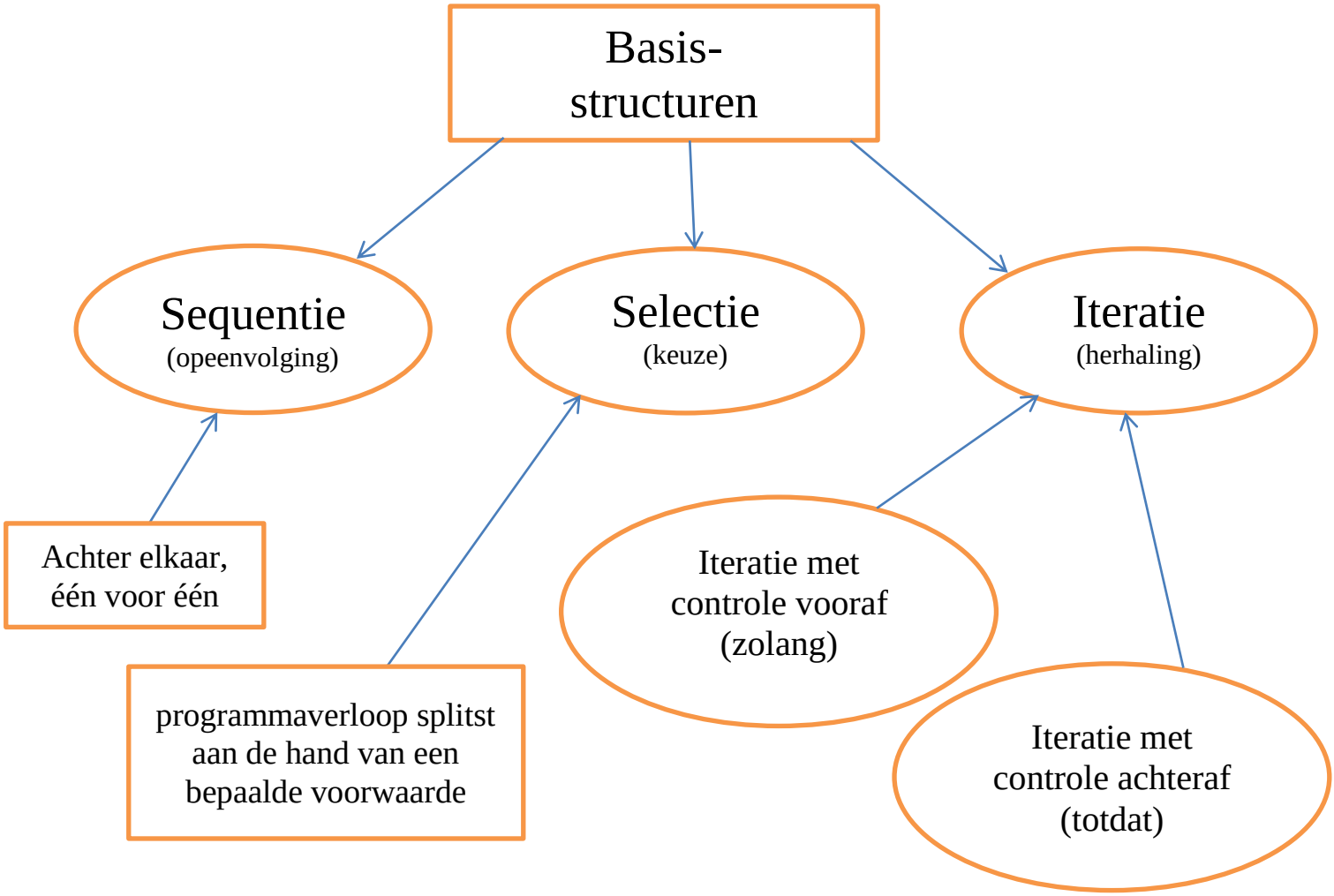
- Met pen en papier de loop simuleren.
- Twee loops geven en vragen welke van de loops een gegeven probleem oplost.
- Een foute loop geven en de leerlingen de fout laten zoeken. (in het technical report[4] zijn docenten het oneens of dit een goede methode is)

De suggesties die worden gedaan in het artikel van Saeli komen niet voor in de methode van Enigma. De uitleg in het boek geeft geen stapsgewijze uitleg van de werking van de loop. De opdrachten in het verwerkingsboek beperken zich tot de theoretische kant, en gaan niet in op de praktische werking van de loop.

2.3 Vergelijking met Instruct

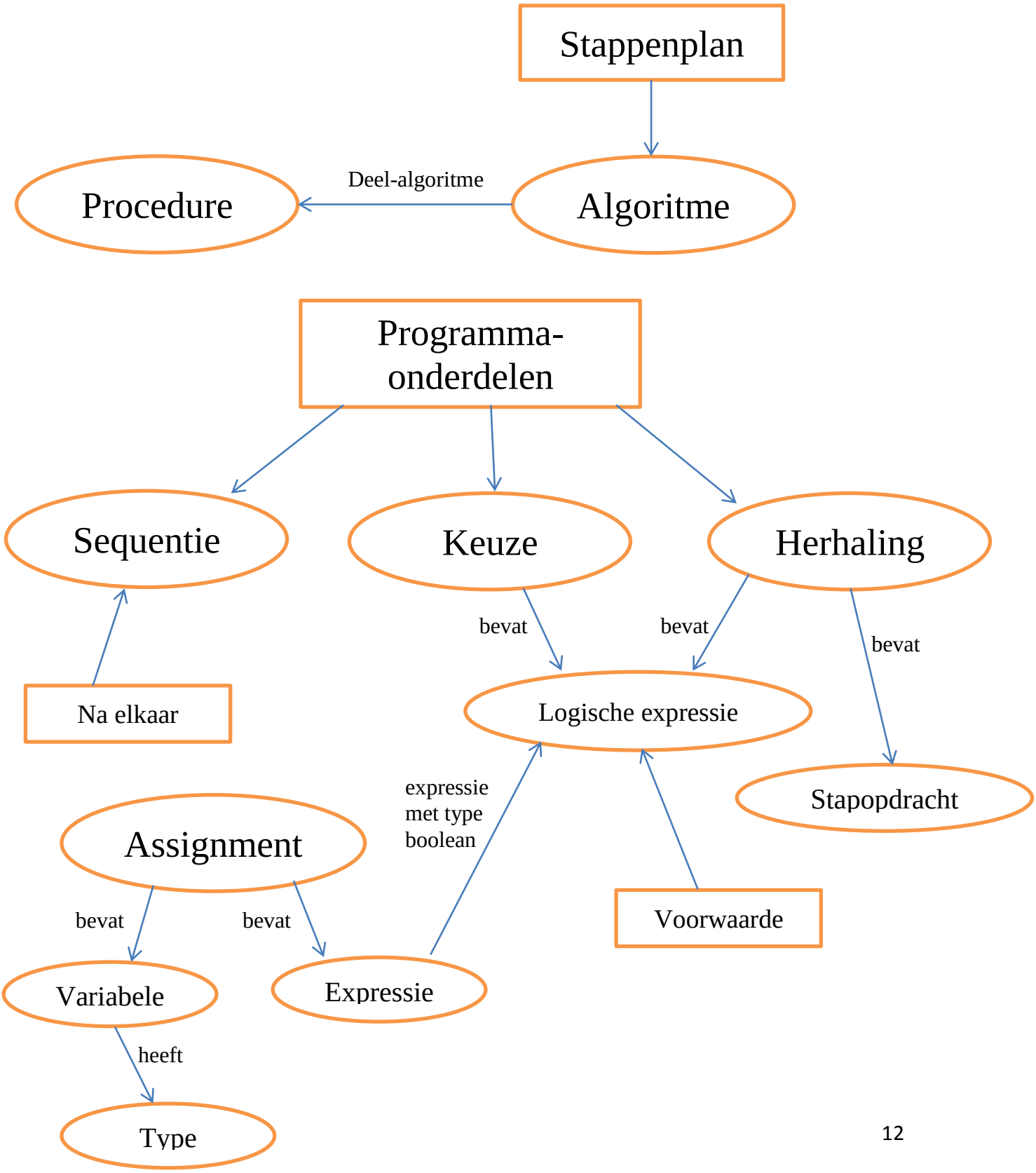
Instruct is een van de andere lesmethodes die in Nederland gebruikt worden op het Voortgezet Onderwijs. Er is een conceptuele analyse gemaakt (analoog aan de methode van Mead [3]) van de introducerende programmeerhoofdstukken van Instruct en Enigma ter vergelijking.

2.3.1 Conceptuele analyse Instruct



2.3.2 Conceptuele analyse Enigma

Vanwege de lengte van het hoofdstuk van Enigma is ervoor gekozen alleen de eerste 6 paragrafen te gebruiken in de conceptuele analyse.



2.3.3 Vergelijking tussen de conceptuele schema's

Het belangrijkste verschil tussen Enigma en Instruct is dat de Instruct methode maar een beperkt hoofdstuk aanbiedt. In het hoofdstuk van Instruct wordt de basis van programma-structuren uitgelegd, de rest van het programmeeronderwijs is in afzonderlijke modules geplaatst. Deze modules zijn taal-specifiek. Hierdoor is het introducerende programmeerhoofdstuk een stuk korter dan dat van Enigma.

Bij de Enigma methode worden alle belangrijke begrippen uitgelegd in een uitgebreid hoofdstuk bestaande uit 25 pagina's. Er is voor gekozen om niet het gehele hoofdstuk te analyseren in het conceptuele schema omdat de eerste 6 paragrafen al meer omvatten dan de Instruct methode.

De drie basisstructuren komen wel in beide hoofdstukken centraal aan de orde. Bij de Instruct methode wordt bij elk van deze structuren een tweede naam gegeven die de structuur extra duidelijk maakt.

2.4 Voorbeelden

Om lesstof interessanter en beter begrijpelijk te maken voor leerlingen zijn goede voorbeelden van belang. In de methode van Enigma wordt gebruik gemaakt van verschillende voorbeelden:
In 4.1 wordt het bakken van een appeltaart als voorbeeld gebruikt.
In 3.4 tot 4.3 wordt er rente uitgerekend met een Applet.
In 4.4 wordt er een Applet beschreven om decimale getallen om te zetten naar binaire getallen.
In 4.6 wordt er gewerkt met een programma voor het berekenen van overwerkuren.
In 5.4 wordt een Applet beschreven om een dobbelsteen te simuleren.
In 6.1 wordt een Applet beschreven om te controleren of een woord een palindroom is.

Het verwerkingsboek bevat andere voorbeelden waarin de opdrachten gemaakt worden. Deze voorbeelden worden in deze analyse buiten beschouwing gelaten.

2.5 Oriëntatie

Zoals in paragraaf 1.1.1 al genoemd werd is het hoofdstuk van Enigma voornamelijk taalgericht. Er wordt bij de uitleg van nieuwe begrippen vooral geleund op het functionele ("wat doet het") en op voorbeelden. Er wordt gekeken hoe een bepaald probleem met Java kan worden opgelost.

2.6 Ordening van leerinhoud

Het hoofdstuk van Enigma is voornamelijk lineair en conceptueel geordend. Het hoofdstuk is opgedeeld in paragrafen die subkopjes bevatten. De subkopjes zijn meestal lineair georderend. De paragrafen zijn voornamelijk conceptueel geordend.

Hoofdstuk 3: Evaluatie

3.1 Voorbeelden

De voorbeelden die worden gebruikt bij de Enigma methode waren: het bakken van een appeltaart, rente uitrekenen, decimaal naar binair, overwerkuren berekenen, dobbelsteen simuleren en palindromen. Het is waarschijnlijk dat deze voorbeelden niet echt binnen de belevingswereld van een middelbare scholier vallen.
Het is een aanbeveling bij een mogelijk nieuwe versie van de methode de voorbeelden iets levendiger te maken. Wellicht dat nieuwe technologieën zoals Social Media of computergames een rol kunnen spelen in de voorbeelden die genoemd worden in het tekstboek.

3.2 Oriëntatie

Volgens de eindtermen van het vak informatica zijn er de volgende voorbeelden van oriëntaties:
- *gegevensoriëntatie*
- *procesoriëntatie*
- *objectoriëntatie*
- *taaloriëntatie*
- *programmeerparadigma's.*

Wat opvalt in het hoofdstuk van Enigma is dat er niet een duidelijke oriëntatie naar voren komt. Het hoofdstuk is voornamelijk taal-georiënteerd. Er wordt weinig object-georiënteerd geprogrammeerd, terwijl Java daar juist geschikt voor is. Ook de gegevensoriëntatie wordt niet in de hand genomen, die juist bij een inleidend programmeerhoofdstuk houvast zou kunnen bieden bij het begrijpen van nieuwe begrippen.
Het is een aanbeveling om nieuwe begrippen op een concrete manier te introduceren en te koppelen aan begrippen die een lezer of leerling al kent.

3.3 Ordening van leerinhoud

Bij de analyse van het hoofdstuk van Enigma kwam naar voren dat de lesstof voornamelijk lineair geordend is. De website van de Universiteit van Leuven[5] geeft enkele nadelen aan van de lineaire ordening:
”*Het gevaar is reëel dat het streven naar volledigheid leidt tot overlading bij lerenden, omdat men zoveel mogelijk onderwerpen aan de orde wil stellen. Wat de lerenden over elk onderwerp meedragen, is dan noodzakelijk beperkt. Zo ontstaat een eerder encyclopedische kennis: de leerlingen of studenten weten iets over een groot aantal onderwerpen, maar kennen niets echt grondig.“*
Het is een aanbeveling om na te denken over de volgorde van leerinhoud en het hoofdstuk te herstructureren. Het is misschien mogelijk een iets grotere praktische opdracht al eerder in het hoofdstuk te specificeren zodat leerlingen in een vroeg stadium zien wat er mogelijk is met de taal Java. Misschien dat een simpel spelletje als boter-kaas-en-eieren gemaakt kan worden.

Hoofdstuk 4: Leren programmeren met Greenfoot

Naast de methodes die in de vorige hoofdstukken zijn beschreven zijn er nog andere leermethodes. Er bestaat al enige tijd een nieuwe leermethode, Objects First. In deze methode staat de context, de wereld centraal. Er zijn verschillende onderzoeken naar deze methode gedaan [6] [7].

Greenfoot is een programmeeromgeving die gebruik maakt van de leermethode Objects First. Greenfoot is een educatieve programmeeromgeving die de nadruk legt op een visuele weergave van een 2-dimensionale wereld. Het is ontwikkeld door de makers van de programmeeromgeving BlueJ. [8]

Om te leren programmeren met Greenfoot is er een opdrachtenreeks ontwikkeld door de Radboud Universiteit. Deze opdrachten zijn oorspronkelijk bedoeld om als ondersteuning en oefening te dienen bij een lessenreeks waarmee object georiënteerd programmeren wordt geleerd. Er zijn enkele experimenten gedaan als case study om te kijken welke uitleg nodig is om deze opdrachten te kunnen maken. Het doel van het experiment was om de opdracht “self contained” te maken. Dat wil zeggen, de opdracht dusdanig uitbreiden dat een docent er zelfstandig mee uit de voeten zou kunnen. De proefpersonen hadden beide geen noemenswaardige programmeerervaring en waren dus beginners op programmeergebied.

4.1 Eerste experiment

Bij het eerste experiment kreeg de proefpersoon alleen de opdrachten en de beschikking over de programmeeromgeving. De proefpersoon kreeg de opdracht om de programmeeropdracht zoveel mogelijk zelfstandig te maken, en alleen hulp te vragen als het probleem echt niet op te lossen was. Het experiment werd opgenomen om later te kunnen achterhalen op welke punten de proefpersoon vastliep.

Sessie 1
De eerste opdracht was bijna geheel zelfstandig uit te voeren. Begrippen die lastig waren te begrijpen waren de types boolean en void. Verder kwamen er wat kleine foutjes in de opdracht naar voren.

Sessie 2
De tweede opdracht ging moeizaam. Veel nieuwe begrippen kwamen aan bod, en werden vaak niet uitgelegd in de opdracht. De begrippen attribuut en constructor werden als lastig ervaren. De code die bij deze opdracht gebruikt werd was ook niet altijd even helder opgesteld. Er werd vele malen om hulp gevraagd en sommige delen van de opdracht waren zelfs met extra uitleg lastig te voltooien.

Na deze twee sessies zijn de opdrachten bijgewerkt, naar aanleiding van de moeilijkheden en opmerkingen van de eerste proefpersoon. Tevens is er een bijlage

geschreven die als handleiding gebruikt kan worden bij de opdrachten (zie Appendix I). Deze bijlage kan eveneens dienen als handleiding voor een docent om een overzicht te hebben van de begrippen en concepten die een student moet begrijpen om de opdrachten te kunnen maken.

4.2 Tweede experiment

Bij het tweede experiment werd gebruik gemaakt van de bijgewerkte opdracht en was de handleiding beschikbaar om in te zien. In het experiment werd gebruik gemaakt van de methode “hardop denken”[9]. Dit hield in dat de proefpersoon het denkproces inzichtelijker maakt door hardop te denken. Het experiment werd opgenomen om later een duidelijk overzicht te hebben van de punten waarop de proefpersoon vastliep en de reden van het vastlopen te kunnen vaststellen.

Sessie 1
De opdracht werd geheel zelfstandig en succesvol afgerond door de tweede proefpersoon. De bijlage werd een aantal keer geraadpleegd, echter werden niet alle onderdelen van de bijlage gebruikt.

Sessie 2
De tweede opdracht was duidelijk een stuk lastiger. De proefpersoon vroeg op 4 momenten om hulp:

- Bij attributen. De proefpersoon had de opdracht niet goed genoeg gelezen en had daardoor een verkeerde aanpak gebruikt waarin zij vastliep.
- De concepten bij de methode *hatchEgg* bleven lastig, ook met de extra uitleg van de handleiding erbij. De code werd niet echt doorgrond en daardoor was de code ook lastig aan te passen.
- Een zelfgemaakte methode levert een onjuist resultaat op. De reden is een klassieke fout in een if-statement, namelijk:
  ```
  if(hatched=false)
  ```
  Aangezien de compiler hier niet voor waarschuwt is deze fout lastig te vinden. In de bijlage werd hier ook niet voor gewaarschuwd.
- Het aanpassen van het if-statement in de methode *foundEgg* lukt niet zonder hulp. Hier miste uitleg over *!=* en het keyword *null*. Tevens leveren wederom de concepten van punt twee problemen op. Het idee dat er een methode van een andere klasse moet worden aangeroepen is duidelijk, maar het lukt de proefpersoon niet om te achterhalen hoe dat syntactisch gezien moet worden opgeschreven.

Omdat bij het maken van de handleiding het boek van Michael Kölling als naslagwerk wordt gebruikt heeft de proefpersoon na het maken van de tweede opdracht nog een extra opdracht gemaakt. Dit was een controle-opdracht waarbij het gedrag van de kip moest worden aangepast. Er moest een kleine kans worden ingebouwd om rechtsom te draaien. Deze opdracht werd met succes afgerond.

4.3 Afronding

Naar aanleiding van het tweede experiment zijn er nog kleine wijzigingen doorgevoerd in de tekst van de opdracht en de handleiding. Een van de paragrafen is uit de handleiding gehaald omdat deze overbodig bleek. De opdracht is iets helderder gemaakt op het punt waar de proefpersoon de opdracht verkeerd interpreteerde. Verder zijn er een aantal punten toegevoegd aan de handleiding die nog misten.

Conclusie

De oorspronkelijke onderzoeksvraag die aan dit onderzoek ten grondslag lag was specifiek gericht op de Enigma methode. Uiteindelijk is ervoor gekozen het onderzoek uit te breiden en ook te kijken naar een nieuwere methode om het onderzoek interessanter te maken.
Het is niet gemakkelijk een objectieve analyse uit te voeren van een lesmethode. Er is geen standaard manier om lesstof te beoordelen. Het beoordelen van een methode kan op veel verschillende manieren, want er kan naar allerlei facetten worden gekeken van de methode. Ook het vakgebied heeft invloed op de beoordeling, want bepaalde facetten zijn in één vakgebied belangrijker dan in een ander vakgebied.
In hoofdstuk twee is naar voren gekomen dat er punten zijn waarop het hoofdstuk van Enigma verbeterd kan worden. Hopelijk zullen deze aanbevelingen worden meegenomen in een eventuele nieuwe uitgave van de methode.
De Objects First methode die gebruikt wordt door Greenfoot is naar mijn mening veelbelovend. De proefpersonen die in dit onderzoek meewerkten waren erg enthousiast over de opdracht. Het principe dat eerst visueel wordt gekeken hoe het gedrag van een bepaald object is en dat pas daarna wordt gekeken naar de code die daarbij hoort is een manier die mogelijk goed aansluit bij beginnend programmeurs.
Enigma en Greenfoot verschillen veel in aanpak. De vraag is of de doelen van beide leermethodes ook verschillend zijn. Het doel van de methode van Enigma wordt niet genoemd in het voorwoord van het boek. Het is waarschijnlijk dat het doel van de methode is een poging te doen leerlingen het vak informatica te leren zoals wordt verwacht uit de eindtermen die voor het vak zijn gespecificeerd. De doelen van het hoofdstuk over programmeren staan genoemd in het verwerkingsboek. Deze doelen omvatten kennis van de basisbeginselen van het programmeren in Java.
Het doel van het boek dat bij Greenfoot hoort is gespecificeerd in de inleiding. Er wordt geschreven: "*Dit boek heeft meerdere doelstellingen: een daarvan is de lezer leren programmeren, een andere is de lezer daar ook plezier aan te laten beleven*." Greenfoot specificeert dus ook een affectief doel, en niet alleen cognitieve doelen.
Omdat de doelen van Enigma en Greenfoot verschillen is het niet mogelijk om deze methodes eerlijk met elkaar te vergelijken.

Vervolgonderzoek

Een van de punten van analyse die niet is behandeld is te kijken in hoeverre er rekening wordt gehouden met de leerstijlen van Kolb. Hiermee zou de analyse kunnen worden uitgebreid en daarmee een beter beeld geven van het hoofdstuk.
Het experiment dat is gedaan met Greenfoot is twee keer uitgevoerd met twee verschillende proefpersonen. Hiervoor is gekozen om de ontwikkelcyclus van de opdrachten kort te houden, na elk experiment kon de opdracht verbeterd worden. Tevens neemt het maken van een programmeeropdracht veel tijd in beslag, zeker bij proefpersonen die nog geen programmeerervaring hebben. Nu de opdrachten zijn bijgewerkt zou het mogelijk zijn om een groter experiment te doen meer meerdere proefpersonen.

Bronnen

[1]*Report by the ITiCSE 2001 Working Group on Assessment of Programming Skills of First-year CS Students*, McCracken et al., 2001

[2] *11 redenen waarom programmeren zo moeilijk is*, Nico van Diepen

[3*] A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition*, Jerry Mead, Simon Gray, John Hamer, Richard James, Juha Sorva, Caroline St. Clair, Lynda Thomas, 2006

[4] *Portray the Pedagogical Content Knowledge of Programming, Technical Report*, W.M.G. Jochems, G. Zwaneveld, J. Perrenet, en M. Saeli.

[5] *Volgorde van leerinhoud*, Katholieke Universiteit Leuven https://www.kuleuven.be/algdid/difdyn1.php3?klikt=cmaps3tli312-01&gr=1

[6] *Teaching Objects-first In Introductory Computer Science*, Stephen Cooper, Wanda Dann, Randy Pausch, 2003

[7] *What does "Objects-First" Mean? An International Study of Teachers' Perceptions of Objects-First*, Jens Bennedsen & Carsten Schulte, 2008

[8] G*reenfoot: Combining Object Visualisation with Interaction*, Poul Henriksen & Michael Kölling, 2004

[9] *The Think Aloud Method: A practical guide to modelling cognitive processes*, Maarten W. van Someren, Yvonne F. Barnard, Jacobijn A.C. Sandberg, 1994

Appendix A: opdracht 1
Objectgeoriënteerd Programmeren in Java

Opdracht 1: Maak kennis met Frida

1. Achtergrond
In deze opgavenreeks leer je over Objectgeoriënteerd programmeren en hoe je dat systematisch aanpakt. Je programmeert in de taal Java. Je maakt kennis met de basisbouwstenen waarmee je algoritmen en programma's maakt. In de opdrachten werken we met de Greenfoot-programmeeromgeving. In de eerste opdracht leer je werken met objecten en methoden in een virtuele wereld.

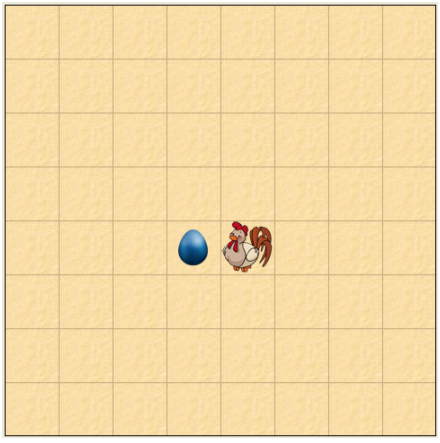
2. Leerdoelen
Na het voltooien van deze opdracht kun je:

- De programmeeromgeving '*Greenfoot'* gebruiken:
  - De knoppen Act, Run, Compile, new .., Open Editor weten te vinden en te weten wanneer welke knop gebruikt moet worden.
  - Objecten kunnen toevoegen aan de wereld en methodes kunnen aanroepen bij deze objecten.
- Uitleggen hoe invoerwaarden en resultaat-types werken.
- De betekenis van een klassediagram uitleggen in termen van methoden en overerving.
- Programmacode aanpassen(gegeven een handleiding), compileren en uitvoeren in Greenfoot.
- Aangeven welke soorten foutmeldingen Greenfoot kan genereren.

3. Instructies

3.1. Kippen en eieren

Hiernaast zie je onze kip Frida.
Het digitale materiaal bij deze opdracht is nog even Engelstalig. Daarom vind je in Greenfoot de Amerikaanse nicht van Frida. Die heet Fryde en woont in Kentucky. Om verwarring te voorkomen gebruiken we vanaf nu de Engelse naam.



Fryde leeft in een eenvoudige virtuele wereld die bestaat uit 8 bij 8 hokjes. Ze brengt haar dag door met het leggen van eieren en het broeden op die eieren zodat ze uitkomen (en verdwijnen van de virtuele wereld). In het plaatje hiernaast zie je Fryde met haar snavel naar het westen gericht. Voor haar ligt een ei.

Fryde is een heel brave kip. Ze doet altijd precies wat haar wordt gezegd en geeft altijd antwoord op je vragen (en ze liegt nooit). Opdrachten geven en vragen stellen doe je via **methoden**.

Je kunt een kip verschillende dingen laten doen. Deze methoden zijn opdrachten (of **commando's**), zoals

- *move()* stap een hokje verder
- *turnLeft()* draai een kwartslag naar links
- *layEgg()* leg een ei
- *hatchEgg()* broed een ei uit

De overige methoden zijn vragen (**queries** in het Engels), zoals

- *facingNorth()* Wijst je snavel naar het noorden?
- *foundEgg()* Heb je een ei gevonden?
- *fenceAhead*() Staat er een hek voor je?

3.2 Aan de slag met Greenfoot

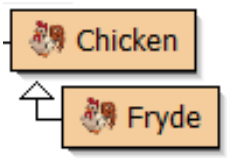
Start Greenfoot. Selecteer 'Scenario' in het hoofdmenu en daarna 'Open'. Navigeer naar het scenario 'chickenworld' en kies 'Open'.

1. Maak het scenario dat je in het plaatje hierboven ziet:
    a. Klik met de rechtermuisknop op Fryde.
    b. Kies *new Fryde()* om een nieuw Fryde object te maken. Als deze optie niet beschikbaar is, klik dan eerst onderaan op de knop "Compile" en probeer het opnieuw.
    c. Plaats het object, onze kip Fryde, in de wereld.
    d. Op dezelfde manier leg je een ei voor Fryde neer.

2. Onze kip is geprogrammeerd om een aantal dingen te doen.
    a. Zet een Fryde in de wereld neer. Klik er met de rechtermuisknop op. Zo zie je wat je Fryde kunt laten doen (met andere woorden: welke methoden je kunt aanroepen).
    b. Wat gebeurt er als je de methode *boolean eggAhead ()* aanroept? Denk je dat deze methode altijd 'true' oplevert? Verplaats Fryde of het ei tot je een ander antwoord krijgt.
    c. Verplaats Fryde en het ei zodat het ei precies voor Fryde ligt. Wat gebeurt er als je de methode *void act()* aanroept?
    d. Welk antwoord verwacht je als je Fryde zou vragen of er een ei voor haar ligt? Probeer het eens. Je ziet dat deze methode een *boolean* ('true' of 'false', waar of onwaar) oplevert.
    e. Wat gebeurt er als je *turn ()* aanroept? In welke richting wijst haar snavel (North, East, South of West)? Probeer de methode *boolean facingNorth()* (die kun je vinden tussen de methoden die zijn 'geërfd' van *Chicken*; deze 'overerving' wordt verderop uitgelegd) en verklaar het resultaat.
    f. Roep nu de methode *turnLeft()* aan, net zolang tot Fryde naar het noorden kijkt. Controleer het door haar te vragen of ze naar het noorden kijkt. Wat antwoordt ze?
    g. Klik met de rechtermuisknop op Fryde en kies 'Inspect'. Op welke coördinaten (x en y) staat Fryde?
    h. Bekijk het hokjesrooster. Wat zijn de coördinaten van het hokje in de linkerbovenhoek? En van de rechterbovenhoek? Verplaats Fryde naar deze hokjes en controleer je antwoorden.
    i. Wat zou Fryde doen als ze in een hoek staat met haar snavel naar buiten gericht en je de methode *act()* aanroept? Probeer het eens.

Lees nu §1 van de handleiding.

3.3 Overerving



1. Door rechts te klikken op een Fryde in de wereld kun je de dingen zien waar Fryde goed in is. Maar Fryde kan ook de dingen doen die elke kip kan: ze is tenslotte een kip. Je kunt dat in het klassediagram zien, rechts op het scherm. De pijl geeft een 'is-een' relatie aan, dus Fryde is een kip: Fryde is een subklasse van Chicken. Welke andere 'is-een' relaties zie je? Noem er tenminste twee.
2. Omdat Fryde een kip is, krijgt ('erft') ze alle kip-methoden.
    a. Klik rechts op Fryde (in de wereld) en bekijk welke Chicken-methoden ze erft. Noem er minstens drie.
    b. Probeer de methode *int getEggsHatched()*. Deze methode kun je gebruiken om te zien hoeveel eieren deze kip heeft uitgebroed. Deze methode levert een geheel getal op: een integer, kortweg *int*.
    c. Kun je een situatie maken zodat het resultaat van deze methode 3 is? (Met andere woorden, kun je je kip drie eieren laten uitbroeden?)
    d. Verplaats het ei (of de kip) zodat er een ei precies voor haar ligt en haar snavel naar het ei wijst. Vraag Fryde of er een ei voor haar ligt. Tip: je kunt Fryde verplaatsen met de muis; klik op Fryde, houdt de linkermuisknop ingedrukt en sleep haar naar een ander vakje.
    e. Vraag Fryde hoeveel eieren ze heeft uitgebroed. Schrijf het antwoord op.
    f. Vraag Fryde om een stap naar voren te doen met de methode *void move()*.
    g. Vraag Fryde nogmaals hoeveel eieren ze heeft uitgebroed. Schrijf het antwoord weer op.
    h. Herhaal de stappen d, e, f. Roep dan de methode *void act()* aan. Wat is het verschil tussen *void act()* en *void move()?*

3. Fryde kan niet over een hek springen of vliegen.
    i. Bouw een hek in de wereld. Controleer dat Fryde inderdaad niet over de hek kan komen door de methoden *move()* en *turn()* aan te roepen. Tip: Je kunt een hek weer weghalen door met de rechtermuisknop op een hek te klikken en voor "Remove" te kiezen.

3.4 Greenfoot uitvoeren

1. Leg een boel eieren neer in de wereld en roep Frydes methode *act()* een aantal keren aan. Experimenteer met de eieren op verschillende plekken. Wat doet de methode *act()* precies? Zorg dat je verschillende situaties probeert, bijvoorbeeld met Fryde aan de rand van de wereld met haar snavel naar buiten gericht, of terwijl ze voor, boven of bovenop een ei staat. Tip: je kunt Fryde met de muis verplaatsen – dat maakt het testen een stuk gemakkelijker.

2. Beschrijf zo nauwkeurig mogelijk wat de methode *act()* volgens jou doet.

3. Greenfoot heeft een aantal knoppen onderaan het venster.
    a. Herstel de wereld door op de knop 'Reset' te drukken.
    b. Plaats enkele kippen en eieren in de wereld.
    c. Klik op de 'Act' knop. Wat gebeurt er?

d. Wat gebeurt er als je er nog eens op klikt? En nog eens?
e. Wat is het verschil tussen het klikken op de ‘Act’ knop en het aanroepen van de *act()* methode?
f. Druk op de ‘Run’ knop. Wat gebeurt er?

3.5 Invoerwaarden en resultaat-types

1. Roep de methode *setDirection(int direction)* aan. Aan deze methode geef je een getal mee dat een richting (‘direction’) aangeeft. Toets een invoerwaarde in en kijk wat er gebeurt. Welk getal komt overeen met welke richting? Noteer ze. Wat gebeurt er als je een waarde groter dan 3 invoert? En wat gebeurt er als je een waarde invoert die geen integer is, bijvoorbeeld 2.5, of een woord, bijvoorbeeld ‘twee’)? Verklaar wat je ziet.
2. Sommige methoden hebben resultaat-types als *boolean* en *int*. Wat zou het resultaat-type *void* betekenen?

Lees nu §2 van de handleiding.

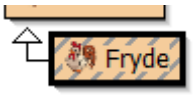
3.6 Aanpassen, compileren en testen van programma’s

1. Het lezen van programmacode
    a. Klik rechts op Fryde in het klassediagram en kies ‘Open editor’.
    b. Zoek de methode *act()* op. Toelichting:
        - de typering (‘signature’) van de methode is *public void act()*
        - het resultaat-type is *void*; dat betekent dat de methode geen waarde oplevert
        - een *public* methode kun je aanroepen door rechts te klikken op het object als het in de wereld is geplaatst
        - een *private* methode is van buitenaf (in de wereld) onzichtbaar
        - het enige dat *act()* doet is het aanroepen van de methode *hatchEggs()*
        - alles tussen /* en */ is commentaar; in de editor wordt deze tekst in blauw weergegeven. Greenfoot doet niets met dat commentaar. Programmeurs schrijven commentaar boven en in hun methoden om anderen te helpen hun programma’s te begrijpen.
    c. Zoek de code van de methode *hatchEggs()* op. Toelichting:
        - het symbool ‘!’ betekent ‘niet. Dus: als egg*Ahead()* waar is (‘true’), dan is *! eggAhead()* onwaar (‘false’).
        - deze methode is *private*, dus kan alleen gebruikt worden binnen de klasse en kan niet via het rechtermuismenu in de wereld worden aangeroepen
    d. Lees de code van deze methode. Wat doet *hatchEggs()* precies?
    e. Voeg met /* en */ commentaar toe (in gewoon Nederlands) aan het einde van elke coderegel en leg zo uit wat er in die regel gebeurt.
    f. Gebruik /* en */ ook om commentaar boven de methode *hatchEggs()* te plaatsen om uit te leggen wat de methode doet.
    g. Vergelijk je resultaten met je antwoord op vraag 3.4.2.
    h. Maak een lijst van alle *public* methoden in deze klasse.
    i. Maak een lijst van alle *private* methoden in deze klasse.

2. Code toevoegen en compileren
    a. Open de code van de klasse Fryde in de editor.
    b. Voeg een nieuwe methode toe aan deze klasse door de volgende tekst onderin het editorscherm te typen (maar wel voor de laatste '}', anders valt de methode buiten de klasse):

```
public void jump() {
    move();
    move();
}
```

    c. Wat doet deze methode? Zet een beschrijving als commentaar boven de methode.
    d. Als je de editor sluit, zie je dat de klasse er anders uitziet: het blokje is grijs gearceerd. Dat betekent dat de code veranderd is en nog moet worden gecompileerd. Druk op de knop 'Compile' om alle klassen opnieuw te compileren. Herstel zonodig de fouten die de compiler meldt.



    e. Roep je *jump* methode aan en test of die werkt zoals verwacht.

3. Foutmeldingen. De compiler is erg kieskeurig. Soms maak je een foutje of vergeet je iets – dan zal de compiler daarover klagen. Het is handig als je enkele veel voorkomende klachten herkent, zodat je het probleem gemakkelijk kunt vinden en oplossen. Laten we er een paar bekijken.
    a. Open de klasse Fryde in de editor. Verwijder de ';' in de methode *act()*. Sluit de editor en compileer. Welke foutmelding zie je onderin het editorscherm?
    b. Herstel de ';' en compileer opnieuw. Dat zou nu probleemloos moeten gaan.
    c. Verander iets in de spelling van *hatchEggs()* en compileer. Welke foutmelding krijg je?
    d. Herstel de fout en compileer opnieuw.
    e. Verander *hatchEggs()* in *hatchEggs(5)*. Welke foutmelding krijg je? Wat betekent de melding?
    f. Verwijder de 5 en compileer opnieuw. Werkt het nu weer?

4. Afronden

Je bent klaar met de eerste opdracht. Bewaar je werk, want je hebt het nodig voor de volgende opdrachten. Sla je werk op zodat je er later gebruik van kunt maken. Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save a copy as'. Vul de bestandsnaam aan met je eigen naam en het opgavenummer, bijvoorbeeld 'Fryde_Michel_opdr1'.

Appendix B: opdracht 2
Objectgeoriënteerd Programmeren in Java

Opdracht 2: Het maken van eigen code

1. Achtergrond
In de vorige opdracht heb je kennis gemaakt met Greenfoot. Nadat je de basis hebt geleerd kun je nu zelf code gaan schrijven.

2. Leerdoelen
Na het voltooien van deze opdracht kun je:

- Een simpele nieuwe methode ontwerpen en implementeren.
- Het doel en de werking van attributen begrijpen.
- Een nieuw attribuut aan een klasse toevoegen.
- Methodes ontwerpen en implementeren die een attribuut aanpassen en die de inhoud van een attribuut als resultaat opleveren.
- Het doel en de werking van een constructor begrijpen.
- Een simpele constructor implementeren.
- Een public methode kunnen aanroepen vanuit een andere klasse, gebruikmakend van de punt-notatie.
- Een gecombineerd if-statement ontwerpen door gebruik te maken van && of ||.
- Resultaat-types van methodes gebruiken.

3. Instructies

Nu je de basis van Greenfoot hebt gezien, gaan we de methodes gebruiken om complexere algoritmes te ontwikkelen en testen. Je kunt een nieuw scenario genaamd ‘chickenworld2’ openen (opgeslagen in de ‘day 2’ map).

3.1. Je eigen methode
1. Schrijf je eigen methode om Fryde naar rechts te draaien:
    - Kijk naar de beschikbare methodes die Fryde heeft (ook de methodes van Chicken die Fryde heeft ge-inherit). Welke methodes zou je kunnen gebruiken om Fryde naar rechts te draaien? Tip: om de beschikbare methodes te zien, plaats Fryde in de wereld en klik met de rechtermuisknop op haar.
    - Open de editor van Fryde. (let op: open niet Chicken!)
    - Gebruik de methodes van opdracht a) om een methode *public void turnRight()* te maken waarmee Fryde 90 graden met de klok mee draait. Vergeet niet commentaar boven je methode toe te voegen..
    - Compileer en test je methode. Werkt het zoals je had verwacht?

Lees nu §4 en § 5 uit de handleiding.

Nadat een ei is uitgebroed verdwijnt hij van de wereld. In plaats van hem te laten verdwijnen willen we een plaatje weergeven! We gaan het volgende plaatje gebruiken:

Dit ziet er veel realistischer uit. Dus, als Fryde een ei uitbroedt, zou het plaatje moeten veranderen.

2. Laten we eerst de Egg klasse aanpassen zodat hij onthoudt of hij is uitgebroed of niet. Als we een object iets willen laten onthouden slaan we dit op in een attribuut (eigenschap). In dit geval gaan we een attribuut van type *boolean* toevoegen met naam '*hatched*'. Het is netjes om attributen altijd *private* te maken.

a. Bekijk de klasse Fence in de editor. Op welke plek horen attributen te staan?
b. Open de Egg klasse in de editor en voeg het nieuwe attribuut *hatched* toe.
c. Attributen horen altijd de goede waarde te hebben, ook meteen nadat een object is aangemaakt. Een attribuut een start-waarde geven noemen we (attribuut)*initialisatie*. Normaliter heeft een klasse een speciale methode, genaamd de *constructor*, die de attributen initialiseert. Deze constructor(-methode) kan gemakkelijk herkend worden, hij heeft dezelfde naam als de klasse en geen resultaattype. In de Egg klasse staat al een constructor, die nog niks doet, de inhoud is nog leeg. Welke waarde moet *hatched* krijgen in het begin? (*true* of *false*?) Geef *hatched* de juiste waarde in de constructor.
d. Dit is nog niet genoeg. We moeten ook de een ei wat nog niet is uitgebroed kunnen uitbroeden. In andere woorden, de Egg klasse moet worden uitgebreid met een methode die dat regelt. Laten we deze methode *hatch* noemen. Aangezien *hatch* alleen iets in het ei gaat veranderen hoeft deze methode niets op te leveren(geen resultaat). Hoe geven we dat aan? Bedenk ook of de methode public of private moet worden. Schrijf de methode op, en laat de inhoud nog even leeg.
e. We gaan nu de *hatch* methode inhoud geven. Als eerste moet *hatch* de waarde van het attribuut aanpassen. Dat kun je op dezelfde manier doen als in de constructor gebeurt.
f. Als tweede moet het zichtbaar zijn op het scherm dat het ei is uitgebroed. Dit kun je doen door het originele plaatje (een blauw ei) te veranderen in een rood, uitgebroed ei. Dit kun je doen door de volgende instructies:

```
GreenfootImage hatched_egg = new GreenfootImage("egg_hatched.png");
setImage(hatched_egg);
```

Lees nu §6, 7 en 8 uit de handleiding.

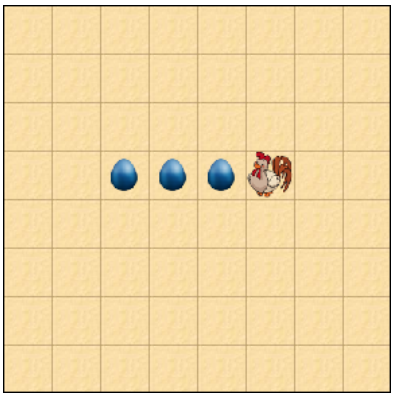
3. Nu de hatch methode af is kunnen we de laatste dingen aanpassen om onze nieuwe toevoeging werkend te krijgen:

a. Er is nog een methode handig als we deze klasse willen gebruiken. Het zou handig zijn om aan een ei te kunnen vragen of hij al uitgebroed is. Schrijf een methode *public boolean isHatched()* die de waarde van het *hatched* attribuut teruggeeft.
b. Als Fryde op een ei zit, zou dit ei moeten veranderen in een uitgebroed ei (door *hatch* aan te roepen). Nu gaan we kijken welke methode we daarvoor moeten aanpassen. Bekijk de klasse *Chicken*. Welke methode gaan we aanpassen?
c. Bekijk de methode. Wat doet elke regel? Schrijf commentaar bij elke regel met uitleg wat er gebeurt.

d. Om een ei te veranderen in een uitgebroed ei, moeten we het ei niet uit de wereld verwijderen maar de status aanpassen door de *hatch* methode aan te roepen van het ei. Verwijder de code die niet meer nodig is en voeg de regel toe waarin de *hatch* methode wordt aangeroepen.
e. Compileer en test of het werkt.

3.2. Fryde moet eieren niet opnieuw uitbroeden: !, &&, ==

Fryde is altijd op zoek naar eieren om uit te broeden. Maar als een ei al is uitgebroed, zou ze hem moeten negeren.



a. Bekijk de methode *public boolean foundEgg()* in de klasse Chicken. Wat doet deze methode?
b. Leg een rij met eieren voor Fryde neer en voer het scenario uit met de knop "Run". Wat gebeurt er?
c. Laten we de methode *foundEgg*() aanpassen zodat deze true teruggeeft als hij een ei vind die nog niet is uitgebroed. Voordat we de code gaan aanpassen, veranderen we eerst de naam van de methode in *foundUnhatchedEgg*() en zorgen we dat het commentaar erboven weer klopt.
d. Compileer je code om te zien of alles nog werkt.
e. De compiler zou moeten klagen omdat in de Fryde klasse nog steeds *foundEgg*() staat in de methode eggAhead(). Hier moet Fryde ook alleen weten of er een ei is dat nog niet is uitgebroed. Verander de methode aanroep naar *foundUnhatchedEgg*().
f. Laten we nu teruggaan naar de methode *foundUnhatchedEgg*(). Verander de methode zodat hij alleen true teruggeeft als het ei dat is gevonden nog niet uitgebroed is.
g. Compileer en test je code. Plaats Fryde in hetzelfde vakje als een niet uitgebroed ei, klik met de rechtermuisknop op haar en bekijk het resultaat van *foundUnhatchedEgg*() (die in de Chicken klasse staat). Welk resultaat levert de methode op? Verplaats Fryde ook naar een vakje met een uitgebroed ei. Wat voor resultaat geeft het nu?
h. Voer opdracht b opnieuw uit.

4. Afronden

Je bent klaar met de eerste opdracht. Bewaar je werk, want je hebt het nodig voor de volgende opdrachten. Sla je werk op zodat je er later gebruik van kunt maken. Kies in Greenfoot 'Scenario' in het bovenste menu, en dan 'Save a copy as'. Vul de bestandsnaam aan met je eigen naam en het opgavenummer, bijvoorbeeld 'Fryde_Michel_opdr2'.

Appendix C: Opdrachtenhandleiding

Deze handleiding is bedoeld als hulp bij de opdrachten. Hij kan ook als overzicht gebruikt worden voor docenten om te zien welke lesstof uitgelegd moet worden. De extra uitleg waarnaar gerefereerd wordt is te vinden in de Nederlandse bewerking van het boek “Programmeren in Java met Greenfoot” van Michael Kölling.

§1. Klassen en objecten

Uitleg over objecten en klassen is te vinden in paragraaf 1.2 op bladzijde 2.

Een klasse vertegenwoordigt een algemeen concept van een ding uit de echte wereld. In de opdracht hebben we bijvoorbeeld de klasse Chicken en de klasse Egg. De klasse Egg beschrijft dus eigenlijk alle eieren. Als we een klasse hebben dan kunnen we daar een object van aanmaken. We kunnen bijvoorbeeld nieuw ei maken, we hebben dan een ei-object. Met een klasse kun je zoveel objecten aanmaken als je wilt.

§2. Types:

Uitleg over resultaat-types is te vinden in paragraaf 1.4 op bladzijde 5.

Boolean:

Boolean is een type die alleen true of false kan zijn (waar of onwaar). Dit type kan gebruikt worden bij een methode om iets te vragen wat alleen true of false kan opleveren, oftewel een ja/nee vraag.

Int:

Int is de afkorting voor integer. Dit is een type dat een geheel getal in zich heeft, bijvoorbeeld 0 of 1 of 2. Een int kun je gebruiken bij een methode om iets te vragen wat een hoeveelheid oplevert.

Void:

Void is het “lege” type. Void betekent leeg. Een methode die als type void heeft levert niks op. Dit kan bijvoorbeeld een methode zijn om te bewegen of te draaien.

§3. Public en Private

Uitleg over private methodes is te vinden in paragraaf 6.6 op bladzijde 99.

Public

Public methoden kunnen worden aangeroepen door op de rechtermuisknop te drukken en daar de methode te kiezen. Andere klassen kunnen ook public methodes aanroepen.

Private

Private methoden zijn als het ware “geheim” en kunnen niet aangeroepen worden vanuit de wereld of vanuit een andere klasse. Een private methode kan alleen aangeroepen worden door een andere methode in dezelfde klasse.
Bijvoorbeeld: we hebben klasse A die twee methodes heeft: open() en geheim(). Methode open() is public en methode geheim() is private. Stel dat methode open() methode geheim() aanroept. Als we dan de private methode geheim() willen uitvoeren dan kunnen we dat alleen doen via methode open(). Een andere klasse kan dus nooit de private methode geheim() aanroepen behalve via de public methode open().

§4. ‘Slimme’ objecten

In de eerste opdracht hebben we aan Fryde gevraagd hoeveel eieren ze had uitgebroed. Misschien heb je toen al opgemerkt dat dat vreemd is, een kip in de echte wereld onthoudt dat immers niet, en als de kip het zou onthouden dan kun je het lastig vragen. In deze opdracht gaan we verder met het ei, en ook aan het ei gaan we dingen vragen, en het ei opdrachten geven. Dit lijkt misschien vreemd, maar het is goed om te onthouden dat in het programma **alle** objecten “intelligent” zijn. Ook objecten die in de echte wereld weinig kunnen (een boom, een steen, enz.) kunnen in de computer soms hele slimme dingen doen.

§5. Klassen

Een klasse heeft normaliter 3 dingen in zich:
* Attributen
* Constructor
* Methodes

Attributen

Uitleg over attributen is te vinden in paragraaf 4.5 op bladzijde 53. In het boek worden attributen instantievariabelen genoemd.

Attributen zijn eigenschappen van de klasse. Een attribuut kan iets onthouden. Stel we hebben een klasse Auto, dan kunnen attributen bijvoorbeeld zijn: merk, kleur, aantalDeuren, heeftTrekhaak. (heeftTrekhaak zou van type boolean kunnen zijn, of de auto wel of geen trekhaak heeft).

Constructor

Uitleg over constructors is te vinden in paragraaf 4.7 op bladzijde 55.

De constructor is een methode die altijd maar 1x wordt aangeroepen, namelijk als het object wordt aangemaakt. In Greenfoot wordt deze methode dus aangeroepen als je op “new Fryde” drukt. De constructor wordt gebruikt om de attributen de juiste waarde te geven.

Methodes
Zoals al in de opdracht wordt uitgelegd, met methodes kun je opdrachten geven en vragen stellen aan een object.

§6. if-statement

Uitleg over het if-statement is te vinden in paragraaf 4.5 op bladzijde 58.
Een if-statement ziet er normaal gesproken zo uit:

```
if (voorwaarde) {
        actie();
        actie();
}
else {
        andereActie();
}
```

Op de plaats van *voorwaarde* moet iets staan van type boolean. Hier kan bijvoorbeeld een methode staan die een boolean oplevert, of een boolean variabele.
Je kunt ook twee waardes combineren door een logische samenvoeging of vergelijking. We hebben daarvoor verschillende mogelijkheden:
Vergelijking van twee waarden:
== betekent gelijk aan
!= betekent ongelijk aan
< betekent kleiner dan
> betekent groter dan
Samenvoegingen
&& betekent EN
|| betekent OF
Voorbeeld:

```
if(heeftVierWielen() && jeKuntErInRijden ) {
        antwoord = auto;
}
```

In het bovenstaande voorbeeldje is heeftVierWielen() een methode (dat kun je zien aan de twee haakjes) en jeKuntErInRijden is een variabele.

Valkuil:
Een fout die vaak gemaakt wordt is dat er if (A=B) wordt geschreven in plaats van if (A==B).
Onthoudt dat als je twee waardes wil vergelijken je altijd == moet gebruiken!

§7. Variabelen in methodes

We willen soms in een methode variabelen gebruiken om tussenresultaten op te slaan. Stel dat we een methode willen maken waarmee we het aantal eieren willen tellen die twee kippen hebben uitgebroed. De onderstaande methode gebruikt niet helemaal correcte code maar is bedoeld om het concept te begrijpen:

```
public int telEieren () {
        int eieren1 = telEierenKip1();
        int eieren2 = telEierenKip2();
        int totaal = eieren1+ eieren2;
        return totaal;
}
```

Uitleg: De methode is van type int, het levert een getal op. In de eerste regel van de methode maken we een *int* variabele aan met naam *eieren1*. We voeren de methode *telEierenKip1()* uit en stoppen het resultaat in *eieren1*. De tweede kip gaat op dezelfde manier. In de derde regel worden de twee getallen bij elkaar opgeteld, en in de nieuwe int variabele *totaal* gestopt. In de vierde regel leveren we de variabele *totaal* op als resultaat. Als de eerste kip 2 eieren heeft uitgebroed en de tweede kip 4 dan geeft deze methode als resultaat 6.

In de algemene zin: Als we een variabele aanmaken dan gaat dat als volgt:

type naam = inhoud ;

De inhoud is afhankelijk van het type. Stel dat we een variabele van type int aanmaken dan kunnen we er 5 in doen, maar we kunnen ook een methode aanroepen die een int oplevert. Hetzelfde geldt voor andere types zoals boolean.

We kunnen ook een variabele aanmaken van type Chicken of Egg. In dat geval wijst de variabele naar een bepaalde kip of ei. Wat we daarmee kunnen zien we in de andere paragraaf.

§8. Aanroepen van methodes van een andere klasse

Uitleg over het aanroepen van methodes is te vinden in paragraaf 3.1 op bladzijde 30.

We hebben al gezien hoe we methodes kunnen aanroepen. Er is echter een probleem. Stel dat we in de klasse A zitten, en we willen een methode uitvoeren van klasse B. Hoe doen we dat?

Als eerste moeten we precies weten van welk object van de klasse B we de methode willen uitvoeren. Stel bijvoorbeeld dat klasse B een Auto is, dan moeten we precies weten in wélke auto we de methode moeten aanroepen. We moeten dan een variabele hebben die naar het object wijst.

Bijvoorbeeld:

Auto mijnAuto = getAuto();

We hebben nu een Auto variabele aangemaakt, met naam *mijnAuto*. De getAuto() is een methode die een Auto oplevert. We kunnen nu een methode uitvoeren in dit object door er een punt achter te zetten, dat gaat als volgt:

mijnAuto.rijVooruit();

In algemene zin gaat het dus als volgt:

object.methode();