Inhoud hoofdstuk 1

Tekenen in JavaLogo

Introductie

Leerkern

Zelftoets

Terugkoppeling

Hoofdstuk 1

Tekenen in JavaLogo

INTRODUCTIE

Een computer doet niets uit zichzelf. Voor iedere taak die door een computer wordt uitgevoerd, is een *programma* nodig: een voorschrift dat door de computer kan worden verwerkt, waarin precies en ondubbelzinnig is vastgelegd wat de computer moet doen.

Hoe verloopt die verwerking? Het hart van een computer is de processor, ook wel de centrale verwerkingseenheid genoemd. Deze processor kan eenvoudige instructies verwerken, met betekenissen als 'tel deze twee getallen bij elkaar op', 'zet deze rij enen en nullen in geheugenplaats A' of 'ga verder met de instructie op geheugenplaats X als de inhoud van geheugenplaats A ongelijk is aan nul'. Die instructies zijn gesteld in *machinetaal*. Ieder type processor heeft zijn eigen machinetaal. Een programma is niets anders dan een rij van die instructies.

Toen computers pas bestonden, werden programma's direct in machinetaal geschreven. Inmiddels hoeven we als programmeur niet eens meer te weten hoe de machinetaal er uitziet van de computer die we gebruiken. We formuleren ons programma in een geschikte algemene programmeertaal en laten het aan de computer over daar machinetaal van te maken. Zo'n *hogere programmeertaal* is dus, in tegenstelling tot de machinetaal, *onafhankelijk* van welke processor dan ook.

In het begin leken hogere programmeertalen nog vrij veel op machinetalen. Hieronder ziet u een stukje uit een programma geschreven in de allereerste hogere programmeertaal *Fortran*, waarvan de oudste versie stamt uit 1954.

```
  X = G
  N = 0
1 IF (X) 2, 2, 3
2 STOP
3 X = X - D
  N = N + 1
  GOTO 1
```

Een Fortran-programma bestond net als een machinetaalprogramma uit een rij eenvoudige instructies. Veel structuur had een Fortran-programma verder niet. Naarmate de hardware krachtiger werd en de programma's groter werden, werd de noodzaak van een goede structurering van programma's ook steeds groter. Programmeertalen kwamen daardoor steeds verder van de machinetaal af te staan. Dat maakt het vertaalproces naar machinetaal moeilijker, maar het schrijven van programma's een stuk makkelijker.

Midden jaren ’60 bedacht de wiskundige Seymour Papert de taal Logo. Hij had gewerkt bij Piaget, een bekende psycholoog die veel ontdekt heeft over het verloop van leerprocessen, vooral bij kinderen. Bij het ontwerp van Logo ging het niet om het programmeren op zich, maar om aan kinderen een gereedschap te geven waarmee ze zelf al ontdekkend van alles zouden kunnen leren. Logo kende verschillende vormen waarmee verschillende kennisgebieden verkend konden worden, zoals taal, muziek, wiskunde en robotica. Het populairst werd de omgeving die gebruikmaakte van een turtle: een bestuurbare afbeelding van een schildpad die bij het lopen over het scherm een spoor naliet. De bestuurbare pen waarmee u in deze cursus gaat werken, is een nazaat van deze schildpad.

Logo is een leuke en aansprekende taal om de basisprincipes van programmeren te verkennen, maar wordt inmiddels (vrijwel) niet meer gebruikt. Een taal die op dit moment wel zeer veel gebruikt wordt, is de programmeertaal Java. Java is ontstaan in 1995. De taal dankt zijn populariteit onder meer aan het feit dat het de eerste taal was die het mogelijk maakte om een programma in te bouwen in een webpagina. Tot de komst van Java waren webpagina’s niet veel meer dan folders.

In deze cursus combineren we het tekenen in Logo met de kracht van Java. We gebruiken de taal JavaLogo, die is bedacht door Peter Boon en ontwikkeld door Peter Boon en Paul Bergervoet als een uitbreiding van Java, wat het mogelijk maakt om JavaLogo-programma’s direct op het web te zetten.

In dit eerste hoofdstuk leert u de pen besturen en daarmee ook vlakvullingen te maken. En u leert zelf nieuwe opdrachten te formuleren, zodat u bijvoorbeeld maar één opdracht hoeft te geven om een vierkant van een bepaalde kleur en met een bepaalde afmeting te tekenen.

Software

Het is de bedoeling dat u de programma’s die u in de opgaven schrijft, ook echt door uw computer laat uitvoeren. In de bijlage bij dit hoofdstuk wordt uitgelegd hoe u de benodigde software daarvoor op uw computer kunt installeren en hoe u daarmee om moet gaan.
De installatie wordt beschreven in paragraaf 1 van de bijlage. U kunt deze naar keuze nu uitvoeren of voordat u opgave 1.2 maakt.
De uitwerkingen van de opgaven vindt u aan het eind van het hoofdstuk.

LEERKERN

1 Eerste programma’s

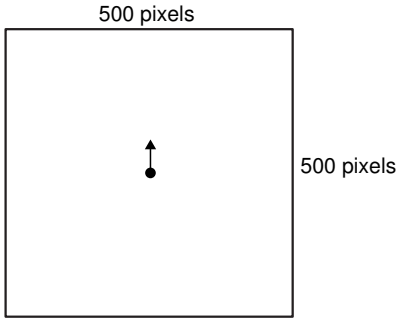
1.1 PEN EN TEKENBLAD

In JavaLogo tekenen we met een pen op een tekenblad van 500 bij 500 pixels.

Bij het besturen van de pen moeten we vier dingen in de gaten houden:
– de positie van de pen op het tekenblad
– de richting waarin de pen aan het bewegen is (in Logo werden lijnen getekend door een schildpad die altijd vooruit liep; de richting bepaalde welke kant de schildpad op een bepaald moment opkeek)

– of de pen aan is (op het tekenblad staat) of uit (van het tekenblad af, wat het mogelijk maakt om de pen te bewegen zonder dat er iets wordt getekend)
– de kleur waarmee de pen tekent.

Als we beginnen te tekenen, staat de pen altijd in het midden van het blad, en wijst naar boven (zie figuur 1.1). De pen is dan bovendien aan.



FIGUUR 1.1 Tekenblad en pen (het bolletje geeft de plaats aan van de pen; het pijltje de huidige richting)

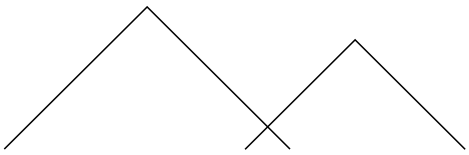
We kunnen de pen besturen met behulp van de volgende opdrachten.
– pen.aan(kleur) zet de pen op het tekenblad en bepaalt de tekenkleur. Voor de kleur, die tussen haakjes staat, moet in deze opdracht een concrete kleur worden ingevuld tussen aanhalingstekens: bijvoorbeeld pen.aan("rood"), of pen.aan("blauw"). We noemen die kleur een *parameter* van de opdracht, in feite een nadere precisering (‘zet de pen aan, en wel met als tekenkleur rood’).

Parameter

– pen.uit() haalt de pen van het tekenblad.
– pen.vooruit(pixels) beweegt de pen het opgegeven aantal pixels vooruit, in de huidige richting. Als de pen aan is, wordt een lijn getekend. Ook deze opdracht heeft een parameter: het aantal pixels. We moeten dus bijvoorbeeld schrijven pen.vooruit(30), of pen.vooruit(122).
– pen.links(hoek) draait de pen naar links (tegen de wijzers van de klok in), over de aangegeven hoek (in graden).
– pen.rechts(hoek) draait de pen naar rechts (met de klok mee), over de aangegeven hoek. Bij de opdrachten links en rechts moet de hoek als parameter worden opgegeven.

Voorbeeld: twee bergen

Als eerste voorbeeld gaan we de pen een klein berglandschap laten tekenen, als getoond in figuur 1.2.



FIGUUR 1.2 Twee bergen

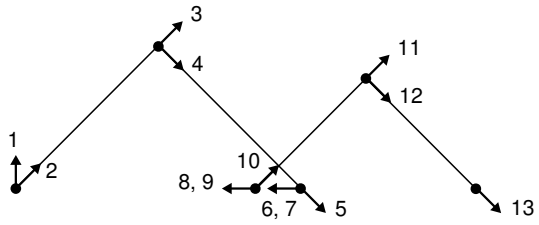
De reeks opdrachten die samen deze tekening maken, ziet er als volgt uit (de regelnummers horen niet bij de opdrachten):

```
1    pen.aan("zwart");
2    pen.rechts(45);
3    pen.vooruit(100);
4    pen.rechts(90);
5    pen.vooruit(100);
6    pen.rechts(135);
7    pen.uit();
8    pen.vooruit(30);
9    pen.aan("zwart");
10   pen.rechts(135);
11   pen.vooruit(80);
12   pen.rechts(90);
13   pen.vooruit(80);
```

Merk op dat na elke opdracht een puntkomma moet staan en dat de kleur zwart tussen aanhalingstekens staat. Uw programma werkt niet als u intypt: pen.aan(zwart)!

Merk ook op dat alle opdrachten een parameter hebben, behalve pen.uit(); dat behoeft geen nadere precisering. De haakjes moeten er echter wel staan: typt u daar pen.uit, dan gaat het fout.

Figuur 1.3 laat precies zien waar de pen na elke opdracht staat en in welke richting deze wijst. Opdracht 7 haalt de pen van het tekenblad en beweegt deze dus niet; na de verplaatsing in opdracht 8 wordt de pen in opdracht 9 weer op het tekenblad geplaatst.
Ga zelf na dat de hoeken waarover gedraaid wordt, kloppen. De eerste draai gaat over 45° (de helft van een rechte hoek), voor de bergtoppen wordt 90° gedraaid en in opdrachten 6 en 10 wordt gedraaid over de som van die twee (een rechte hoek en dan nog 45° verder).



FIGUUR 1.3 Richting en positie van de pen na elke opdracht

OPGAVE 1.1
Geef een reeks opdrachten voor het tekenen van een vierkant met een zijde van 100 pixels.

1.2 EEN VOLLEDIG JAVALOGO-PROGRAMMA

Een volledig JavaLogo-programma moet meer bevatten dan alleen de opdrachten aan de pen.

Figuur 1.4 toont een volledig JavaLogo-programma voor het berglandschap uit figuur 1.2.

```
1   import logotekenap.*;

    public class Bergen extends TekenApplet
    {
5     public void initialiseer()
      {
      }

      public void tekenprogramma()
10    {
        pen.aan("zwart");
        pen.rechts(45);
        pen.vooruit(100);
        pen.rechts(90);
15      pen.vooruit(100);
        pen.rechts(135);
        pen.uit();
        pen.vooruit(30);
        pen.aan("zwart");
20      pen.rechts(135);
        pen.vooruit(80);
        pen.rechts(90);
        pen.vooruit(80);
      }
25  }
```

FIGUUR 1.4 Een volledig JavaLogo-programma

Programmakop

We lichten deze code globaal toe.
Regels 1 t/m 3 vormen de kop van het programma. Die kop is altijd hetzelfde op de naam van het programma na: die varieert (we noemden dit programma Bergen). We verdiepen ons niet in de vraag waarom die kop er zo uit moet zien. We zullen zien dat u die eerste regels zelfs niet zelf in hoeft te typen; JCreator zet die regels al voor u klaar.
Meteen na de kop van het programma volgt in regel 4 een accolade; het programma eindigt met de bijbehorende sluitaccolade in regel 25.
Tussen die accolades staan twee vaste programmagedeeltes met de namen initialiseer en tekenprogramma (later komt daar nog het een en ander bij).
Beide programmadelen bevatten een reeks opdrachten, die u zelf moet invullen en die dus bij elk programma anders zijn. In Java heet een reeks opdrachten met een bepaalde naam een *methode*. Elk JavaLogo-programma bevat dus minimaal twee methoden, namelijk een met de naam initialiseer en een met de naam tekenprogramma. We kijken eerst naar de methode tekenprogramma.

Methode

Methode tekenprogramma

De methode tekenprogramma bevat de penbewegingen. Deze methode begint op regel 9. Ook een methode begint met een kopregel. Deze begint altijd met de woorden 'public void' (dat is een eis van Java), gevolgd door de naam van de methode, gevolgd door een paar haakjes (we zullen in paragraaf 2 zien waar die goed voor zijn).
Na de kopregel volgt een openingsaccolade (regel 10), de serie opdrachten die u al kent uit de vorige paragraaf (regels 11-23), en een sluitaccolade waarmee de methode wordt afgesloten (regel 24).

Methode initialiseer

De methode initialiseer bevat opdrachten die uitgevoerd moeten worden vóór met tekenen wordt begonnen. Ook deze methode bestaat uit een

kopregel (regel 5), een openingsaccolade (regel 6) en een sluitaccolade (regel 7). Op dit moment staan tussen die accolades geen opdrachten: deze methode hoeft nog niets te doen!

Bij het uitvoeren van een JavaLogo-programma worden de opdrachten in deze twee vaste methoden uitgevoerd: eerst die uit initialiseer en dan die uit tekenprogramma.

Tot slot merken we nog op, dat sommige woorden vet staan gedrukt; deze hebben een speciale betekenis in Java.

OPGAVE 1.2

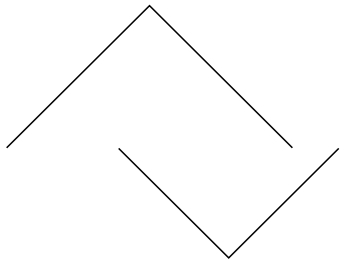
Installeer, als u dat nog niet gedaan hebt, nu eerst de software (zie paragraaf 1 van de bijlage).
a Maak gebruik van de aanwijzingen in de bijlage om het programma uit figuur 1.4 daadwerkelijk op uw computer in te typen en uit te voeren. Neem de opdrachten precies zo over (denk aan de puntkomma's en gebruik in dit geval nergens hoofdletters)!
b Voeg aan de methode initialiseer één opdracht toe, als volgt. Neem ook nu de opdracht precies zo over; denk aan de puntkomma en let op de hoofdletters T en M.

```
public void initialiseer()
{
       maakTraceMogelijk();
}
```

Voer het programma opnieuw uit. Wat is het effect? Wat kunt u met de verschillende knoppen die u er nu bij krijgt?

Tracen

Tracen betekent de uitvoering van een programma stap voor stap volgen. De tracemogelijkheid in JavaLogo is onmisbaar bij het aan de praat krijgen van programma's. Bij het besturen van de pen is een fout namelijk snel gemaakt, waardoor uw tekening er geheel anders uit komt te zien dan de bedoeling was. Draait u bijvoorbeeld in regel 16 van het programma 45° naar links in plaats van 135° naar rechts, dan ziet uw tekening er opeens uit als getoond in figuur 1.5. Met behulp van de tracemogelijkheid kunt u dan achterhalen welke opdracht fout is.



FIGUUR 1.5 Eén foute draai en de berg staat op zijn kop.

1.3 PEN EN TEKENBLAD NADER BEKEKEN

Pen

Figuur 1.6 toont alle opdrachten die aan de pen gegeven kunnen worden. We zullen nu precies uitleggen wat die opdrachten doen (in paragraaf 1.1

hebben we een aantal bijzonderheden nog niet vermeld). De schuingedrukte woorden staan voor de parameters; daar moet bij het geven van de opdracht steeds een waarde worden ingevuld.

```
pen.aan(kleur);
pen.uit();
pen.vooruit(aantalPixels);
pen.links(hoek);
pen.vulAan(kleur);
pen.vulUit();
pen.stap(x, y);
```

FIGUUR 1.6 Alle opdrachten aan de pen

aan

pen.aan(*kleur*)
Plaatst de pen op het tekenblad, met als tekenkleur de opgegeven kleur. Tabel 1.1 toont mogelijke waarden van deze kleur. Vergeet bij het gebruik van deze kleuren niet de aanhalingstekens!
Aan het begin van het programma staat de pen al op het tekenblad, met als tekenkleur zwart.

TABEL 1.1 Kleuraanduidingen in JavaLogo

"rood"	"groen"	"lichtgrijs"
"roze"	"magenta"	"grijs"
"oranje"	"blauw"	"wit"
"geel"	"cyaan"	"zwart"

Het is ook mogelijk om andere kleuren te gebruiken, door de RGB-waarden van een kleur op te geven. Dat zijn drie getallen tussen 0 en 255, die de aandelen van achtereenvolgens de kleuren rood, groen en blauw weergeven. Zo kunt u in plaats van pen.aan("rood") ook schrijven pen.aan(255, 0, 0), of in plaats van pen.aan("cyaan") pen.aan(0, 255, 255). U hebt daarmee meer dan 16 miljoen kleuren tot uw beschikking. Bekijk deze kleuren desgewenst door bijvoorbeeld in het Windows-programma Paint uit het menu Kleuren te kiezen voor Kleuren bewerken en dan te klikken op Aangepaste kleuren. U krijgt dan het volledige kleurenspectrum te zien, met de RGB-waarden rechts onderin.

uit

pen.uit();
Haalt de pen van het tekenblad.

vooruit

vooruit(*aantalPixels*);
Beweegt de pen het opgegeven aantal pixels. Als een positief getal wordt opgegeven, bijvoorbeeld pen.vooruit(30), dan is de beweging in de huidige penrichting; wordt een negatief getal opgegeven, bijvoorbeeld pen.vooruit(–30), dan beweegt de pen achteruit, dus in de tegenovergestelde richting.
Het aantal hoeft niet per se een geheel getal te zijn; het is toegestaan om bijvoorbeeld pen.vooruit(2.5) te schrijven. In het volgende hoofdstuk zullen we dat ook wel doen.

links

pen.links(*hoek*);
Draait de pen de opgegeven hoek (in graden) naar links. De hoek mag ook een gebroken getal zijn, bijvoorbeeld pen.links(22.5). Negatieve getallen zijn wel toegestaan, maar het gebruik ervan is niet nodig: pen.links(–22.5) is hetzelfde als pen.rechts(22.5).

rechts

pen.rechts(*hoek*);
Draait de pen de opgegeven hoek naar rechts (zie verder bij links).

vulAan

pen.vulAan(*kleur*);
Zie bij pen.vulUit().

vulUit

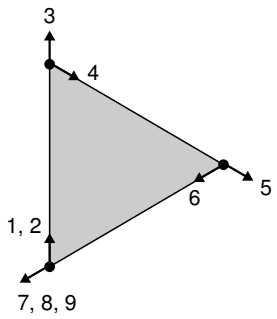
pen.vulUit();
De methoden vulAan(kleur) en vulUit() worden gebruikt om een figuur in te kleuren. Eerst wordt de opdracht pen.vulAan(*kleur*) gegeven, dan tekent de pen een gesloten figuur (een vierkant, een driehoek, een ster of wat dan ook, als de pen maar weer terugkomt op het uitgangspunt) en tot slot wordt de opdracht pen.vulUit() gegeven en wordt de figuur ingekleurd.

Voorbeeld

Met behulp van de volgende reeks opdrachten wordt een zwartomlijnde rode driehoek getekend (zie figuur 1.7). De driehoek is gelijkzijdig, en dus zijn alle hoeken 60°.

```
1    pen.aan("zwart");
     pen.vulAan("rood");
     pen.vooruit(100);
     pen.rechts(120);
5    pen.vooruit(100);
     pen.rechts(120);
     pen.vooruit(100);
     pen.vulUit();
9    pen.uit();
```

De eerste opdracht is eigenlijk overbodig; de pen staat aan het begin van het programma altijd al aan met zwart als tekenkleur. Figuur 1.7 toont ook de penpositie en penrichting na elke opdracht, zodat u duidelijk kunt zien waarom er op elke hoek 120° gedraaid moet worden en niet, zoals u misschien verwacht had, 60°.



FIGUUR 1.7 Een gekleurde driehoek met een zwarte rand

Het kan ook zonder zwarte rand, door de pen expliciet uit te zetten, maar het vullen aan:

```
pen.uit();
pen.vulAan("rood");
pen.vooruit(100);
pen.rechts(120);
pen.vooruit(100);
pen.rechts(120);
pen.vooruit(100);
```

```
pen.vulUit();
```

Figuur 1.8 toont het resultaat.

FIGUUR 1.8 Een gekleurde driehoek zonder rand

stap

pen.stap(*x*, *y*);
Met behulp van deze methode kunt u de pen in één keer bewegen naar een punt dat *y* pixels verderop ligt in de huidige richting van de pen, en *x* pixels in de richting daar loodrecht op (met de klok mee), zonder dat de pen na afloop van richting is veranderd. Wijst de pen bijvoorbeeld omhoog, dan tekent pen.stap(100, 0) een lijn naar rechts, terwijl de pen nog steeds omhoog wijst. De opdracht pen.stap(80, 100) tekent een schuine lijn naar een punt dat 80 pixels naar rechts en 100 pixels naar boven ligt.

Pen mag buiten het tekenblad komen.

Wat gebeurt er als u een opdracht aan de pen geeft die deze buiten het tekenblad brengt, dus bijvoorbeeld pen.vooruit(1000)? Het antwoord is: niets. U mag rustig buiten het tekenblad tekenen, alleen is een deel van de tekening dan onzichtbaar. Dat lijkt zinloos, maar soms is het heel handig. In het volgende hoofdstuk zullen we er ook wel eens gebruik van maken.

Tekenblad

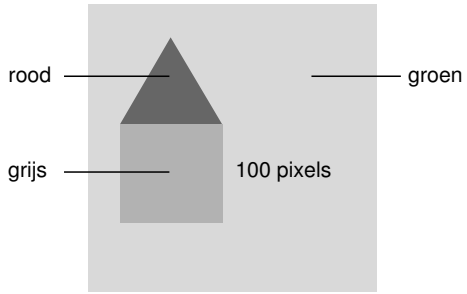
Tot nu toe hebben we alleen gekeken naar opdrachten aan de pen. We kunnen echter ook een opdracht geven aan het tekenblad, en wel om de kleur daarvan te wijzigen (de standaardkleur is wit).

achtergrondkleur

tekenblad.achtergrondkleur(*kleur*)
Geeft het tekenblad de opgegeven kleur. Deze opdracht moet vóór de opdrachten aan de pen worden geplaatst, anders wordt hetgeen al getekend is, weer uitgewist. Het volgende tekenprogramma tekent een zwarte lijn op een geel blad:

```
tekenblad.achtergrondkleur("geel");
pen.vooruit(100);
```

OPGAVE 1.3
Schrijf een JavaLogo-programma dat op een groen tekenblad een grijs huisje tekent met een rood dak, als getoond in figuur 1.9 (zonder kleuren). De zijden van vierkant en driehoek zijn 100 pixels. Voer het programma uit op uw computer.
Aanwijzingen
– Sluit eerst het vorige project en maak dan een nieuw.
– Let goed op details zoals puntkomma's, hoofdletters (bijvoorbeeld in vulAan en vulUit) en aanhalingstekens rond de kleuren.



FIGUUR 1.9 Grijs huisje met rood dak op groen veld. Het huisje is 100 X 100 pixels; het dak is gelijkzijdig.

Commentaar

In de uitwerking van opgave 1.3 hebben we in het programma regels geplaatst die bedoeld zijn als toelichting voor een menselijke lezer. Zulke regels worden *commentaar* genoemd. Het JavaLogo-systeem slaat die regels gewoon over. Het moet dan natuurlijk wel weten wat tot het commentaar en wat tot het programma zelf behoort. Het commentaar moet dus herkenbaar zijn aan zijn vorm. In JavaLogo (en in Java) wordt dat op twee manieren aangegeven:
– na `//` is alles tot aan het einde van de regel commentaar
– na `/*` is alles commentaar tot er ergens `*/` staat (dat kan vele regels verderop zijn).
De eerste vorm wordt dus gebruikt voor relatief korte commentaren; de tweede vorm voor langere.

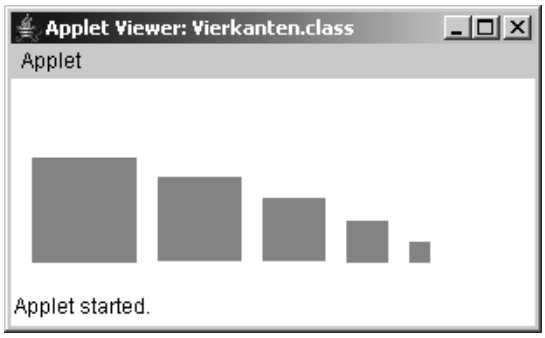
Voorbeeld

In het volgende programmafragment worden beide vormen gebruikt. De sterretjes aan het begin van regels 3-5 zijn niet verplicht, maar wel gebruikelijk.

```
1    pen.rechts(90);  // kort commentaar tot einde regel
     /*
      * Commentaar dat zich uitstrekt over verschillende
      * regels. Meestal wordt aan het begin van elke regel
5     * een sterretje geplaatst.
      */
     pen.vooruit(100);
     ...
```

2 Extra methoden

Figuur 1.10 toont een tekening van vijf rode vierkanten, die steeds iets kleiner worden. Het JavaLogo-programma dat deze vierkanten getekend heeft, heet Vierkanten.



FIGUUR 1.10 Vijf verschillende vierkanten

Met behulp van de opdrachten die u in de vorige paragraaf geleerd hebt, kunt u deze tekening natuurlijk prima maken, in de vorm van een lange lijst opdrachten aan de pen. We tonen het begin van die lijst (we hebben eerst de pen uitgezet en een stuk naar links verplaatst, maar dat laten we even weg):

```
// teken een rood vierkant met zijde 100
pen.vulAan("rood");
pen.vooruit(100);
pen.rechts(90);
pen.vooruit(100);
pen.rechts(90);
pen.vooruit(100);
pen.rechts(90);
pen.vooruit(100);
pen.rechts(90);
vulUit();

// ga naar het volgende vierkant
pen.rechts(90);
pen.vooruit(120);
pen.links(90);

// teken een rood vierkant met zijde 80
pen.vulAan("rood")
pen.vooruit(80);
pen.rechts(90);
...
```

En zo verder. Afwisselend tekenen we een vierkant en bewegen we de pen naar het volgende vierkant. Een saai programma. Jammer eigenlijk, dat we de pen geen opdrachten vierkant en naarRechts kunnen sturen; dat zou het programma een stuk korter maken.

OPGAVE 1.4
Stel dat aan de pen ook de volgende opdrachten gegeven konden worden.

pen.vierkant(*zijde*)
Tekent een gevuld, rood vierkant met de gegeven zijde. Na het tekenen staat de pen weer precies zoals in het begin.

pen.naarRechts(*stap*)

Beweegt de pen *stap* pixels naar rechts, zonder van richting te veranderen.

Welke opdrachten zouden er dan nodig zijn om de vijf vierkanten te tekenen? De zijden van de vierkanten zijn 100, 80, 60, 40 en 20 pixels, en de afstand tussen de vierkanten is 20 pixels.

We kunnen deze opdrachten niet aan de pen geven. Maar iets anders kan wel: we kunnen in het JavaLogo-programma extra methoden vierkant en naarRechts opnemen en die kunnen we vervolgens in opdrachten gebruiken.
Om te laten zien hoe dat werkt, moeten we twee vragen beantwoorden:
– Hoe kunnen we extra methoden toevoegen?
– Hoe formuleren we de bijbehorende opdrachten?

We bekijken die vragen een voor een.

2.1 ZELF METHODEN TOEVOEGEN

De JavaLogo-programma's die we tot nu toe gezien hebben, bevatten steeds twee vaste methoden: een methode initialiseer, waar opdrachten in staan die moeten worden uitgevoerd voor er getekend wordt, en een methode tekenprogramma. Die laatste methode is de belangrijkste, want daarin worden de opdrachten gegeven die de tekening maken.

Aan een JavaLogo-programma kunnen we zelf methoden toevoegen. Anders dan initialiseer en tekenprogramma worden die niet vanzelf na het starten van het programma uitgevoerd; daar moeten we zelf voor zorgen en daar hebben we dus ook zelf controle over. Hoe we dat doen, bekijken we echter pas in paragraaf 2.2; hier bekijken we alleen hoe we zo'n extra methode op moeten schrijven.

Als eerste voorbeeld tonen we een programma met een extra methode die een rood vierkant met zijde 100 tekent.

```
1   import logotekenap.*;

    public class Vierkanten extends TekenApplet
    {
5
      public void initialiseer()
      {
      }

10    public void tekenprogramma()
      {
        ...
      }

15    public void vierkant()
      {
        pen.vulAan("rood");
        pen.vooruit(100);
        pen.rechts(90);
20      pen.vooruit(100);
        pen.rechts(90);
        pen.vooruit(100);
        pen.rechts(90);
        pen.vooruit(100);
```

```
25        pen.rechts(90);
          pen.vulUit();
        }
      }
```

Met deze toevoeging staat in het programma nu niet alleen hoe een tekening geïnitialiseerd en getekend moet worden, maar ook hoe een vierkant getekend moet worden. Dat verklaart de naam methode: een methode bevat een reeks opdrachten die aangeven hoe iets bepaalds gedaan moet worden.
De methode vierkant heeft precies dezelfde opbouw als de methoden initialiseer en tekenprogramma:
– de methode begint met een kopregel, waarin nu als naam vierkant staat
– daarna volgt een reeks instructies tussen accolades.

Methode met parameters

Door deze methode toe te voegen, kunnen we straks in tekenprogramma een opdracht geven om een vierkant te tekenen, zoals we graag wilden (zie opgave 1.4). We zijn echter nog niet tevreden, want we wilden aan die opdracht ook een zijde mee kunnen geven als parameter. Dat kan, maar dan moeten we zowel de kopregel als de opdrachten in methode vierkant veranderen. De kopregel gaat er als volgt uitzien:

```
public void vierkant(double zijde)
```

Type

Tussen de haakjes achter de naam vierkant staat nu een parameter. Van die parameter wordt eerst het *type* vermeld (double) en dan de naam (zijde). Het type geeft aan wat voor soort parameter het is, dus wat voor soort waarden we mee mogen geven in een opdracht vierkant. Voorlopig zullen we maar twee typen gebruiken.

double

– Het type *double* gebruiken we voor getallen, zoals 100, 75, 0, –25, 22,5, 48,234 (in de programmacode gebruiken we een decimale punt). De aanduiding double voor getallen is afkomstig uit Java en ligt niet meteen voor de hand; deze verwijst naar de manier waarop zulke getallen in het computergeheugen worden voorgesteld.

String

– Het type *String* (met een hoofdletter!) gebruiken we voor stukjes tekst tussen dubbele aanhalingstekens; in onze programma's zullen dat bijna altijd kleuren zijn.

Zouden we de methode vierkant een kleur als parameter willen geven in plaats van een zijde, dan zou de kop er als volgt uitzien:

```
public void vierkant(String kleur)
```

Methoden kunnen ook meer dan één parameter hebben. Hier is de kop van een methode huis, die een huis tekent als in figuur 1.9, maar dan met de hoogte van het huis (100 in figuur 1.9), de kleur van het huis zelf en de kleur van het dak als parameters:

```
public void huis(double hoogte, String kleur, String kleurDak)
```

Bij *elke* parameter staat een type en tussen de parameters staan komma's.

We gaan weer terug naar de methode vierkant met alleen de zijde als parameter. We moeten de opdrachten in die methode nu zo formuleren, dat er een vierkant met de opgegeven zijde wordt getekend. Met andere

woorden: de pen moet niet 100 pixels vooruit gaan, maar een aantal pixels gelijk aan zijde. We kunnen dat eenvoudig opschrijven, als volgt:

```
public void vierkant(double zijde)
{
  pen.vulAan("rood");
  pen.vooruit(zijde);
  pen.rechts(90);
  pen.vooruit(zijde);
  pen.rechts(90);
  pen.vooruit(zijde);
  pen.rechts(90);
  pen.vooruit(zijde);
  pen.rechts(90);
  pen.vulUit();
}
```

Overal waar eerst 100 stond, schrijven we nu zijde.

OPGAVE 1.5
Schrijf een methode driehoek met als parameters de zijde en de kleur. De driehoek moet gelijkzijdig zijn, een gegeven zijde hebben en opgevuld worden met de meegegeven kleur (vergelijk figuur 1.9).
NB: u hoeft er nu nog niet voor te zorgen dat die methode ook wordt uitgevoerd; dat komt hierna.

2.2 ZELFGEDEFINIEERDE METHODEN UITVOEREN

In de methode tekenprogramma gaan we nu niet alleen opdrachten geven aan de pen, maar ook opdrachten die ervoor zorgen dat onze zelfgedefinieerde methoden worden uitgevoerd.
In een dergelijke opdracht ontbreekt de aanduiding 'pen.' (of 'tekenblad.'). Het JavaLogo-systeem weet daardoor dat we een toegevoegde methode willen uitvoeren.

Voorbeeld

Als we willen dat de methode vierkant wordt uitgevoerd, schrijven we:

```
vierkant(100);
```

Als er meer parameters zijn, dan schrijven we die achter elkaar, met een komma ertussen, dus bijvoorbeeld

```
huis(100, "grijs", "rood");
```

Aantal, type en volgorde van parameters moet juist zijn

De parameters in de opdrachten moeten kloppen met die in de kop van de methode. Bij de kop

```
public void vierkant(double zijde)
```

passen opdrachten als vierkant(100) en vierkant(25), maar niet vierkant("rood") (verkeerde soort parameter), of vierkant(100, "rood") (teveel parameters).
Ook de volgorde moet kloppen. Bij de kop

```
public void huis(double hoogte,
String kleur, String kleurDak)
```

past bijvoorbeeld geen opdracht huis("grijs", 100, "rood"), want "grijs" is geen getal en 100 is geen String.

Figuur 1.11 toont een volledig programma voor de vijf rode vierkanten uit figuur 1.10, met extra methoden vierkant en naarRechts. Vergelijk de opdrachten in tekenprogramma met die uit de terugkoppeling bij opgave 1.4!

```
import logotekenap.*;

public class Vierkanten extends TekenApplet
{

  public void initialiseer()
  {
  }

  // teken vijf rode vierkanten van afnemende grootte
  public void tekenprogramma()
  {
    pen.uit();
    pen.links(90);
    pen.vooruit(230);
    pen.rechts(90);
    vierkant(100);
    naarRechts(120);
    vierkant(80);
    naarRechts(100);
    vierkant(60);
    naarRechts(80);
    vierkant(40);
    naarRechts(60);
    vierkant(20);
  }

  // teken gevuld, rood vierkant met gegeven zijde
  public void vierkant(double zijde)
  {
    pen.vulAan("rood");
    pen.vooruit(zijde);
    pen.rechts(90);
    pen.vooruit(zijde);
    pen.rechts(90);
    pen.vooruit(zijde);
    pen.rechts(90);
    pen.vooruit(zijde);
    pen.rechts(90);
    pen.vulUit();
  }

  // ga stap pixels naar rechts zonder pen te draaien
  public void naarRechts(double stap)
  {
    pen.rechts(90);
    pen.vooruit(stap);
    pen.links(90);
  }
}
```

FIGUUR 1.11 Een JavaLogo-programma met twee extra methoden

Merk op dat de opdrachten in dit programma nu niet langer van boven naar beneden worden uitgevoerd. De eerste opdracht die wordt

uitgevoerd, is pen.uit() op regel 13, gevolgd door de drie opdrachten aan de pen op regels 14, 15 en 16. Dan moet er een vierkant met zijde 100 getekend worden, en dat betekent dat de eerstvolgende opdracht die op regel 30 is: de pen wordt aangezet met vulkleur rood. Dan volgen de andere opdrachten in methode vierkant, met als laatste pen.vulUit().
De opdracht vierkant(100) op regel 17 is daarmee klaar, zodat het programma dan verder gaat met de opdracht op regel 18: naarRechts(120). Ook dat is een eigen methode, dus worden nu de opdrachten op regels 46, 47 en 48 uitgevoerd. En dan gaan we verder met regel 19: weer een opdracht om een vierkant te tekenen, dit keer met zijde 80. En dus worden opnieuw de opdrachten op regels 31 t/m 40 uitgevoerd.
U kunt dit zelf heel goed volgen door dit programma in te voeren in JCreator, met de tracemogelijkheid aan. We raden u sterk aan dat ook echt te doen.

Staat het vierkant dat getekend wordt door de opdracht vierkant(100), altijd rechtop?

In dit geval wel, maar in het algemeen hangt dat af van de richting van de pen op het moment dat de methode wordt uitgevoerd. Beginnen we in het tekenprogramma met een draai van 45° voor we de opdracht vierkant() geven, dan staan alle vierkanten scheef. We kunnen dezelfde methode vierkant gebruiken om vierkanten in verschillende standen te tekenen.

OPGAVE 1.6

Stel we vervangen in figuur 1.11 de methode tekenprogramma door de volgende:

```
public void tekenprogramma()
{
  pen.uit();
  vierkant(100);
  pen.rechts(120);
  vierkant(100);
  pen.rechts(120);
  vierkant(100);
  pen.rechts(120);
}
```

Hoe ziet de tekening er uit die nu wordt gemaakt?

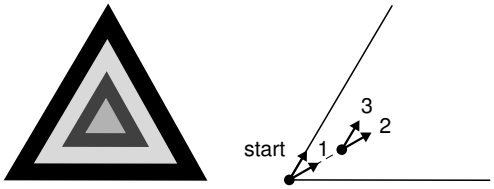
Belangrijke aanwijzing

Wij hebben de methoden vierkant en driehoek (terugkoppeling van opgave 1.5) zo geformuleerd, dat de pen aan het einde van de methode op dezelfde plek staat en in dezelfde richting wijst als aan het begin. We raden u sterk aan om, als u een JavaLogo-programma schrijft, methoden op die manier te formuleren (als dat kan), ook als daar extra stappen en draaien voor nodig zijn. In de methoden vierkant en driehoek bijvoorbeeld, zijn de laatste draaien feitelijk overbodig. Controleer dan ook altijd of u dat goed gedaan hebt, door eerst een tekenprogramma te schrijven waarin alleen die ene methode wordt uitgevoerd. Als u een ingewikkelde tekening wilt maken, raakt u hierdoor minder snel de controle over de pen kwijt.

De penrichting kunt u ook controleren zonder de methode uit te voeren, door na te gaan of de pen een netto draai maakt van 360° of een veelvoud daarvan.

OPGAVE 1.7

Schrijf een JavaLogo-programma dat de tekening uit figuur 1.12 (links) maakt en voer dit programma uit op uw computer.
Aanwijzingen
– Gebruik de methode driehoek uit opgave 1.5.
– Begin met de grootste driehoek en teken de andere daar dan overheen.
– Schrijf een tweede extra methode naarBinnen() die de pen naar het startpunt van de volgende driehoek brengt. In figuur 1.12 ziet u de nodige penbewegingen. Uiteraard verplaatst deze methode de pen wel!
– De kleuren van de driehoeken zijn zwart, geel, blauw en rood (van buiten naar binnen).
– De zijden van de driehoeken zijn 160, 120, 80 en 40 pixels. De stap naar binnen is 24 pixels.



FIGUUR 1.12 Geneste driehoeken (links) en penbewegingen naar binnen (rechts, niet in verhouding)

Het programma uit de terugkoppeling is eigenlijk nog niet helemaal bevredigend. Zowel in de methode tekenprogramma als in de methode driehoek staan herhaaldelijk vrijwel dezelfde opdrachten; als we 20 achthoeken in elkaar zouden willen tekenen, zou dat weer erg saai worden. Hogere programmeertalen als Java en JavaLogo bieden echter allerlei mogelijkheden om structuur aan te brengen in een reeks opdrachten, dus om bijvoorbeeld aan te geven dat een opdracht een aantal keren herhaald moet worden. In hoofdstuk 2 gaan we deze mogelijkheden verder onderzoeken.

Veelgemaakte vormfouten

Tot slot geven we een paar aanwijzingen om veelgemaakte vormfouten bij het definiëren en uitvoeren van methoden te voorkomen.
– Na iedere opdracht staat een puntkomma, maar na de kop van een methode staat *geen* puntkomma!
– Kleuren zelf staan tussen aanhalingstekens, maar de parameter kleur *niet*. Het is dus pen.vulAan(kleur) en niet pen.vulAan("kleur").
– Typeaanduidingen staan alleen in de kopregels van extra methoden, maar nooit in opdrachten: we schrijven pen.vooruit(zijde) en niet pen.vooruit(double zijde) en ook geen opdracht vierkant(double zijde).
– Denk aan hoofdletters en kleine letters. De typen zijn double (met kleine letter) en String (met hoofdletter). Als de methode vierkant heet, dan moet in de opdracht ook vierkant staan en niet Vierkant.
– Fouten in het aantal en het type van de parameters leiden tot een mogelijk voor u wat onduidelijke foutmelding. Schrijft u bijvoorbeeld vierkant("rood") in plaats van vierkant(100), dan krijgt u de melding

vierkant(double) in Vierkanten cannot be applied to Java.lang.String

Onthoud, dat '... cannot be applied to ...' duidt op een parameterfout.

ZELFTOETS

Met behulp van de vragen in deze zelftoets kunt u controleren of u de theorie van dit hoofdstuk begrepen hebt. Maak deze zelftoets zonder de computer te gebruiken. De antwoorden staan na de uitwerkingen van de opgaven.

1 Gegeven de volgende eigen methode:

```
public void halveZeshoek()
{
  pen.rechts(30);
  pen.vooruit(100);
  pen.rechts(60);
  pen.vooruit(100);
  pen.rechts(60);
  pen.vooruit(100);
  pen.rechts(120);
  pen.vooruit(200);
}
```

Welke opdracht zou u toe moeten voegen opdat de pen aan het eind van deze methode in dezelfde richting wijst als in het begin?

2 Gegeven is de methode driehoek uit de terugkoppeling van opgave 1.5, die een gevulde, gelijkzijdige driehoek met een gegeven zijde en van een gegeven kleur tekent. Welke van de volgende vier tekenprogramma's tekenen de zwarte zandloper uit figuur 1.13, waarbij beide driehoeken zijde 100 hebben? Er is precies één antwoord goed.
NB: de methode driehoek tekent deze rechtsom, dus met de wijzers van de klok mee, en laat de pen achter op de plaats en in de richting waar deze begon.

FIGUUR 1.13 Een zwarte zandloper

a

```
public void tekenprogramma()
{
  pen.rechts(30);
  driehoek(100, "zwart");
     pen.rechts(180);
     driehoek(100, "zwart");
}
```

b

```
public void tekenprogramma()
{
  pen.links(30);
  driehoek(100, "zwart");
  pen.rechts(180);
  driehoek(100, "zwart");
}
```

c

```
public void tekenprogramma()
{
  pen.rechts(30);
  driehoek(100, "zwart");
  pen.rechts(60);
  driehoek(100, "zwart");
}
```

d

```
public void tekenprogramma()
{
  pen.links(30);
  driehoek(100, "zwart");
  pen.rechts(60);
  driehoek(100, "zwart");
}
```

3 Gegeven een methode met de volgende kop:

```
public void tekenIets(double p1, String p2, double p3)
```

Geef voor elk van de volgende tekenopdrachten aan of deze in overeenstemming is met deze methodekop en zo niet, wat er mis mee is.

a `tekenIets(0.5, 7, "zwart");`
b `tekenIets(7, "zwart");`
c `tekenIets(7, "zwart", 0.5);`
d `tekenIets(double 0.5, 0.7, String "zwart");`
e `tekenIets(double 0.5, String "zwart", double 7);`

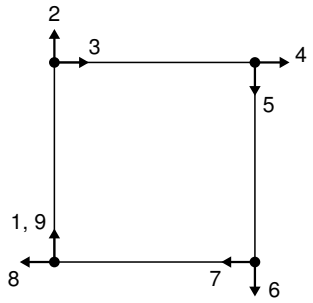
TERUGKOPPELING

1 Uitwerking van de opgaven

1.1 Als we rechtsom tekenen, dan ziet de reeks er als volgt uit:

```
1   pen.aan("zwart");
2   pen.vooruit(100);
3   pen.rechts(90);
4   pen.vooruit(100);
5   pen.rechts(90);
6   pen.vooruit(100);
7   pen.rechts(90);
8   pen.vooruit(100);
9   pen.rechts(90);
```

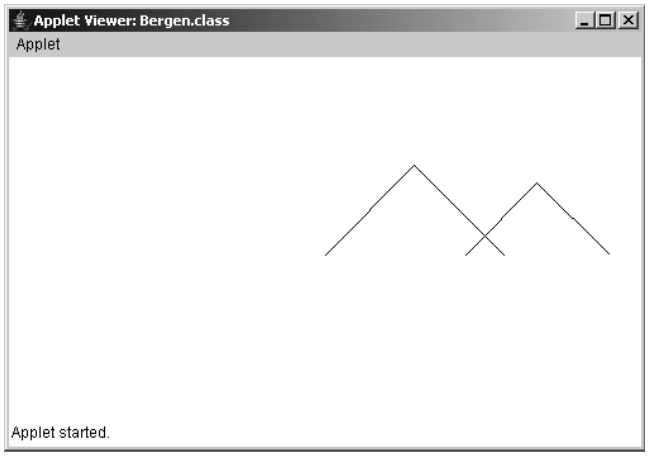
Figuur 1.14 laat weer richting en positie van de pen zien aan het eind van elke opdracht. U ziet dat de pen in dit geval eindigt op dezelfde plaats en in dezelfde richting als aan het begin. Strikt nodig is dit niet; dus eventueel zou opdracht 9 ook weg kunnen.



FIGUUR 1.14 Richting en positie van de pen na elke opdracht

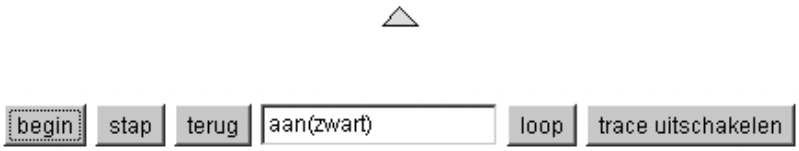
Opmerking
Strikt gesproken is deze uitwerking niet helemaal correct. De getekende lijn heeft zelf een dikte van 1 pixel; de buitenafmetingen van dit vierkant zijn daardoor 101 bij 101 pixels. We houden hier verder geen rekening mee.

1.2 a Geen uitwerking (zie bijlage voor aanwijzingen). Als u het goed gedaan hebt, ziet het resultaat er uit als getoond in figuur 1.15.



FIGUUR 1.15 Tekenblad met bergen

b Met behulp van de opdracht maakTraceMogelijk() kunt u het programma stap voor stap doorlopen. Na het toevoegen ervan verschijnt onder in beeld een knop met het opschrift 'trace aanschakelen'. Klikt u op die knop, dan gaat de pen naar de beginpositie en ziet u de knoppen uit figuur 1.16. Tevens wordt de positie en richting van de pen aangegeven door een (geel) driehoekje.



FIGUUR 1.16 Dit ziet u na klikken op de knop 'trace aanschakelen'; het (gele) driehoekje toont positie en richting van de pen.

U kunt nu de bewegingen van de pen stap voor stap volgen, door herhaaldelijk op de knop stap te klikken. Het venster toont steeds de laatste aan de pen gegeven opdracht (zonder pen.). Klikt u op terug, dan wordt de laatste penbeweging ongedaan gemaakt. Klikken op loop leidt tot het (vertraagd) uitvoeren van de rest van het programma. De knop loop krijgt dan opschrift stop; klikt u daarop, dan kunt u verder weer stap voor stap door het programma heen lopen. Met de knop begin kunt u terug naar het begin van het programma. Met de knop trace uitschakelen ten slotte gaat u terug naar de normale uitvoering.

1.3 Dit programma ziet er bijvoorbeeld als volgt uit. (Variaties zijn bij programma's altijd mogelijk; er is meer dan één correcte uitwerking. In dit programma kunt u bijvoorbeeld ook eerst het dak tekenen, of u kunt het vierkant linksom in plaats van rechtsom tekenen.) We hebben een aantal regels opgenomen die beginnen met //; daarin wordt aangegeven wat de volgende opdrachten doen. Zulke regels zijn speciaal bedoeld voor de menselijke lezer van een programma; het JavaLogo-systeem doet er niets mee.

```
import logotekenap.*;

public class Huisje extends TekenApplet
{

  public void initialiseer()
  {
    maakTraceMogelijk();
  }

  public void tekenprogramma()
  {
    // maak de achtergrondkleur groen en zet de pen uit
    tekenblad.achtergrondkleur("groen");
    pen.uit();

    // teken eerst een grijs, gevuld vierkant
    pen.vulAan("grijs");
    pen.vooruit(100);
    pen.rechts(90);
    pen.vooruit(100);
    pen.rechts(90);
    pen.vooruit(100);
    pen.rechts(90);
    pen.vooruit(100);
    pen.rechts(90);
    pen.vulUit();

    // ga 100 pixels naar boven en draai 30□ naar rechts
    pen.vooruit(100);
    pen.rechts(30);

    // teken het rode dak
    pen.vulAan("rood");
    pen.vooruit(100);
    pen.rechts(120);
    pen.vooruit(100);
    pen.rechts(120);
    pen.vooruit(100);
    pen.vulUit();
  }
}
```

1.4 We zouden die vijf vierkanten dan kunnen tekenen met de volgende negen opdrachten:

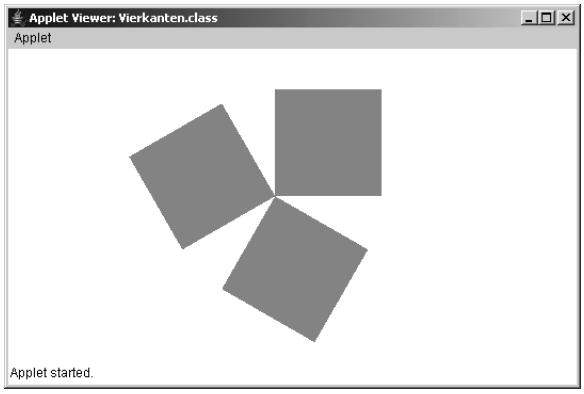
```
pen.vierkant(100);
pen.naarRechts(120);
pen.vierkant(80);
pen.naarRechts(100);
pen.vierkant(60);
pen.naarRechts(80);
pen.vierkant(40);
pen.naarRechts(60);
pen.vierkant(20);
```

1.5 Deze methode ziet er bijvoorbeeld als volgt uit:

```
public void driehoek(double zijde, String kleur)
{
  pen.vulAan(kleur);
  pen.vooruit(zijde);
  pen.rechts(120);
  pen.vooruit(zijde);
  pen.rechts(120);
```

```
  pen.vooruit(zijde);
  pen.rechts(120);
  pen.vulUit();
}
```

1.6 Er worden drie vierkanten in verschillende standen getekend, zie figuur 1.17.



FIGUUR 1.17 Drie vierkanten

1.7 Dit programma ziet er bijvoorbeeld als volgt uit (merk op, dat we bij alle methoden commentaar hebben opgenomen):

```
import logotekenap.*;

public class Driehoeken extends TekenApplet
{

  public void initialiseer()
  {
      maakTraceMogelijk();
  }

  // tekent vier geneste driehoeken in verschillende
  // kleuren
  public void tekenprogramma()
  {
    pen.uit();
    pen.rechts(30);
    driehoek(160, "zwart");
    naarBinnen();
    driehoek(120, "geel");
    naarBinnen();
    driehoek(80, "blauw");
    naarBinnen();
    driehoek(40, "rood");

  }

  // tekent een gelijkzijdige driehoek met de opgegeven
  // zijde en kleur
  public void driehoek(double zijde, String kleur)
  {
    pen.vulAan(kleur);
    pen.vooruit(zijde);
    pen.rechts(120);
    pen.vooruit(zijde);
    pen.rechts(120);
    pen.vooruit(zijde);
```

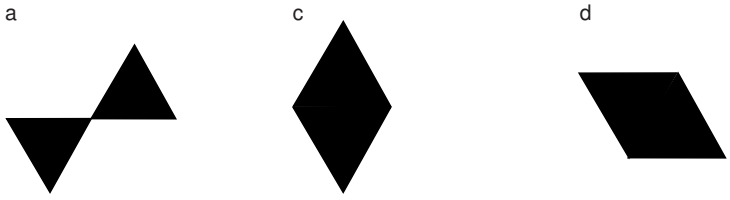
```
    pen.rechts(120);
    pen.vulUit();
  }

  // beweegt de pen naar het startpunt van de volgende
  // driehoek
  public void naarBinnen()
  {
    pen.rechts(30);
    pen.vooruit(24);
    pen.links(30);
  }
}
```

2 Uitwerking van de zelftoets

1 In de getoonde opdrachten draait de pen 30° + 60° + 60° + 120° = 270° naar rechts. Om weer op zijn uitgangsrichting terug te komen, moet dus 270° naar links of 90° naar rechts gedraaid worden. Er moet dus een opdracht pen.links(270) of pen.rechts(90) worden toegevoegd. Merk op, dat u deze vraag kunt beantwoorden zonder te weten wat er precies getekend wordt!

2 Het juiste antwoord is b. Om de bovenste driehoek in de juiste stand te krijgen, moet eerst 30° naar links worden gedraaid. Om vanuit hetzelfde punt de onderste driehoek rechtsom te tekenen, moet begonnen worden met de rechterzijde en dus 180° gedraaid worden.
Antwoord a tekent wel een zandloper, maar zet die scheef; de antwoorden c en d tekenen geen zandloper, maar twee driehoeken die samen een ruit vormen. Zie figuur 1.18.



FIGUUR 1.18 De figuren getekend door de onjuiste antwoorden

3 a In deze opdracht is de volgorde van de parameters onjuist: er moet eerst een getal staan, dan een String en dan weer een getal.
b Deze opdracht heeft maar twee parameters, terwijl er drie nodig zijn.
c Deze opdracht is juist.
d Onjuist; in een opdracht mogen geen typeaanduidingen voorkomen.
e Zie antwoord d.