OO-programmeren in Java met BlueJ

GERTJAN LAAN

ACADEMIC SERVICE

Tweede druk

OO-programmeren in Java met BlueJ

Tweede druk

Gertjan Laan

ACADEMIC SERVICE

Meer informatie over deze en andere uitgaven kunt u verkrijgen bij:
Sdu Klantenservice
Postbus 20014
2500 EA Den Haag
tel.: (070) 378 98 80
www.sdu.nl/service

Omslag: Studio Bassa, Culemborg
Zetwerk: Redactiebureau Ron Heijer, Markelo

ISBN 978 90 395 2705 4
NUR 123 / 989

Voorwoord

De traditionele aanpak bij het leren van Java of andere programmeertalen heeft meestal de *taalelementen* als uitgangspunt. Deze aanpak start vaak met primitieve typen als `int` en `double`, toekenningsopdrachten, operatoren, selecties en herhalingen, om dan via methoden geleidelijk over te gaan op klassen en objecten en alle begrippen die daarmee samenhangen.

Het is een beproefd recept waarmee de meeste programmeurs zijn opgevoed en opgegroeid. Het volgt bovendien de historische ontwikkeling die programmeertalen en -concepten hebben doorgemaakt: van gestructureerd, proceduregericht naar objectgeoriënteerd programmeren.

Een nadeel van deze aanpak is dat de kernbegrippen *objecten* en *klassen* relatief laat aan bod komen. Veel studenten blijken moeite te hebben met de mate van abstractie die bij deze begrippen hoort. Dat komt enerzijds door het feit dat mensen in het algemeen moeite hebben met abstracte begrippen die betrekking hebben op een gebied waarin ze nieuwkomer zijn. En anderzijds met de beperkte tijd die studenten meestal gegund is zich deze begrippen eigen te maken.

Dit pleit ervoor in een zo vroeg mogelijk stadium met objecten te beginnen. De vraag dringt zich op: hoe vroeg is *zo vroeg mogelijk*? Simpelweg met objecten beginnen heeft als nadeel dat een groot deel van de uitleg op drijfzand is gebouwd: docent en student staan voor de onmogelijke opgave zich tegelijkertijd rekenschap geven van lastige concepten als klasse en objecten, attributen, variabelen, typen, constructors, methoden, terugkeerwaarden en parameters. En dat alles in liefst één lesuur.

De komst van *BlueJ* maakt deze opgave een stuk makkelijker. Het programma stelt studenten in staat zelf instanties van voorgedefinieerde klassen te creëren en die zichtbaar te maken. Op een interactieve manier kunnen studenten het gedrag van objecten bestuderen en op een concrete en correcte manier kennismaken met attributen, constructors en methoden zonder zelf code te hoeven schrijven of zelfs maar te hoeven bekijken.

BlueJ is een krachtig hulpmiddel bij de introductie van objectgeoriënteerde begrippen, omdat het maken van een gebruikersinterface niet nodig is. Elke student kan zich daardoor volledig richten op het ontwerpen van domeinklassen en het testen daarvan in BlueJ.

Om te leren programmeren is BlueJ alleen niet voldoende. In goed en volledig programmeeronderwijs dient er aandacht te zijn voor ten minste deze vier facetten:

- taalelementen van Java;
- algoritmiek;
- objectgeoriënteerde concepten;
- analyse, ontwerp en UML.

In vrijwel alle boeken over programmeren komen klassen als het ware uit de lucht vallen. In de praktijk echter zijn klassen dikwijls het resultaat van analyse en ontwerp, testen, heroverwegen en opnieuw analyseren en ontwerpen. Ik vind het essentieel studenten in een zo vroeg mogelijk stadium kennis te laten maken met analyse en ontwerp, en het gebruik van UML daarbij. Het nadenken over klassen en hun onderlinge relaties, het maken van diagrammen en het communiceren daarover met medestudenten en docenten zorgt ervoor dat studenten zich in korte tijd deze concepten eigen kunnen maken.

Uitgangspunten van dit boek zijn dan ook analyse en ontwerpen, UML en objectgeoriënteerde concepten, zonder daarbij de taal Java en algoritmiek uit het oog te verliezen.
In deze tweede editie zijn voorbeelden en opgaven gestroomlijnd, fouten verbeterd, UML-diagrammen aangepast. Twee hoofdstukken (de hoofdstukken 10 en 12 uit de eerste editie) zijn online geplaatst en kunnen als pdf worden gedownload.

Aanvullende materiaal, zoals de broncode van de voorbeelden, de uitwerkingen van de opgaven en PowerPoint-presentaties bij de hoofdstukken zijn te vinden op de website van de uitgever: www.academicservice.nl en op mijn website www.gertjanlaan.com.

Opmerkingen van lezers ontvang ik graag per e-mail: GJ@gertjanlaan.nl

Zaandam, januari 2013

Gertjan Laan

Inhoud

Hoofdstuk 1

Klassen en objecten

1.1 Inleiding

De wereld om je heen bestaat uit objecten. Een raam, een stoel, een tafel, een mobiele telefoon. Als je een omschrijving van een tafel moet geven zeg je bijvoorbeeld: een zwarte, rechthoekige tafel. *Zwart* en *rechthoekig* zijn eigenschappen van de tafel. We zeggen dat de eigenschappen zwart en rechthoekig *attributen* (gegevens, data, velden) van de tafel zijn.

Veel objecten zijn gemaakt om iets te doen: voor een mobiele telefoon is het allerbelangrijkste dat hij in staat is op elke plek te bellen en gesprekken te ontvangen. Dit zijn *functies* (operaties, methoden) van het object.

Als je een zwarte tafel rood verft, verandert een van de attributen van die tafel. We zeggen dat de *toestand* (Engels: *state*) van het object verandert.

In Java maak je zogeheten objectgeoriënteerde programma's. In zulke programma's werk je met objecten. Het zijn geen fysieke objecten die je kunt vastpakken, maar objecten die in het geheugen van een computer bestaan. Om zo'n object te kunnen maken moet je eerst een *klasse* hebben. Een klasse is een beschrijving van de attributen en functies van gelijksoortige objecten. Als een programma een object maakt in overeenstemming met die beschrijving heet dat een *instantie* van de klasse (Engels: *instance*). De woorden object en instantie worden vaak door elkaar gebruikt.

In de volgende paragrafen zie je hoe je objecten kunt maken in BlueJ. Maar voor het zover is moet je eerst de juiste software installeren.

1.2 De nodige software installeren

Om met BlueJ te kunnen werken, moet je eerst de zogeheten Java Development Kit (JDK) installeren. Dit is een groot softwarepakket dat Java-ontwikkelaars gebruiken om programma's te schrijven. Ook BlueJ maakt gebruik van dit pakket.

1. Ga naar www.oracle.com/technetwork/java/javase/downloads/index.html.
2. Download de JDK die geschikt is voor je besturingssysteem. In plaats van JDK wordt ook de term Java Platform gebruikt in combinatie met Java SE, of ook wel Java SE Development Kit.

Mocht je de JDK op deze manier niet kunnen vinden, googel dan met de zoekterm *Java SE Development Kit*.

3. Installeer de JDK.

De volgende stap is het installeren van BlueJ.
1. Ga naar bluej.org.
2. Download en installeer BlueJ.

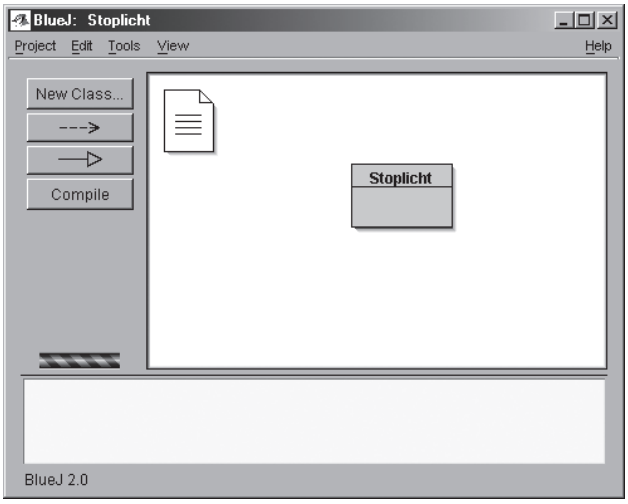
In dit hoofdstuk (en de volgende hoofdstukken) staan veel voorbeelden die je in BlueJ kunt uitvoeren. Deze voorbeelden kun downloaden van de website bij dit boek.
1. Ga naar www.gertjanlaan.com.
2. Ga naar OO-programmeren in Java met BlueJ 2e editie, en download de voorbeelden (zip-bestand).
3. Pak de voorbeelden uit in een map waarvan je de naam makkelijk kunt onthouden.

1.3 Een object maken in BlueJ

Oefening 1.1

1. Start BlueJ en open het project Stoplicht van de voorbeelden die je hebt gedownload (zie vorige paragraaf); klik op Project | Open project...
2. Navigeer naar de map waarin de voorbeelden staan.
3. Klik dubbel op de map Voorbeelden om deze te openen
4. Klik dubbel op de map Hoofdstuk 01 om deze te openen
5. Klik dubbel op het project Stoplicht om dit te openen

Het scherm van BlueJ ziet er nu uit zoals in figuur 1.1.



Figuur 1.1

In het midden van het witte gedeelte in het venster van BlueJ staat een rechthoekig figuur met de naam `Stoplicht`. Deze figuur symboliseert de klasse `Stoplicht`. We

noemen dit ook wel een *klassendiagram*. In de klasse `Stoplicht` staat een beschrijving voor stoplichten, zoals je verderop in dit hoofdstuk zult zien. Deze beschrijving is op dit moment niet zo van belang. Belangrijker is dat je met behulp van deze klasse stoplichtobjecten kunt maken. Zo veel je wilt. En elk van die stoplichten voldoet aan de beschrijving die in de klasse staat.

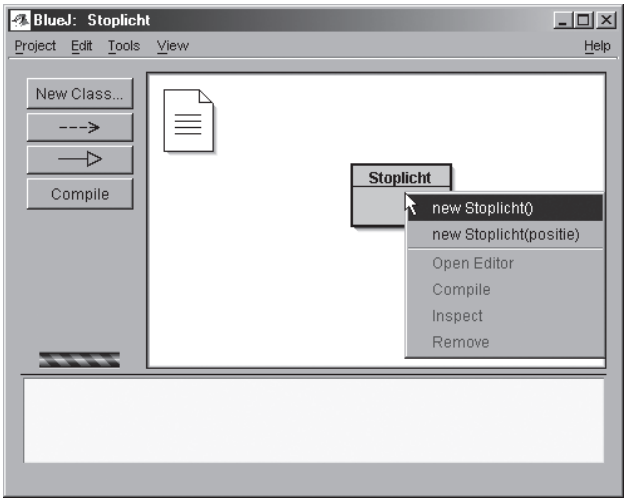
Oefening 1.2

Een instantie (object) van de klasse `Stoplicht` maken gaat als volgt:

1. Controleer eerst of de afbeelding van de klasse `Stoplicht` gearceerd is zoals in figuur 1.2.
2. Als de klasse gearceerd is, klik dan aan de linkerkant van het venster van BlueJ op de knop Compile; is de klasse niet gearceerd, ga dan door met het volgende punt.

Figuur 1.2

3. Klik met de rechtermuisknop op de klasse. Er verschijnt een snelmenu, zie figuur 1.3.



Figuur 1.3

4. Kies uit dit menu de bovenste optie: `new Stoplicht()`.

De uitdrukking `new Stoplicht()` heet het *aanroepen van een constructor* van de klasse. Een constructor is simpel gezegd een stukje code dat een instantie van de klasse maakt.

Het dialoogvenster Create Object verschijnt, zie figuur 1.4.

Figuur 1.4

In dit venster kun je een zelfverzonnen naam voor het object tikken. BlueJ stelt de naam `stoplich1` voor. Het is het simpelst deze naam te accepteren en op Ok te klikken.

De instantie die je zojuist gemaakt hebt, is zichtbaar als een rode rechthoek onder in het venster op de zogeheten *Object Bench* (objectenbank). In de rode rechthoek staat onderstreept de naam van het object en daarachter de klasse waar hij een instantie van is: `stoplich1:Stoplicht`.

1.3.1 Een bericht sturen

De rode rechthoek op de objectenbank symboliseert een object. In feite bevindt het `Stoplicht`-object zich in het geheugen van je computer en kun je het stoplicht opdrachten geven. Welke opdrachten dat zijn, kun je zien door met de rechtermuisknop op het rode object te klikken. Er verschijnt een menu met een hele lijst mogelijkheden. Dit zijn de functies waarover het stoplicht beschikt. In Java worden functies meestal *methoden* genoemd. De complete lijst van methoden van een object heet ook wel de *interface* van dat object. Door het kiezen van een van de methoden uit zijn interface kun je het stoplicht iets laten doen.

Oefening 1.3

1. Klik op `void maakZichtbaar()`.

In een apart venster verschijnt nu de afbeelding van het stoplicht. Alle lampen zijn uit.

Het laten uitvoeren van een methode van een object heet het *aanroepen* van een methode (*calling a method*). Een andere term die wel gebruikt wordt is het *sturen van een bericht* (*sending a message*).

2. Klik met de rechtermuisknop opnieuw op het rode object op de objectenbank en kies `void setGroen()`.

Hiermee stuur je het bericht `setGroen()` naar het stoplicht. Het stoplicht reageert hierop door de groene lamp aan te doen. Met de methoden `setOranje()` en `setRood()` kun je een andere lamp laten branden. Met `reset()` gaan alle lampen uit.

1.3.2 Meer objecten maken

Van een klasse kun je meer dan één object maken. Telkens moet je daartoe gebruik maken van de constructor van de klasse. Nu heeft de klasse `Stoplicht` niet een, maar twee constructors. Met de tweede constructor kun je aangeven op welke positie het stoplicht komen moet, gemeten in pixels vanaf de linkerkant van het venster.

Oefening 1.4

1. Klik met de rechtermuisknop op het *klassendiagram* van `Stoplicht` midden in het venster van BlueJ.
2. Kies uit het menu de tweede constructor: `new Stoplicht(positie)`.

Opnieuw verschijnt het dialoogvenster Create Object, maar nu met twee invoervakken, zie figuur 1.5.

Figuur 1.5

In het bovenste vakje staat een voorstel voor de naam van het nieuwe object: `stoplich2`. In het onderste vakje moet je een getal tikken voor de positie van het stoplicht.

3. Tik het getal 200 in het vakje en klik op Ok.

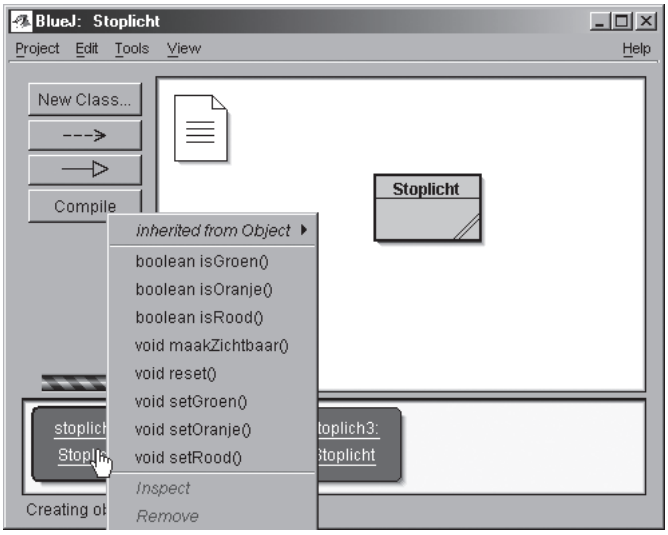
Op de objectenbank onder in het venster staan nu twee rode objecten: `stoplich1` en `stoplich2`. Maak het tweede stoplicht zichtbaar door met de rechtermuisknop op het tweede object te klikken en het bericht `maakZichtbaar()` te versturen. Dit stoplicht wordt 200 pixels vanaf de linkerkant van het venster getekend.

Er zijn nu twee stoplichten zichtbaar. Ze hebben hetzelfde uiterlijk en gedragen zich ook hetzelfde: het zijn instanties van dezelfde klasse. Tegelijkertijd zijn het onafhankelijke objecten. Het ene stoplicht kun je op groen zetten en het andere op rood. Of beide op groen.

Op dezelfde manier kun je met behulp van de tweede constructor een derde stoplicht maken op positie 300.

1.4 Methoden

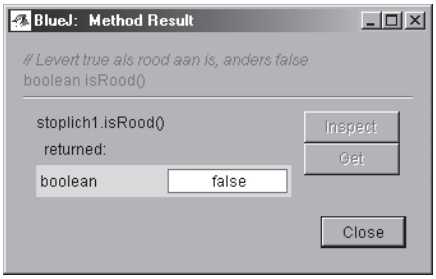
Elk object beschikt over een aantal methoden waarmee je het object iets kunt laten doen. Door rechts te klikken op een object toont BlueJ de lijst van methoden die bij dat object horen, zoals je in Figuur 6 kunt zien. Met de methoden `setRood()`, `setOranje()` en `setGroen()` verander je de kleur van het stoplicht. En daarmee verandert de *toestand* (state) van het object.



Figuur 1.6

1.4.1 Een methode die iets aflevert

Er zijn ook methoden die niets aan het object veranderen, maar die informatie verschaffen over het stoplicht: `isRood()`, `isOranje()` en `isGroen()`. Deze methoden vertellen of het betreffende stoplicht op dit moment rood, oranje of groen is. Wanneer je bijvoorbeeld de methode `isRood()` laat uitvoeren krijg je in een venster met de naam Method Result het resultaat van de methode, zoals in Figuur 1.7.



Figuur 1.7

Er staat in het venster:
returned:
`boolean false`.

Als een methode, zoals in dit geval, een waarde aflevert heet die waarde een *retourwaarde* of *terugkeerwaarde* (Engels: return value). Verderop zul je zien dat er allerlei soorten (typen) terugkeerwaarden zijn, `boolean` is een van de mogelijke typen.

Het woord `false` betekent *niet waar* en dat wil hier zeggen dat het betreffende stoplicht niet rood is. Als het wel rood is wordt de retourwaarde van de methode `true`. De waarden `true` (waar) en `false` (niet waar) zijn zogeheten `boolean` waarden, zie ook paragraaf 5.5.

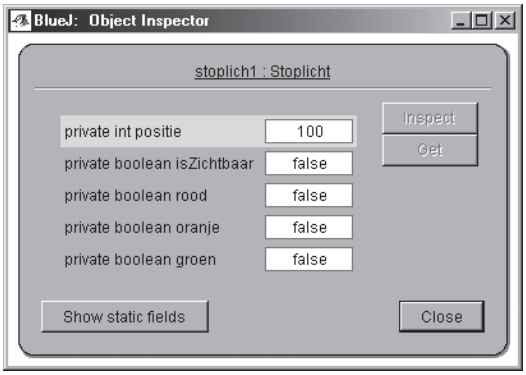
Als je met de rechtermuisknop op een object klikt zie je in het snelmenu (bijvoorbeeld in figuur 1.6) dat het type van de terugkeerwaarde vóór de naam van de methode staat: `boolean isRood()`.
Als een methode geen waarde aflevert staat er het woord `void`, zoals in `void reset()`. Het Engelse woord void betekent: leeg, niets.

1.4.2 Object-inspector van BlueJ

Uit de vorige paragraaf blijkt dat een stoplicht-object in staat is antwoord te geven op de vraag of de rode lamp wel of niet brandt. Zo'n object heeft blijkbaar een geheugen. Dat geheugen bevindt zich in het object in de vorm van *velden* (Engels: field) of *attributen*. De waarden van de attributen zijn opgeslagen in het geheugen van je computer. De attributen in een object zijn doorgaans afgeschermd voor de buitenwereld, het zijn privégegevens waar alleen het object zelf over kan beschikken. Maar met BlueJ kun je als het ware *in* het object kijken, en de privégegevens toch zichtbaar maken.

Het zichtbaar maken van de attributen van een object gaat met behulp van de Object Inspector:
1. Klik met de rechtermuisknop op een object.
2. Kies uit het menu *Inspect*.

Het venster van de Object Inspector verschijnt, zie figuur 1.8.

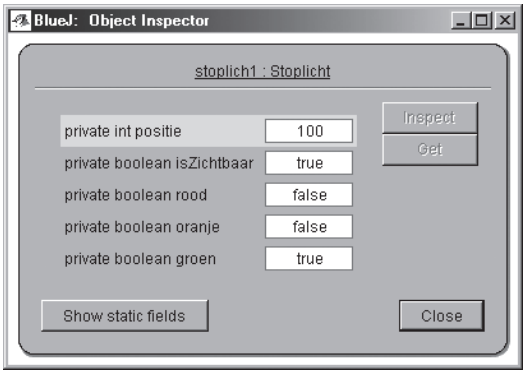


Figuur 1.8

De velden hebben de namen `positie`, `isZichtbaar`, `rood`, `oranje` en `groen`. In figuur 1.8 is te zien dat de positie van het stoplicht 100 is en dat de overige attributen allemaal de waarde `false` hebben.
Het woord `private` dat er telkens voor staat duidt erop dat dit privégegevens van het object zijn.

1.4.3 Methode die de toestand verandert

De Object Inspector is handig als je wilt weten in welke toestand een object zich bevindt. Als een van de attributen in een object een andere waarde krijgt, verandert de van toestand dat object. Een attribuut verandert vaak door het aanroepen van een methode. Voorbeelden van methoden die de toestand kunnen wijzigen zijn `maakZichtbaar()` en `setGroen()`. Na het aanroepen van deze methoden hebben de attributen de waarden die je in figuur 1.9 ziet.



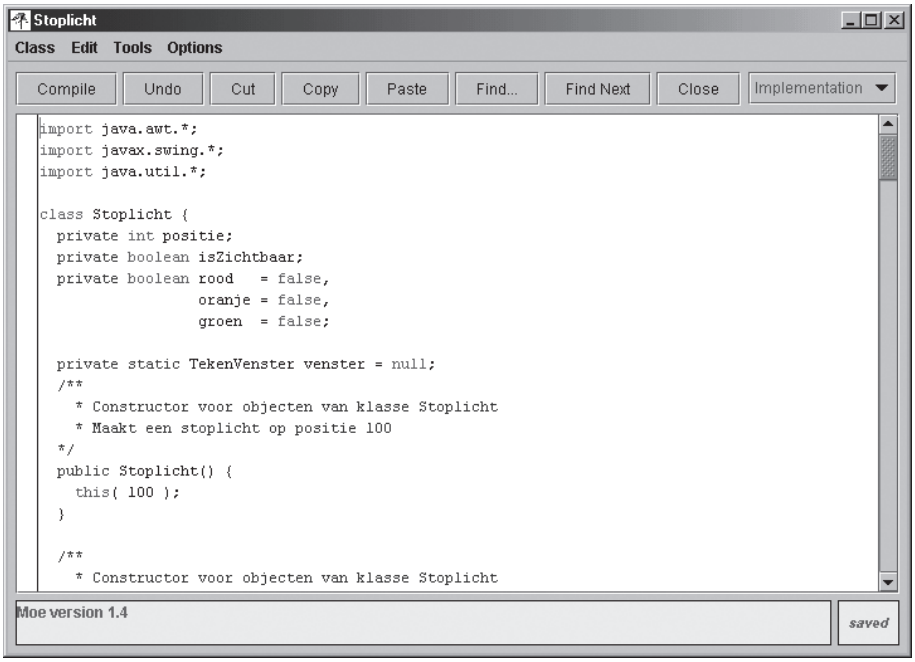
Figuur 1.9

1.5 Broncode

De definitie van elke klasse is vastgelegd in zogeheten *broncode* (Engels: source code). De broncode is geschreven in de programmeertaal Java en de broncode kun je in BlueJ zichtbaar maken:

1. Klik met de rechtermuisknop op de klasse `Stoplicht`.
2. Kies Open Editor.

In een apart venster verschijnt de broncode van de klasse, zie figuur 1.10.



Figuur 1.10

Het gaat er nu niet om precies te begrijpen hoe deze broncode in elkaar zit, maar als je in de broncode naar beneden scrollt kom je de definities van de verschillende methoden van het stoplicht tegen zoals `reset()`, `setRood()` en `setOranje()`.

Op de vijfde regel van boven begint de definitie van de klasse `Stoplicht`:

```
class Stoplicht {
  private int positie;
  private boolean isZichtbaar;
  private boolean rood   = false,
                  oranje = false,
                  groen  = false;
```

Zoals je ziet krijgen de attributen `rood`, `oranje` en `groen` alle drie de waarde `false`. Om die reden zal elke instantie van `Stoplicht` beginnen met alle lampen uitgeschakeld. Als je wilt kun je dit veranderen. De editor van BlueJ is een simpele tekstverwerker, vergelijkbaar met Windows Kladblok. Verander bijvoorbeeld de waarde van `rood` in `true`. Er komt dan te staan:

```
private boolean rood   = true,
                oranje = false,
                groen  = false;
```

Merk op dat het klassendiagram van `Stoplicht` nu gearceerd is, zoals in figuur 1.11.

Figuur 1.11

Dit duidt erop dat de broncode van de klasse opnieuw *vertaald* (gecompileerd) moet worden. Waarom zo'n vertaling nodig is leg ik uit in paragraaf 1.8. De broncode van een klasse kun je als volgt laten vertalen (compileren):

1. Klik in aan de linkerkant van het venster op de knop Compile.

Tijdens het compileren verdwijnen de gemaakte objecten van de objectenbank. Je kunt nu opnieuw instanties maken en bij deze stoplichten zal vanaf het begin de rode lamp branden.

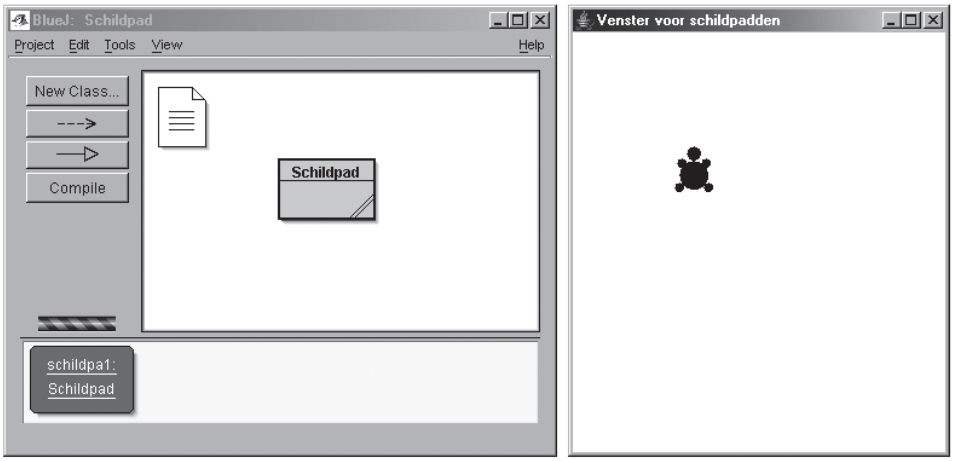
1.6 Een andere klasse

Een andere klasse die een grotere verscheidenheid aan methoden heeft dan het stoplicht van de vorige paragrafen is de klasse `Schildpad`.

Oefening 1.5

1. Sluit eventueel eerst het voorgaande project in BlueJ: kies in de menubalk Project | Close.
2. Open in de map met voorbeelden de map Hoofdstuk 01 en open het project met de naam Schildpad.
3. Maak een instantie van de klasse `Schildpad`.
4. Roep de methode `maakZichtbaar()` aan.

Het resultaat zie je in figuur 1.12.



Figuur 1.12

Een schildpad-object heeft een groot aantal methoden. Om te beginnen twee methoden om de schildpad naar links en naar rechts te laten draaien:

```
void draaiLinks(int)
void draaiRechts(int)
```

Achter de naam van de methode zie je tussen de haakjes het woord `int` staan. Dit is een afkorting van het Engelse woord integer, wat geheel getal betekent. Wanneer je de methode `draaiLinks()` of `draaiRechts()` aanroept moet je een geheel getal opgeven: het aantal graden dat de schildpad draaien moet. Zo'n getal heet een *argument* of *parameter*. Als je het bericht `draaiLinks()` naar de schildpad verzendt, stuur je het aantal graden dat hij draaien moet mee.

5. Laat de schildpad 45 graden naar links draaien.
6. Laat de schildpad 90 graden naar rechts draaien.

Belangrijk om te onthouden is dat je via het argument van een methode gegevens van buiten naar een object kunt sturen.

Andere methoden van de schildpad zijn:

```
void terug(double)
void vooruit(double)
void reset()
```

Met de eerste twee methoden kun je de schildpad een stukje (aantal pixels) achteruit respectievelijk vooruit laten lopen. Ook deze methoden hebben een argument: een `double`. In Java is een `double` een gebroken getal met een decimale punt, zoals 2.5 of 10.1, maar je mag ook gehele getallen opgeven. Maak de getallen niet te groot, anders loopt

de schildpad het venster uit. Mocht dat het geval zijn dan kun je hem met de methode `reset()` weer terugbrengen.

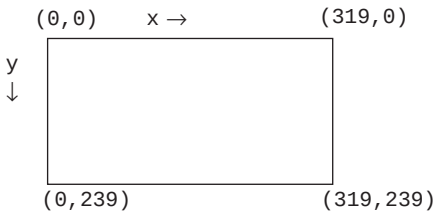
1.6.1 Methoden met een retourwaarde

De schildpad heeft drie methoden met een retourwaarde:

```
int getX()
int getY()
int getRichting()
```

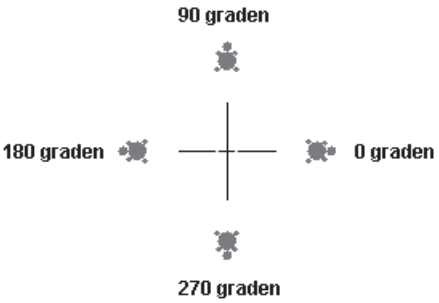
Deze methoden leveren een `int` (een geheel getal) als resultaat af. Respectievelijk de x-coördinaat van de vensterpositie van de schildpad, de y-coördinaat van de positie en de richting waarin de schildpad kijkt.

De venstercoördinaten beginnen linksboven; daar is het punt (0, 0). De x-coördinaat neemt toe van links naar rechts en de y-coördinaat neemt toe van *boven* naar *beneden*. Als je een venster hebt van 320 breed en 240 pixels hoog, dan kun je de coördinaten in beeld brengen als in figuur 1.13.



Figuur 1.13

De richting waarin de schildpad kijkt wordt in graden gemeten, zie figuur 1.14.



Figuur 1.14

Methoden met een retourwaarde zoals `getX()`, `getY()` en `getRichting()` leveren informatie af over de schildpad. Zulke methoden zorgen er dus voor dat een bepaald gegeven van een object naar buiten bekend wordt gemaakt.

1.6.2 Methode met een String-argument

Een methode waar je geen getal als argument moet invoeren is:

```
void setKleur(String)
```

Het argument is van het type `String`. Een string is in Java een rijtje letters of andere tekens, vaak is het een stukje tekst dat je kunt lezen. De methode `setKleur()` verwacht dat je de naam van een kleur invoert. Een string-argument moet je invoeren tussen aanhalingstekens, dus bijvoorbeeld `"blauw"`. Voor instanties van de klasse `Schildpad` zijn behalve `"blauw"` ook `"geel"`, `"groen"`, `"rood"` en `"zwart"` geldige kleuren. Als je een niet geldige kleur invoert, bijvoorbeeld `"paars"` dan wordt de schildpad met zwart getekend.

Oefening 1.6

1. Maak een gele en groene schildpad.
2. Maak een zwarte schildpad door een niet-geldige kleur in te voeren.

Als je de aanhalingstekens vergeet, krijg je van BlueJ een mededeling: *Error: incompatible types*.
Deze zogeheten *foutmelding* zegt dat het type van de invoer niet overeenkomt met het type dat de methode verwacht, een string.

1.7 Zelf een methode maken

Als je de broncode van een klasse hebt, kun je daar heel makkelijk een methode aan toevoegen. Op die manier kun je de *functionaliteit* uitbreiden van de objecten die je van de klasse maakt. Als voorbeeld neem ik de schildpad. Je kunt hem de opdracht `vooruit(50)` of `draaiRechts(90)` geven, maar per opdracht voert hij slechts één handeling uit.

Het zou aardig zijn als je hem een opdracht kunt geven waarmee hij meer dan één handeling uitvoert. Bij wijze van voorbeeld maak ik een methode die `loopL()` heet en waarmee elk schildpad-object een L-vormig stukje kan lopen. De methode `loopL()` bestaat uit de volgende opdrachten:

```
vooruit( 50 );
draaiRechts( 90 );
vooruit( 50 );
```

Omdat schildpadden trage beesten zijn, is het verstandig de schildpad na elke handeling even te laten slapen:

```
vooruit( 50 );
slaap();
draaiRechts( 90 );
slaap();
vooruit( 50 );
slaap();
```

De aanroep van de methode `slaap()` zorgt ervoor dat de schildpad 1 seconde wacht.

Deze zes opdrachten zet je tussen accolades, waarbij je ervoor zorgt dat elk van de zes opdrachten twee spaties naar rechts inspringt:

```
{
  vooruit( 50 );
  slaap();
  draaiRechts( 90 );
  slaap();
  vooruit( 50 );
  slaap();
}
```

Vervolgens geef je de methode een *kop* (header) waarin onder meer de naam van de methode staat, in dit geval `loopL()`. De complete methode komt er zo uit te zien:

```
public void loopL()
{
  vooruit( 50 );
  slaap();
  draaiRechts( 90 );
  slaap();
  vooruit( 50 );
  slaap();
}
```

Merk op dat alle opdrachten in `loopL()` bestaan uit het aanroepen van een of andere methode van de klasse `Schildpad`.

Je hoeft nu alleen nog `loopL()` aan de broncode van de klasse `Schildpad` toe te voegen.

Oefening 1.7

1. Open de editor zodat de broncode van de klasse `Schildpad` zichtbaar is.
2. Zoek in de editor de plaats waar je de volgende regel ziet:

```
    // Voeg hier eigen methode toe:
```

3. Typ onder deze regel de methode `loopL()`, precies zoals hij hierboven staat.

Let op de openingsaccolade op de tweede regel en de sluitaccolade op de laatste regel van de methode. Let ook op de puntkomma's aan het eind van elke opdracht, maar niet aan het eind van de kop van de methode!
Let op de hoofdletter `L` in `loopL()` en de hoofdletter `R` in `draaiRechts()`.

Laat de broncode vertalen:
4. Klik in de editor op de knop Compile.

Als je alles goed gedaan hebt, zie je onder in het venster van de editor de mededeling: Class compiled – no syntax errors. Dat betekent dat er geen fouten in de code zijn.
Als je niet alles goed gedaan hebt krijg je daarover een foutmelding. Probeer de fout op te sporen door je eigen versie van `loopL()` nog eens heel nauwkeurig te vergelijken met die hierboven, breng een verbetering aan en laat de klasse opnieuw vertalen.

Als er geen fouten zijn, sluit dan de editor, maak een instantie van de nieuwe klasse en laat met het bericht `loopL()` de schildpad een stukje lopen. Merk op dat als je dit bericht vier keer achter elkaar naar de schildpad stuurt, hij weer op dezelfde plek staat als waar hij begon.

1.8 Over Java

Java is rond 1990 ontworpen in de verwachting dat de taal gebruikt zou worden voor de programmering van allerlei elektronische apparatuur, zoals kopieermachines, afstandsbedieningen, televisies en talloze andere producten. Zulke apparatuur is voorzien van elektronische schakelingen in de vorm van chips. Dergelijke chips zijn in staat om gegevens te onthouden of een reeks instructies automatisch uit te voeren. Zo kan ik mijn televisie alle zenders van de kabel laten opzoeken en onthouden, als ik tegelijk op twee bepaalde knopjes van de afstandsbediening druk.

Instructies zijn meestal bij elkaar opgeslagen in een geheugenchip. Zo'n bij elkaar horende reeks instructies noemen we een *programma*. Een programma wordt uitgevoerd door een andere chip: de *processor*. Nu is het lastige dat de processor geen Nederlands, Engels of Java begrijpt. De enige taal die de processor begrijpt is *machinecode*. Om het nog ingewikkelder te maken bestaan er heel veel verschillende soorten machinecode. Elke fabrikant die een nieuwe processor maakt kan daarvoor een eigen, nieuwe machinecode ontwerpen.

Dit gebrek aan standaardisatie is natuurlijk tijdrovend en kost de industrie veel geld. Ook computers hebben onder dit gebrek aan standaardisatie te lijden, omdat computers voor een groot deel uit chips bestaan. Behalve pc's (of microcomputers) zijn er bij grote bedrijven en universiteiten vaak computers die veel groter zijn dan een pc: mini-

computers, of nog groter: zogenaamde *mainframes*. Het verschil tussen die computers zit vooral in de belangrijkste chip, de *(micro)processor*. Bijvoorbeeld:

- een pc met een microprocessor van het merk Intel;
- een mobiele telefoon met een ARM-processor;
- een Sun-computer met een SPARC-processor.

Al die verschillende processors begrijpen uitsluitend hun eigen specifieke machinecode. Daar komt nog bij dat machinecode voor mensen onleesbaar is: het bestaat uit lange rijen nullen en enen. Programmeurs lossen dit probleem op door eerst een programma te schrijven in een zogenaamde *hogere programmeertaal*, en dat programma vervolgens te laten vertalen naar een specifieke machinecode. In een hogere programmeertaal kun je instructies voor een computer opschrijven met behulp van Engelse en Nederlandse woorden. Ook een dergelijke reeks instructies heet een *programma*. Dat programma is voor mensen met enige oefening goed te lezen, maar voor een microprocessor absoluut niet. Er moet een *vertaler* (compiler) aan te pas komen om het programma om te zetten in machinecode.

Zolang je als programmeur met één soort computer en dus één soort machinecode te maken hebt, lijkt er niet veel aan de hand. Veel lastiger wordt het als je te maken krijgt met alle soorten computers die waar ook ter wereld op internet aangesloten zijn. Een programma dat je geschreven hebt voor de Macintosh, dat wil zeggen dat vertaald is naar machinecode voor de Macintosh, draait absoluut niet op de miljoenen pc's en waarschijnlijk ook niet op alle overige computers.

In werkelijkheid is de situatie nog ernstiger: computers beschikken niet alleen over een processor, maar ook over een *besturingssysteem* dat zijn eisen stelt aan de programma's die op die computer uitgevoerd kunnen worden. Bekende besturingssystemen voor pc's zijn Linux en allerlei varianten van Windows en Mac OS. Een programma dat specifiek geschreven is voor Windows zal niet zonder meer werken op een computer met Linux.

De combinatie van een bepaald type processor met een bepaald besturingssysteem noemen we een *platform*. Omdat er veel verschillende typen processors zijn, en bij elke processor vaak weer een handvol besturingssystemen, zijn er dus heel veel verschillende platformen op de wereld. Een programmeur die zijn of haar programma zo breed mogelijk wil verspreiden, ziet zich gesteld voor de onmogelijke taak voor al die platformen een aparte versie te maken.

1.8.1 De oplossing van Java

Java maakt gebruik van een ingenieuze oplossing voor het probleem van de programmeur en de vele platformen. Het idee van deze oplossing is al heel oud, maar de uitvoering ervan op zo'n grote schaal is nieuw. Het ingenieuze idee bestaat uit twee gedeelten:

- Laat elk Java-programma door een compiler vertalen naar een gestandaardiseerde *tussentaal* die betrekkelijk dicht tegen machinecode aan zit.
- Voorzie elk platform van een programma dat de tussentaal begrijpelijk maakt voor deze specifieke processor.

De tussentaal wordt meestal *Java-bytecode* genoemd. Het programma dat de bytecode voor de processor begrijpelijk maakt heet de *Java Virtual Machine*, vaak afgekort tot *JVM*, of in het Nederlands: virtuele machine.

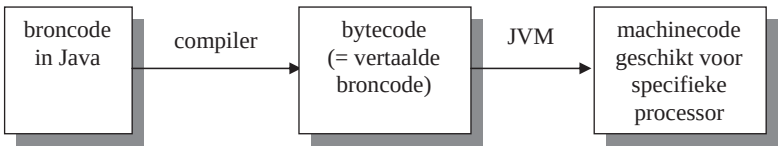
Omdat bytecode niet zo verschrikkelijk veel van alle soorten machinecodes verschilt, is het vertalen van bytecode niet zo'n grote klus en kan de JVM een tamelijk klein programma zijn. De verspreiding van die gratis JVM's is geen probleem: daar is het internet goed voor.

De JVM wordt geleverd als onderdeel van de Java Runtime Environment (JRE). Daarmee kun je de JVM installeren als *plug-in* (aanvullend programma) voor een webbrowser. In een browser kun je dan *applets* draaien: relatief kleine Java-programma's die in een webpagina functioneren.

De JVM wordt via de JRE ook geïnstalleerd als zelfstandig programma waarmee je zogeheten Java-*applicaties* kunt draaien. Een Java-applicatie kan er voor een Windows-gebruiker uitzien als een gewoon Windows-programma. De JRE en daarmee de virtuele machine voor applicaties en voor een browser zijn gratis te krijgen bij www.oracle.com. Als je eerder de JDK hebt geïnstalleerd (zie paragraaf 1.2) dan heb je automatisch ook de JRE geïnstalleerd.

Het probleem van de programmeur die voor elk platform een apart programma moet creëren, is met de komst van Java dus opgelost. De voorwaarde is wel dat er voor elk platform een Java Virtuele Machine gemaakt moet worden en dat deze gemakkelijk te krijgen moet zijn. Voor de meeste platformen was dit binnen twee jaar na de introductie van Java in 1995 het geval.

In het schema in figuur 1.15 is de hele gang van zaken nog eens in beeld gebracht.



Figuur 1.15

Als programmeur (en als gebruiker van BlueJ) heb je niet voldoende aan de JRE, onder andere omdat daar geen compiler in zit. Je moet als programmeur over de zogeheten Java SE Development Kit beschikken (Java SE), vaak afgekort tot JDK. Hierin zit een compiler en heel veel voorgedefinieerde klassen die je in Java-programma's kunt gebruiken. In de latere hoofdstukken maak je kennis met sommige van die klassen. Zoals gezegd is de JRE een onderdeel van de JDK. De JDK bestaat in drie verschillende soorten:

- SE = Standard Edition, voor gewoon gebruik;
- EE = Enterprise Edition, voor bedrijfstoepassingen;
- ME = Micro Edition, voor gebruik in mobiele apparatuur.

Voor het gebruik met dit boek heb je Java SE nodig, zie ook paragraaf 1.2.

1.9 Samenvatting

In dit hoofdstuk heb je kennisgemaakt met BlueJ en met de eerste begrippen en principes die te maken hebben met objectgeoriënteerd programmeren.

- Je hebt een klasse nodig om objecten (instanties) te maken.
- Een klasse bevat de definitie voor de objecten die je er van kunt maken.
- Van een klasse kun je net zoveel instanties maken als je wilt.
- Elk instantie van een bepaalde klasse beschikt over dezelfde soort attributen (velden, gegevens).
- Elk instantie van een bepaalde klasse beschikt over dezelfde methoden (functies).
- Het aanroepen van een methode is het activeren van die methode en heet ook wel het sturen van een bericht naar een object.
- De toestand van een object is de verzameling van waarden van de attributen.
- Als een van de attributen van waarde verandert, komt het object in een andere toestand.
- De toestand van een object kan door het aanroepen van een methode veranderen.
- Een methode kan een argument hebben, waardoor je gegevens naar een object kunt sturen.
- Een methode kan een retourwaarde hebben, waardoor een object gegevens kan terugsturen.
- Gegevens bestaan er in verschillende typen: `int` voor gehele getallen, `double` voor getallen met een decimale punt en `String` voor tekst.
- Aan de broncode van een klasse kun je zelf methoden toevoegen, waardoor je de functionaliteit van de objecten uitbreidt.
- Als je in de broncode van een klasse een wijziging aanbrengt moet je de klasse opnieuw laten vertalen (compileren) voor je er objecten van kunt maken.
- Een compiler vertaalt een Java-klasse naar bytecode.
- Bytecode is een tussentaal die veel lijkt op machinecode.
- Bytecode wordt door een programma dat Java Virtual Machine (JVM) heet omgezet naar de machinecode van een specifieke processor.
- De JVM, compiler en veel andere Java-voorzieningen zijn gratis verkrijgbaar in de vorm van de Java SE Development Kit.

1.10 Vragen

1. Wat is een ander woord voor object?
2. Objecten beschikken doorgaans over gegevens. Welke andere namen zijn in omloop voor de gegevens van een object?
3. Objecten beschikken doorgaans over functies. Welke naam wordt in Java meestal gebruikt voor een functie?
4. Wat betekent: *het aanroepen van een methode* van een object?
5. Wat betekent: *het sturen van een bericht* naar een object?

6. Wat is een klasse?
7. Wat is broncode?
8. Wat is compileren?
9. Waarom moet je broncode compileren?
10. Hoe kun je informatie naar een object sturen?
11. Hoe kan een object informatie terugsturen?
12. Hoeveel instanties kun je van een klasse maken?
13. Hebben alle instanties van een bepaalde klasse dezelfde attributen?
14. Beschikken alle instanties van een bepaalde klasse over dezelfde methoden?
15. Als je twee instanties hebt van dezelfde klasse, hebben de attributen van die twee objecten dan altijd dezelfde waarden?
16. Wat is de toestand van een object?
17. Wat zijn verschillen tussen JRE en JDK?

1.11 Opgaven

1. Bedenk voor de klasse `Schildpad` een methode `loopVierkant()` waarmee je de schildpad een vierkantje kunt laten lopen.
   Aanwijzing: maak in deze methode gebruik van `loopL()` uit paragraaf 1.7. Voeg de methode `loopVierkant()` toe aan de klasse `Schildpad` en test of hij goed werkt.
2. Breid de klasse `Stoplicht` uit met een methode `knipper()` waarmee je het oranje licht een aantal malen kunt laten knipperen.
   Aanwijzing: Kopieer de methode `slaap()` uit de klasse `Schildpad` en gebruik deze voor het knipperen.
3. Maak een object van de klasse `Stoplicht`. Open de Object Inspector en laat deze open. Roep een van de methoden van het stoplicht aan waarmee de toestand van het object verandert en merk op dat de waarde(n) in de Object Inspector automatisch veranderen.

Index

BlueJ is een krachtig hulpmiddel bij de introductie van objectgeoriënteerde begrippen, omdat het maken van een gebruikersinterface niet nodig is. Elke student kan zich daardoor volledig richten op het ontwerpen van klassen, het maken van objecten en het testen daarvan in BlueJ.
In vrijwel alle boeken over programmeren komen klassen als het ware uit de lucht vallen. In de praktijk zijn klassen echter vaak het resultaat van analyse en ontwerp, testen, heroverwegen en opnieuw analyseren en ontwerpen. Het is daarom belangrijk studenten in een zo vroeg mogelijk stadium kennis te laten maken met analyse en ontwerp, en het gebruik van UML. Door het nadenken over klassen, hun onderlinge relaties en het maken van diagrammen kunnen studenten zich in korte tijd deze concepten eigen maken.

In dit boek ligt dan ook vanaf het begin de klemtoon op zorgvuldige analyse en ontwerp, zonder daarbij de aandacht voor de taal Java, controlestructuren en algoritmiek te verwaarlozen. Hierbij wordt op een zinvolle en correcte manier gebruik gemaakt van UML en gedemonstreerd wat dit alles in Java-code betekent.
In deze tweede druk zijn voorbeelden en opgaven gestroomlijnd, onvolkomenheden verbeterd en UML-diagrammen aangepast. Aanvullende materiaal, zoals de broncode van de voorbeelden, extra hoofdstukken over over exception handling en streams, de uitwerkingen van de opgaven en PowerPoint-presentaties bij de hoofdstukken zijn te vinden op websites www.academicservice.nl en www.gertjanlaan.com.
De heldere stijl, de aansprekende voorbeelden en vele vragen en opgaven maken dit boek tot een perfecte introductie in software engineering met Java.

Gertjan Laan is werkzaam als docent informatica in het hoger onderwijs. Hij geeft al jaren les in programmeertalen, waaronder C++ en Java. Eerder zijn van hem de boeken *Aan de slag met C++*, *Aan de slag met Ruby*, *En dan is er... Java* en *Datastructuren in Java*.

ISBN 978 90 395 2705 4
NUR 123/989

9 789039 527054

www.academicservice.nl