HOOFDSTUK 1

Objecten en klassen

Belangrijkste concepten in dit hoofdstuk:

- objecten
- klassen
- methodes
- parameters

We springen meteen in het diepe en maken een begin met onze behandeling van objectgeoriënteerd programmeren. Om te leren programmeren heb je naast enige theoretische kennis vooral veel oefening nodig. In dit boek komen beide aan bod, zodat ze elkaar versterken.

De kern van objectoriëntatie wordt gevormd door twee basisconcepten die we als eerste moeten begrijpen: objecten en klassen. Deze concepten vormen de basis van programmeren in objectgeoriënteerde talen. Laten we dus eens beginnen met een korte behandeling van deze twee basisbegrippen.

1.1 Objecten en klassen

CONCEPT

Java-**objecten** fungeren als model voor de elementen waarin een probleem kan worden opgesplitst.

Als je een computerprogramma schrijft in een objectgeoriënteerde taal, creëer je op de computer een model van een klein stukje van de werkelijkheid. De onderdelen waarmee het model wordt opgebouwd, zijn de objecten waarin het probleem kan worden opgesplitst. Deze objecten moeten op de een of andere manier worden weergegeven in het te maken computermodel. De objecten van het probleemdomein hangen af van het te schrijven programma. Als je een tekstverwerker maakt kan dit om woorden en alinea's gaan, bij een sociaalnetwerksysteem om gebruikers en berichten, en bij een computergame om monsters.

Objecten kunnen worden ingedeeld in categorieën en een klasse beschrijft – op een abstracte manier – alle objecten van een bepaalde soort.

CONCEPT

Objecten worden gemaakt uit **klassen**. De klasse beschrijft de aard van een object; de objecten zelf zijn opzichzelfstaande exemplaren van de klasse.

We zullen deze abstracte begrippen verduidelijken aan de hand van een voorbeeld. Veronderstel dat je een simulatie van een verkeerssituatie wilt maken. Een van de entiteiten waar je dan rekening mee zult moeten houden, zijn bijvoorbeeld auto's. Wat is een auto in onze context; is het een klasse of een object? Door onszelf een paar vragen te stellen kunnen we hierover een beslissing nemen.

Welke kleur heeft een auto? Hoe snel kan een auto rijden? Waar bevindt de auto zich op dit moment?

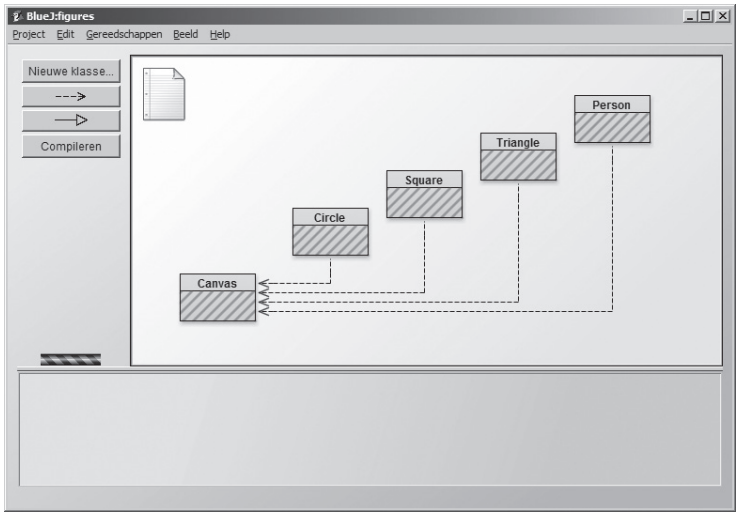
Het zal duidelijk zijn dat deze vragen niet beantwoord kunnen worden als we het niet hebben over één bepaalde auto. Dat komt doordat het woord 'auto' in onze context verwijst naar de *klasse* Auto; we spreken over auto's in het algemeen en niet over één bepaalde auto.

Als ik zeg: 'Mijn oude auto, die bij mij thuis in de garage geparkeerd staat', kunnen we wel de hierboven gestelde vragen beantwoorden. Die auto is rood, is niet zo heel erg snel en hij staat op dit moment in mijn garage. Nu hebben we het over een object; een specifiek exemplaar van een auto. Als we het over een bepaald object hebben, spreken we vaak van een *instantie*. Vanaf nu zullen we de term instantie vaak gebruiken. Instantie is nagenoeg synoniem met object. We zullen objecten instanties noemen wanneer we willen benadrukken dat ze van een bepaalde klasse zijn (zoals: 'dit object is een instantie van de klasse Auto').

Voor we verdergaan met deze theoretische verhandeling zullen we eerst een voorbeeld bekijken.

1.2 Objecten creëren

Start BlueJ en open het project *figures*.[1] Op het scherm zie je een dialoogvenster zoals in figuur 1.1.



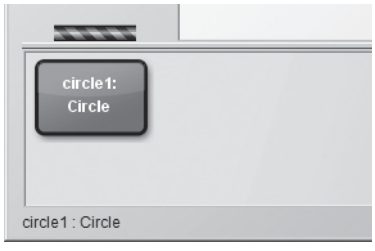
FIGUUR 1.1 *Het project shapes in BlueJ*

In dit venster moet nu een schema weergegeven worden. Elk van de gekleurde rechthoeken in het schema stelt een klasse in dit project voor. In dit project hebben we de klassen `Circle`, `Square`, `Triangle`, `Person` en `Canvas` genoemd.

Rechtsklik op de klasse `Circle` en selecteer de optie

```
new Circle()
```

in het contextmenu. Het systeem vraagt je om een 'Naam van instance'. Klik op OK, aangezien de voorgestelde naam op dit moment goed genoeg is. Aan de onderzijde van het scherm zie je nu een rode rechthoek met het opschrift 'circle1' (figuur 1.2).



FIGUUR 1.2 *Een object in de objectenbank*

1 In dit boek gaan we ervan uit dat je regelmatig bepaalde handelingen en oefeningen zult uitvoeren. Op dit moment veronderstellen we dat je al weet hoe je BlueJ start en de voorbeeldprojecten opent. Als dat niet het geval is, moet je eerst bijlage A doornemen.

AFSPRAAK

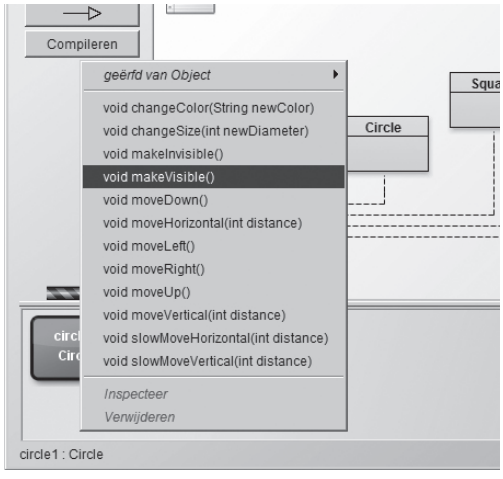
We noteren de namen van klassen met een hoofdletter (zoals `Circle`) en de namen van objecten met een kleine letter (zoals `circle`). Op die manier is het altijd duidelijk waar we het over hebben.

Oefening 1.1 Maak een andere cirkel. Maak daarna een vierkant.

Je hebt zojuist je eerste object gemaakt! 'Circle', het rechthoekige pictogram in figuur 1.1, representeert de klasse `Circle`; `circle1` is een object dat gemaakt is op basis van deze klasse. Het gebied waar het object is weergegeven, aan de onderzijde van het scherm, heet de *objectenbank*.

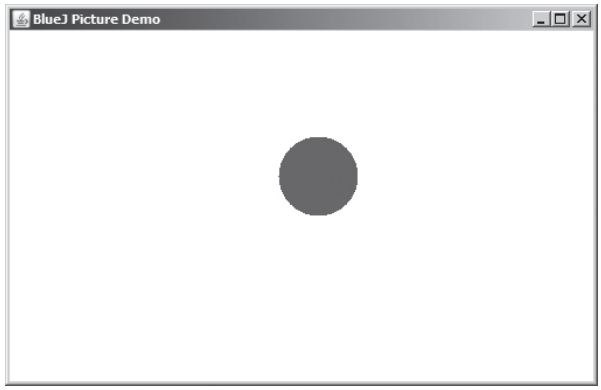
1.3 Methodes aanroepen

Wanneer je rechtsklikt op een van de cirkelobjecten (niet op de klasse!), wordt een contextmenu met verschillende bewerkingen geopend (figuur 1.3). Selecteer de optie `makeVisible` in het menu. Hierdoor wordt een nieuw venster met daarin een representatie van deze cirkel geopend (figuur 1.4).



FIGUUR 1.3 *Het contextmenu van een object met de bewerkingen*

In het menu bij de cirkel zie je nog andere bewerkingen. Bekijk wat er gebeurt als je een paar keer `moveRight` en `moveDown` selecteert om de cirkel dichter naar een van de hoeken van het scherm te verplaatsen. Met de opties `makeInvisible` en `makeVisible` kun je de cirkel verbergen en weer zichtbaar maken.



FIGUUR 1.4 *Een afbeelding van een cirkel*

CONCEPT

We kunnen objecten manipuleren door er **methodes** op toe te passen. Als we een methode toepassen, zullen objecten meestal iets doen.

Oefening 1.2 Wat gebeurt er als je de methode `moveDown` twee keer aanroept? Of drie keer? Wat gebeurt er als je de methode `makeInvisible` twee keer aanroept?

De opties in het menu van de cirkel representeren bewerkingen waarmee je de cirkel kunt manipuleren. In Java worden dit *methodes* genoemd. De gebruikelijke terminologie is dat methodes *aangeroepen* of *toegepast* worden. Vanaf nu zullen ook wij deze terminologie gebruiken. Zo zal het kunnen gebeuren dat we je vragen 'om de methode `moveRight` van `circle1` toe te passen'.

1.4 Parameters

Pas nu de methode `moveHorizontal` toe. Je zult zien dat je dan in het dialoogvenster iets moet invoeren (figuur 1.5). Voer de waarde 50 in en klik op OK. Je ziet nu dat de cirkel 50 pixels naar rechts verplaatst wordt.[2]

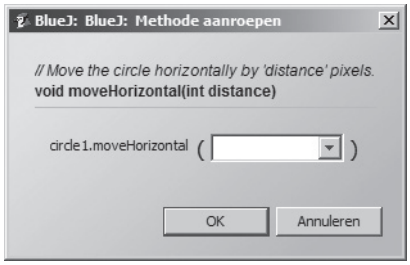
CONCEPT

Methodes kunnen **parameters** hebben die aanvullende informatie voor een bepaalde taak bevatten.

De methode `moveHorizontal`, die je zojuist hebt aangeroepen, is zodanig geschreven dat deze alleen maar kan worden toegepast met aanvullende informatie. In dit geval is de benodigde informatie de afstand waarover de cirkel moet worden verplaatst. De methode `moveHorizontal` is dus flexibeler dan de methodes `moveRight` en `moveLeft`.

2 Een pixel is een enkel stipje op je scherm. Het scherm is opgebouwd uit een raster van afzonderlijke pixels.

De laatste twee verplaatsen de cirkel altijd over een vaste afstand, terwijl je bij `moveHorizontal` exact kunt aangeven hoe ver je de cirkel wilt verplaatsen.



FIGUUR 1.5 *Een dialoogvenster bij een methodeaanroep*

> **Oefening 1.3** Voordat je verdergaat met lezen, pas je eerst de methodes `moveVertical`, `slowMoveVertical` en `changeSize` toe. Zoek uit hoe je `moveHorizontal` kunt gebruiken om de cirkel 70 pixels naar links te verplaatsen.

CONCEPT

De naam van een methode samen met de parametertypes wordt de **signatuur** genoemd. Deze bevat informatie die nodig is om die methode aan te roepen.

De extra waarden die je bij sommige methodes moet invoeren, worden *parameters* genoemd. Een methode geeft aan wat voor parameters deze nodig heeft. Wanneer je bijvoorbeeld de methode `moveHorizontal` aanroept (zie figuur 1.5), wordt in het dialoogvenster de volgende regel weergegeven.

```
void moveHorizontal(int distance)
```

Dit wordt de *header* (soms ook wel *signatuur*) van de methode genoemd. De header bevat aanvullende informatie over de betreffende methode. Het deel tussen haakjes (`int distance`) is de informatie over de benodigde parameter. Voor elke parameter definieert de header een *type* en een *naam*. De header hierboven geeft aan dat de methode werkt met een parameter van type `int` en met naam `distance`. De naam is een aanwijzing voor welke gegevens van de variabele worden verwacht. De naam van de methode en de parametertypes in de header worden samen de *signatuur* van de methode genoemd.

1.5 Datatypes

Een type geeft aan welke soort gegevens geschikt zijn als parameter. Het type `int` betekent gehele getallen (in het Engels ook 'integers' genoemd, vandaar de afkorting 'int').

CONCEPT

Parameters hebben een **type**. Het type geeft aan welke soorten waarden een parameter kan hebben.

In het bovenstaande voorbeeld geeft de signatuur van de methode `moveHorizontal` aan dat deze, voordat de methode kan worden uitgevoerd, eerst een geheel getal als invoer nodig heeft, dat aangeeft over welke afstand het object moet worden verplaatst. In het invoerveld in figuur 1.5 kun je vervolgens die waarde invoeren.

In de voorbeelden tot nu toe hebben we alleen nog maar het datatype `int` gezien. De parameters van de verplaatsingsmethodes en de methode `changeSize` zijn allemaal van dit type.

Als we beter kijken naar het contextmenu van het object zien we dat de methode-opties in het menu ook het parametertype bevatten. Als een methode geen parameter heeft, wordt de naam van de methode gevolgd door haakjes met niets daartussen. Als een methode wel een parameter heeft, worden het type en de naam van die parameter weergegeven. In de lijst met methodes voor een cirkel zul je zien dat er ook een methode is met een ander parametertype: de methode `changeColor` heeft een parameter van het type `String`.

Het type `String` geeft aan dat de methode een stukje tekst (bijvoorbeeld een woord of een zin) verwacht. Strings worden altijd tussen dubbele aanhalingstekens geplaatst. Om bijvoorbeeld het woord *rood* als string in te voeren, typ je "`rood`" in.

Het methodeaanroepdialoogvenster bevat ook een stukje tekst, het *commentaar*, dat boven de header van de methode wordt weergegeven. Met behulp van commentaren krijgt de (menselijke) lezer aanvullende informatie. We komen daar in hoofdstuk 2 op terug. Het commentaar van de methode `changeColor` beschrijft welke kleuren het systeem kent.

Oefening 1.4 Pas de methode `changeColor` toe op een van je cirkelobjecten en voer de string "`red`" in. Hierdoor moet de kleur van de cirkel veranderen. Probeer ook andere kleuren.
Oefening 1.5 Dit is een erg eenvoudig voorbeeld en je kunt maar weinig kleuren gebruiken. Bekijk wat er gebeurt als je een nog onbekende kleur invoert.
Oefening 1.6 Pas de methode `changeColor` toe en voer de kleur in het parameterveld in zonder de aanhalingstekens. Wat gebeurt er?

VALKUIL

Vaak vergeten beginners om de dubbele aanhalingstekens te plaatsen wanneer ze gegevens van het type `String` invoeren. Als je `green` in plaats van **"green"** invoert, zal het systeem een foutmelding genereren zoals 'error: cannot find symbol – variable green' (Fout: onbekend symbool).

Java ondersteunt verschillende andere gegevenstypes, zoals decimale getallen en tekens. We zullen daar nu niet dieper op ingaan. We komen daar echter verderop in dit boek op terug. Als je er nu al meer over wilt weten, kun je bijlage B bekijken.

1.6 Verschillende instanties

> **Oefening 1.7** Maak een aantal cirkelobjecten in de objectenbank. Je kunt dat doen door de optie `new Circle()` te selecteren in het contextmenu van de klasse `Circle`. Maak ze zichtbaar en verspreid ze vervolgens over het scherm met de methode `move`. Maak een van de cirkels groot en geel en een andere klein en groen. Probeer ook de andere vormen: maak een paar driehoeken, vierkanten en personen. Verander hun posities, afmetingen en kleuren.

CONCEPT

Verschillende instanties. Uit één klasse kunnen veel gelijkaardige objecten worden gemaakt.

Zodra je de beschikking hebt over een klasse kun je net zo veel objecten (of instanties) van die klasse maken als je maar wilt. Uit de klasse `Circle` kun je een groot aantal cirkels maken. Uit de klasse `Square` kun je een groot aantal vierkanten maken.1.7

Elk van deze objecten heeft een eigen positie, kleur en afmeting. Je kunt een kenmerk van een object (zoals de afmetingen ervan) veranderen door een methode op dat object toe te passen. Die wordt dan alleen toegepast op dat specifieke object, dus niet op de andere objecten.

Misschien valt je nog iets anders op met betrekking tot parameters. Kijk bijvoorbeeld eens naar de methode `changeSize` van de driehoek. De header ervan is

```
void changeSize(int newHeight, int newWidth)
```

Dit is een voorbeeld van een methode met meer dan één parameter. Deze methode heeft er twee, die door een komma in de header van elkaar worden gescheiden. Methodes kunnen een willekeurig aantal parameters hebben.

1.7 Toestand

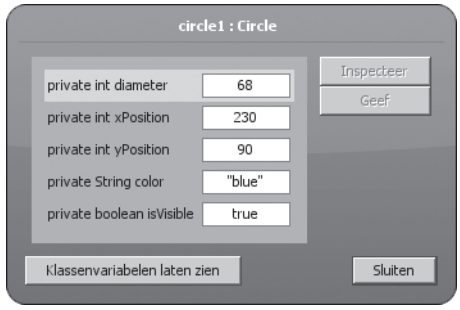
CONCEPT

Objecten hebben een **toestand**. De toestand wordt aangeduid door waarden in velden op te slaan.

Velden zijn kleine geheugenplaatsen binnen in een object die gebruikt kunnen worden om waarden op te slaan. De set waarden van alle *velden* of *attributen* waarmee een object wordt gedefinieerd (zoals de x-coördinaat, de y-coördinaat, de kleur, de diameter en de (on)zichtbaarheid van een cirkel) wordt ook wel de *toestand* van het object genoemd. Dit is een ander voorbeeld van gebruikelijke terminologie die we in dit boek over zullen nemen.

In BlueJ kun je de toestand van een object controleren door de optie *Inspecteer* te selecteren in het contextmenu van dat object. Wanneer je een object nader bekijkt, verschijnt er een *objectinspector*. De objectinspector is een vergrote weergave van het object dat de erin opgeslagen attributen weergeeft (figuur 1.6).

Oefening 1.8 Zorg ervoor dat er verschillende objecten in de objectenbank aanwezig zijn en bekijk ze stuk voor stuk. Probeer de toestand van een object te veranderen (bijvoorbeeld door de methode `moveLeft` toe te passen) terwijl de objectinspector geopend is. Als het goed is, zie je dat de waarden in de objectinspector dan veranderen.



FIGUUR 1.6 *Een objectinspector met informatie over een object*

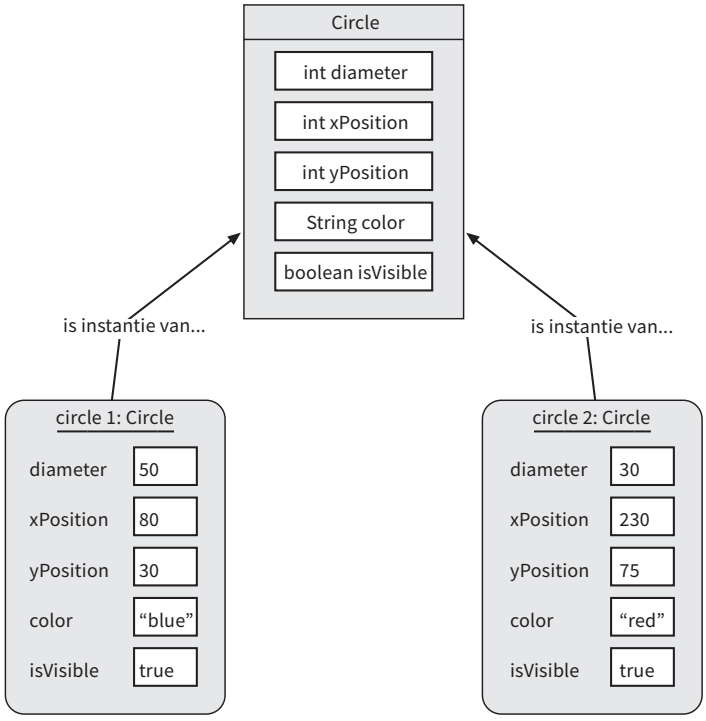
Sommige methodes veranderen, wanneer ze worden aangeroepen, de toestand van een object. De methode `moveLeft` bijvoorbeeld verandert het attribuut `xPosition`. In Java worden deze objectattributen *velden* genoemd.

1.8 Wat zit er in een object?

Wanneer je verschillende objecten bekijkt, zul je zien dat objecten van *dezelfde* klasse allemaal dezelfde velden hebben. Het aantal, het type en de namen van de velden zijn gelijk, maar de waarde van een bepaald veld kan in elk object verschillen. Objecten van een *andere* klasse kunnen echter andere velden hebben. Een cirkel heeft bijvoorbeeld een veld `diameter`, maar een driehoek heeft de velden `width` (voor de breedte) en `height` (voor de hoogte).

Dat komt doordat het aantal, de types en de namen van velden gedefinieerd zijn in een klasse en niet in een object. De klasse `Circle` definieert dat elk cirkelobject vijf velden zal hebben met de namen `diameter`, `xPosition`, `yPosition`, `color` en `isVisible`. De klasse definieert ook de types voor deze velden. Ook definieert de klasse dat de eerste drie van het type `int` zijn, dat de kleur van het type `String` is en de vlag `isVisible` van het type `boolean` is. (Het type `boolean` kan slechts twee waarden, `true` – waar – en `false` – onwaar –, hebben. Daarom wordt een attribuut van dit type ook wel een *vlag* genoemd. We komen later op dit type terug.)

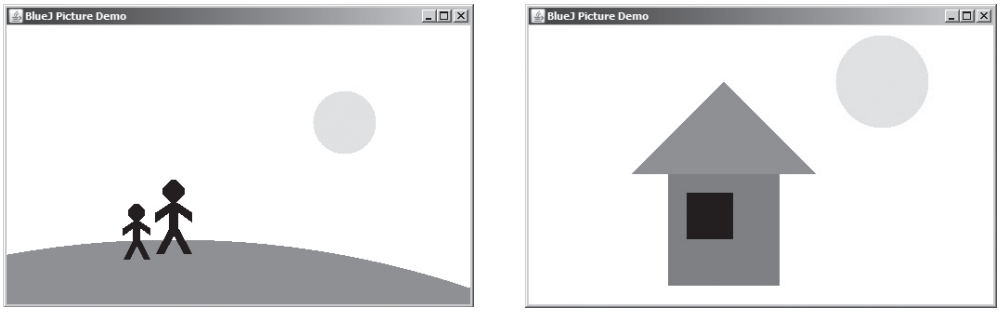
Wanneer een object van de klasse `Circle` gemaakt wordt, zal het automatisch deze velden hebben. De waarden van deze velden worden opgeslagen in het object. Daarmee wordt bereikt dat elke cirkel een kleur heeft, maar ook dat elke cirkel een andere kleur kan hebben (figuur 1.7).



FIGUUR 1.7 *Een klasse en de bijbehorende objecten met velden en waarden*

Voor methodes kunnen we hetzelfde verhaal vertellen. Methodes worden gedefinieerd in de klasse van het object. Dat betekent dat alle objecten van een bepaalde klasse dezelfde methodes hebben. De methodes worden echter toegepast op objecten. Hierdoor is het altijd duidelijk welk object op welk moment gewijzigd moet worden als bijvoorbeeld de methode `moveRight` wordt toegepast.

> **Oefening 1.9** In figuur 1.8 zijn twee verschillende afbeeldingen weergegeven. Kies een van deze afbeeldingen en maak deze opnieuw met behulp van de vormen van het project *figures*. Noteer welke handelingen je hebt uitgevoerd om het resultaat te bereiken. Zou je hetzelfde resultaat ook op andere manieren hebben kunnen bereiken?



FIGUUR 1.8 *Twee afbeeldingen die zijn opgebouwd uit een aantal vormen*