

# ➤ OO-programmeren in Java met BlueJ

GERTJAN LAAN



# OO-programmeren in Java met BlueJ

Tweede druk

Gertjan Laan



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](http://www.sdu.nl/service)

© 2013 Sdu Uitgevers, Den Haag  
Academic Service is een imprint van Sdu Uitgevers bv.

Omslag: Studio Bassa, Culemborg  
Zetwerk: Redactie bureau Ron Heijer, Markelo

ISBN 978 90 395 2705 4  
NUR 123 / 989

Alle rechten voorbehouden. Alle intellectuele eigendomsrechten, zoals auteurs- en databankrechten, ten aanzien van deze uitgave worden uitdrukkelijk voorbehouden. Deze rechten berusten bij Sdu Uitgevers bv en de auteur.

Behoudens de in of krachtens de Auteurswet gestelde uitzonderingen, mag niets uit deze uitgave worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand of openbaar gemaakt in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen of enige andere manier, zonder voorafgaande schriftelijke toestemming van de uitgever.

Voor zover het maken van reprografische verveelvoudigingen uit deze uitgave is toegestaan op grond van artikel 16 h Auteurswet, dient men de daarvoor wettelijk verschuldigde vergoedingen te voldoen aan de Stichting Reprorecht (Postbus 3051, 2130 KB Hoofddorp, [www.reprorecht.nl](http://www.reprorecht.nl)). Voor het overnemen van gedeelte(n) uit deze uitgave in bloemlezingen, readers en andere compilatiewerken (artikel 16 Auteurswet) dient men zich te wenden tot de Stichting PRO (Stichting Publicatie- en Reproductierechten Organisatie, Postbus 3060, 2130 KB Hoofddorp, [www.cedar.nl/pro](http://www.cedar.nl/pro)). Voor het overnemen van een gedeelte van deze uitgave ten behoeve van commerciële doeleinden dient men zich te wenden tot de uitgever.

Hoewel aan de totstandkoming van deze uitgave de uiterste zorg is besteed, kan voor de afwezigheid van eventuele (druk)fouten en onvolledigheden niet worden ingestaan en aanvaarden de auteur(s), redacteur(en) en uitgever deswege geen aansprakelijkheid voor de gevolgen van eventueel voorkomende fouten en onvolledigheden.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the publisher's prior consent.

While every effort has been made to ensure the reliability of the information presented in this publication, Sdu Uitgevers neither guarantees the accuracy of the data contained herein nor accepts responsibility for errors or omissions or their consequences.

# 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;
- algoritmieken;
- 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 algoritmieken 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.academicsservice.nl](http://www.academicsservice.nl) en op mijn website [www.gertjanlaan.com](http://www.gertjanlaan.com).

Opmerkingen van lezers ontvang ik graag per e-mail: [GJ@gertjanlaan.nl](mailto:GJ@gertjanlaan.nl)

Zaandam, januari 2013

Gertjan Laan

# Inhoud

Voorwoord	v	
1	Klassen en objecten	1
1.1	Inleiding	1
1.2	De nodige software installeren	1
1.3	Een object maken in BlueJ	2
1.3.1	Een bericht sturen	4
1.3.2	Meer objecten maken	5
1.4	Methoden	6
1.4.1	Een methode die iets aflevert	6
1.4.2	Object-inspector van BlueJ	7
1.4.3	Methode die de toestand verandert	8
1.5	Broncode	9
1.6	Een andere klasse	10
1.6.1	Methoden met een retourwaarde	12
1.6.2	Methode met een String-argument	13
1.7	Zelf een methode maken	13
1.8	Over Java	15
1.8.1	De oplossing van Java	16
1.9	Samenvatting	18
1.10	Vragen	18
1.11	Opgaven	19
2	Zelf klassen maken	21
2.1	Inleiding	21
2.2	Ontwerpen op basis van verantwoordelijkheid	21
2.2.1	Verantwoordelijkheden van een bankrekening	22
2.2.2	Het type van de attributen en methoden	23
2.2.3	De implementatie van Bankrekening	24
2.2.4	Nieuwe klasse maken in BlueJ	24
2.2.5	Commentaar in broncode	27
2.2.6	De klasse	28
2.2.7	Implementatie van de methoden van Bankrekening	28
2.3	Over methoden	29
2.4	Namen in Java	30
2.4.1	Regels voor de naamgeving in Java	31
2.5	Een constructor	31
2.5.1	Constructor toevoegen	32
2.5.2	Een tweede constructor voor Bankrekening	34
2.5.3	Formele en actuele argumenten of parameters	35

2.6	Opmaak van de broncode	35
2.6.1	Regels wit	37
2.7	Inhoud van een object op het scherm zetten	38
2.7.1	Concatenatie van strings	39
2.7.2	Over System.out.println() en System.out.print()	39
2.8	Een object dat zelf objecten maakt	40
2.8.1	De klasse Bank	40
2.8.2	De bank maakt rekeningen	42
2.8.3	Een externe methode aanroepen	44
2.8.4	Een lokale variabele	45
2.8.5	Initialisatie van lokale variabelen	46
2.8.6	Scope van lokale variabelen en attributen	47
2.8.7	Een interne methodeaanroep	48
2.8.8	De referentie this	48
2.9	Samenvatting	48
2.10	Vragen	50
2.11	Opgaven	50
3	Analyse en ontwerp	51
3.1	Inleiding	51
3.2	Schoolklas, deel 1	51
3.2.1	Analyse	51
3.2.2	Een eerste klassendiagram	53
3.2.3	Multipliciteit	54
3.2.4	Overzicht multipliciteiten in UML	55
3.2.5	Analyse, ontwerp en implementatie	55
3.2.6	Een klasse in UML	56
3.3	Ontwerp van Kind	56
3.4	Ontwerp van Datum	57
3.4.1	Notatieverschillen Java en UML	58
3.4.2	Implementatie van Datum: getters en setters	59
3.4.3	Nogmaals this	60
3.5	Implementatie van Kind	62
3.5.1	De methode toString()	63
3.6	Default-constructor	64
3.6.1	Automatische default-constructor	65
3.7	Constructor-overloading	65
3.7.1	In de ene constructor de andere aanroepen: this()	68
3.7.2	Ambigüiteit	69
3.7.3	Directe initialisatie van object-variabelen	70
3.8	De standaardingredienten van een klasse	71
3.9	Samenvatting	72
3.10	Vragen	72
3.11	Opgaven	73

4	Collecties	77
4.1	Inleiding	77
4.2	Een ArrayList voor de Bank	77
4.2.1	Het project Bank2	77
4.2.2	Packages en import	80
4.3	Broncode van Bank	81
4.3.1	Toevoegen aan ArrayList	82
4.3.2	Het maken van de nummers	83
4.3.3	Een if-statement	84
4.3.4	De for-each loop	85
4.3.5	Integer.parseInt()	86
4.4	Klassendiagram	86
4.4.1	Methode overloading	86
4.4.2	Duplicateer geen code	88
4.4.3	Een private methode	89
4.5	Schoolklas, deel 2	90
4.5.1	Ontwerp	91
4.5.2	Implementatie	91
4.5.3	Schoolklas in BlueJ	92
4.5.4	Het bewaren van objecten in BlueJ	93
4.5.5	UML-klassendiagram van de schoolklas	96
4.6	PC-printersysteem	96
4.6.1	Use cases	97
4.6.2	Analyse van het PC-printersysteem	99
4.6.3	Domeinklassen	100
4.6.4	Multipliciteiten	100
4.6.5	Objectendiagram	101
4.6.6	Navigeerbaarheid	102
4.6.7	PC-printersysteem in BlueJ	102
4.6.8	Implementatieklassen	104
4.6.9	De broncode van het PC-printersysteem	105
4.7	Sms-dienst	108
4.7.1	Analyse van de sms-dienst	110
4.7.2	De functionaliteiten	111
4.7.3	De broncode van Provider	111
4.7.4	De broncode van Mobiel	113
4.7.5	De broncode van SMS	114
4.7.6	De standaardmethode toString()	114
4.8	Javadoc	115
4.8.1	Het schrijven van documentatie voor javadoc	116
4.8.2	Indeling van de tekst voor javadoc	117
4.9	Samenvatting	119
4.10	Vragen	120
4.11	Opgaven	120



5	Keuzes	123
5.1	Inleiding	123
5.2	De typen int en double	123
5.2.1	Rekenkundige operatoren	124
5.2.2	Delen met int: de gehele deling	124
5.2.3	Delen met double: de normale deling	125
5.2.4	Omzetten van int naar double: typecasting	126
5.2.5	Omzetten van double naar int	127
5.2.6	Prioriteiten	127
5.2.7	Het type char	128
5.3	Meer operatoren	129
5.3.1	De toekenningsoperator +=	130
5.3.2	Andere toekenningsoperatoren	130
5.3.3	De increment-operator ++	131
5.3.4	De decrement-operator	131
5.4	Algoritme: de kassa	133
5.4.1	De kassa in BlueJ	133
5.5	Relationele en logische operatoren	134
5.5.1	Relationele operatoren	134
5.5.2	Logische operatoren: de en-operator &&	136
5.5.3	Operanden	137
5.5.4	De of-operator	138
5.5.5	De niet-operator !	138
5.5.6	Een boolean-variabele	139
5.5.7	Deelbaarheid onderzoeken	140
5.6	Het if-statement	141
5.6.1	Een ingewikkelder voorbeeld	143
5.7	Het if-else-statement	144
5.8	Het switch-statement	146
5.8.1	Meer cases op een rij	149
5.8.2	Switch of if-else?	150
5.8.3	Switch met char	151
5.9	Constante: final	151
5.10	Enumeratie	153
5.10.1	Enum met een constructor	155
5.11	Een methode voor toevalsgetallen	156
5.11.1	Andere toevalsgetallen	157
5.11.2	Static import	158
5.11.3	Andere methoden uit de klasse Math	158
5.12	Wanneer zijn twee strings gelijk?	159
5.13	Samenvatting	162
5.14	Vragen	162
5.15	Opgaven	163

6	Herhalingen en algoritmen	169
6.1	Inleiding	169
6.2	Het for-statement	169
6.2.1	Controlegedeelte van het for-statement	170
6.2.2	De body van het for-statement	172
6.2.3	Zet geen puntkomma na het controlegedeelte	172
6.2.4	De tafel van 13	172
6.3	Variaties met een for-statement	174
6.3.1	Andere beginwaarden dan 0 of 1	174
6.3.2	Terugtellen	174
6.3.3	Grotere stappen	175
6.3.4	Variabele begin- en eindwaarden	175
6.3.5	Een for-statement waarvan de body niet wordt uitgevoerd	175
6.3.6	Een for-statement met een char	176
6.4	StringBuffer	176
6.4.1	De methode setCharAt()	178
6.5	Het while-statement	178
6.6	Formatteren met printf()	181
6.6.1	Andere format specifiers	183
6.6.2	Scheidingstekens in getallen	185
6.7	Problemen met while-statement	186
6.7.1	Oneindige herhaling	186
6.7.2	Gelijkwaardigheid van for-statement en while-statement	187
6.8	Het do-while-statement	188
6.9	Algoritmen	188
6.9.1	Algoritme voor het verwisselen van twee waarden	188
6.9.2	Algoritme voor het sorteren van drie waarden	189
6.9.3	Algoritme voor rechthoekige lay-out	189
6.9.4	De methode printRechthoek	190
6.10	Kassa 2	192
6.10.1	Wikkelklasse	193
6.10.2	ArrayList met double	194
6.11	Algoritme: zoeken in een lijst	196
6.11.1	Een kind zoeken	196
6.11.2	De broncode van het zoekalgoritme	197
6.12	Arrays	199
6.12.1	Een array met int-waarden	199
6.12.2	De lengte van een array	202
6.12.3	De grenzen van de index	202
6.12.4	Array en for-each	203
6.12.5	Een array vullen met een for-statement	203
6.12.6	Een array met doubles	203
6.12.7	Verkorte declaratie en initialisatie	204
6.12.8	Kopiëren van een array	206
6.13	Algoritme voor een lay-out	208
6.13.1	Algoritme voor driehoekige lay-out	208
6.13.2	Een genest for-statement	209

6.14	Bubble sort	210
6.14.1	Sorteren van een rijtje van 5 getallen	212
6.14.2	De broncode van bubble sort	213
6.14.3	Hoe efficiënt is bubble sort?	214
6.14.4	Sorteer algoritme uit de Java-library	215
6.15	Samenvatting	216
6.16	Vragen	217
6.17	Opgaven	218
7	Polymorfie en overerving	223
7.1	Inleiding	223
7.2	Het project Bank3	223
7.2.1	De operator instanceof	226
7.2.2	Statische en dynamische type	226
7.2.3	De klassen Bankrekening en Spaarrekening	227
7.3	Overerving	229
7.3.1	Klassendiagram met overerving	230
7.3.2	Het project Bank4	233
7.3.3	Private en protected	234
7.3.4	Een Bankrekening is een Rekening	235
7.4	Het project Rechthoek	236
7.5	Het project Rechthoek2	237
7.5.1	FlexRechthoek in BlueJ	238
7.5.2	Constructor en overerving	238
7.5.3	Aanroep van constructor met super()	239
7.6	Overriding	240
7.7	Dynamische binding	242
7.7.1	Gebreken van Bank3	243
7.7.2	Superioriteit van Bank4	244
7.7.3	Polymorfie	246
7.7.4	De klasse Object	246
7.8	Samenvatting	247
7.9	Vragen	248
7.10	Opgaven	248
8	Gebruikersinterface	249
8.1	Inleiding	249
8.2	Maken van een gebruikersinterface	250
8.3	JPanel	250
8.3.1	JPanel in een Swing-applet	251
8.4	JPanel in een Swing-applicatie	254
8.5	Event-afhandeling	256
8.5.1	Wat gebeurt er precies?	259
8.6	Paneel als aparte klasse	260
8.6.1	Een applet met hetzelfde paneel	262

8.7	Een tekstvak	263
8.7.1	Een andere constructor van JTextField	265
8.7.2	Een tekstvak en een knop	265
8.8	Twee knoppen en een tekstvak	267
8.8.1	Twee knoppen, een handler	268
8.9	JTextField-event	270
8.10	Kleur en HTML in componenten	270
8.10.1	Componenten en kleur	270
8.10.2	Componenten en HTML	271
8.11	Lay-out van de componenten	271
8.11.1	Lay-outmanager uitschakelen	272
8.11.2	De coördinaten van JPanel	274
8.12	BorderLayout	275
8.12.1	BorderLayout met minder dan vijf componenten	277
8.13	GridLayout	278
8.14	Invoer van gehele getallen	279
8.14.1	Invoer van een geheel getal via een tekstvak	279
8.14.2	Verkorte manier voor het omzetten van string naar int	281
8.14.3	NumberFormatException	281
8.14.4	Lokale variabelen	282
8.14.5	Twee lokale variabelen met dezelfde naam	282
8.14.6	NullPointerException	282
8.15	Invoer van double	283
8.16	Componenten uitschakelen	284
8.16.1	Een tekstvak uitschakelen	284
8.17	Gebruikersinterface koppelen aan domeinklassen	285
8.17.1	Gebruikersinterface voor de klasse Datum	285
8.17.2	Gebruikersinterface voor de klasse Kind	290
8.18	Samenvatting	292
8.19	Vragen	293
8.20	Opgaven	293
9	Polymorfie met interfaces	295
9.1	Inleiding	295
9.2	Tekenen met Java	295
9.2.1	Over de naam van de grafische context	297
9.2.2	Tekst	297
9.2.3	Kleur	298
9.2.4	Rechthoeken en ellipsen	299
9.2.5	Vierkant en cirkel	300
9.2.6	Gevulde rechthoeken en ellipsen	300
9.2.7	De constructor van Tekenpaneel	301
9.3	Rondingen en bogen	302
9.3.1	De methoden drawRoundRect() en fillRoundRect()	302
9.3.2	De methoden drawArc() en fillArc()	303
9.3.3	Nieuwe kleuren maken	304

9.4	Een methode met de grafische context	304
9.4.1	Methode voor een willekeurige driehoek	305
9.5	Een interactieve applet	308
9.5.1	Het klassendiagram	309
9.5.2	Navigeerbaarheid in het klassendiagram	310
9.5.3	De broncode van Draw	311
9.5.4	De methode repaint()	313
9.5.5	Drie event handlers	314
9.6	Polygonen	315
9.7	Polymorfie via een interface	318
9.7.1	Referenties van een interface-type	321
9.7.2	De interface ActionListener	326
9.8	Een abstracte klasse	326
9.9	Meer dan één interface implementeren	330
9.9.1	Interfaces en abstracte klassen	331
9.10	Samenvatting	331
9.11	Vragen	332
9.12	Opgaven	333
10	Afbeeldingen en animaties	335
10.1	Inleiding	335
10.2	Tellen met een timer	335
10.2.1	Een Font-object	338
10.3	Starten en stoppen van de timer	339
10.4	De vallende baksteen	341
10.5	De stuiterende bal	343
10.5.1	Het stuiteren	344
10.5.2	Formule voor de parabool	345
10.5.3	De rest van de broncode	346
10.6	Een afbeelding van schijf: ImageIcon	347
10.6.1	Afbeelding op een label of knop	349
10.7	Animatie met botsingen	351
10.7.1	Onderlinge botsingen	353
10.7.2	De klasse Point	354
10.7.3	De interface Plaatje	355
10.7.4	Botsen tegen de rand van het venster	355
10.7.5	De broncode van AbstractPlaatje	356
10.7.6	Concrete klassen	357
10.7.7	De klasse Animatiepaneel	358
10.7.8	Het controleren van botsingen	361
10.7.9	Sublijst	361
10.8	Samenvatting	363
10.9	Vragen	363
10.10	Opgaven	363
	Index	365

## 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](http://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, google 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](http://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.

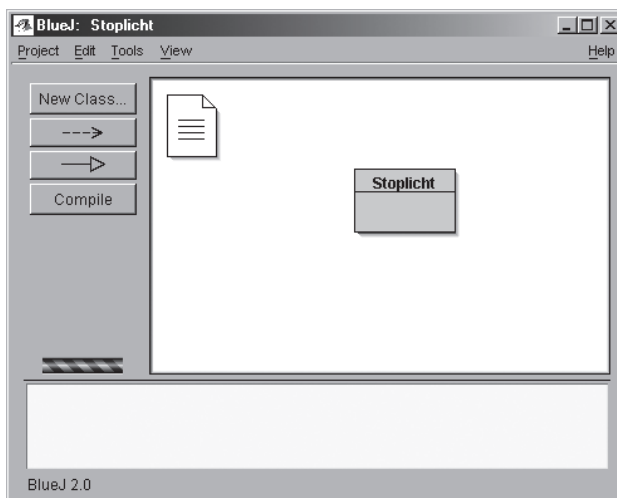
1. Ga naar [www.gertjanlaan.com](http://www.gertjanlaan.com).
2. Ga naar OO-programmeren in Java met BlueJ 2<sup>e</sup> editie, en download de voorbeelden (zip-bestand).
3. Pak de voorbeelden uit in een map waarvan je de naam makkelijk kunt onthouden.

### 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

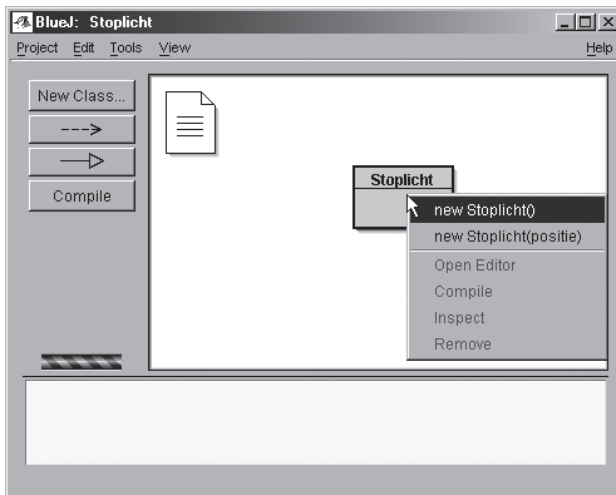
Een instantie (object) van de klasse `Stoplicht` maken gaat als volgt:

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`.

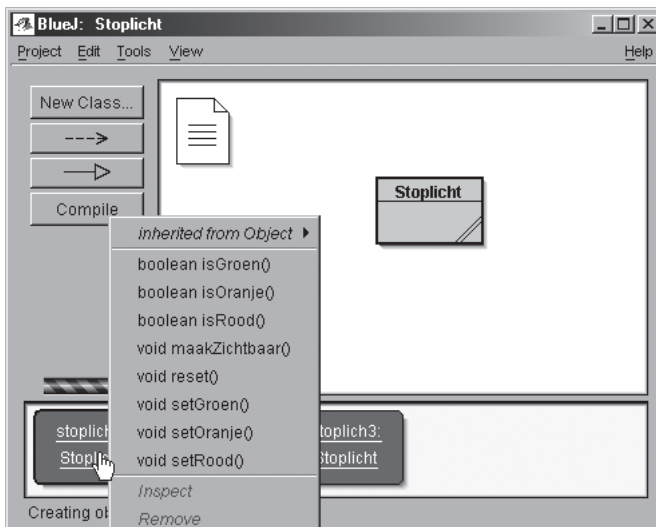
Op de objectenbank onder in het venster staan nu twee rode objecten: `stoplich1` en `stoplich2`. Maak het tweede stoplicht zichtbaar door met de rechtermuisknop op het tweede object te klikken en het bericht `maakZichtbaar()` te versturen. Dit stoplicht wordt 200 pixels vanaf de linkerkant van het venster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

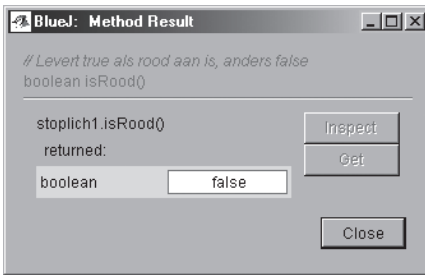
Elk object beschikt over een aantal methoden waarmee je het object iets kunt laten doen. Door rechts te klikken op een object toont BlueJ de lijst van methoden die bij dat object horen, zoals je in Figuur 6 kunt zien. Met de methoden `setRood()`, `setOranje()` en `setGroen()` verander je de kleur van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

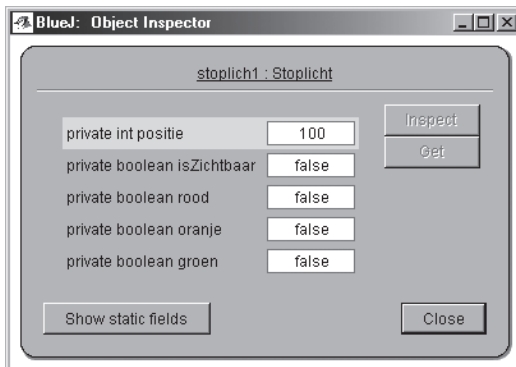
#### 1.4.2 Object-inspector van BlueJ

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.



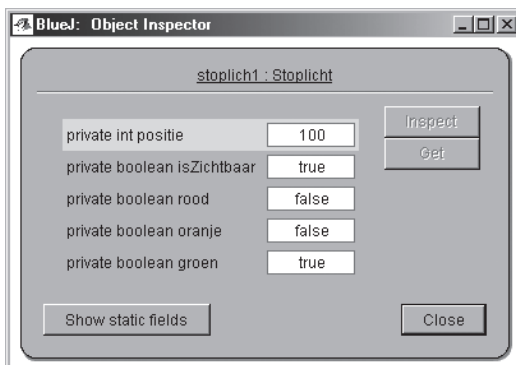
**Figuur 1.8**

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.



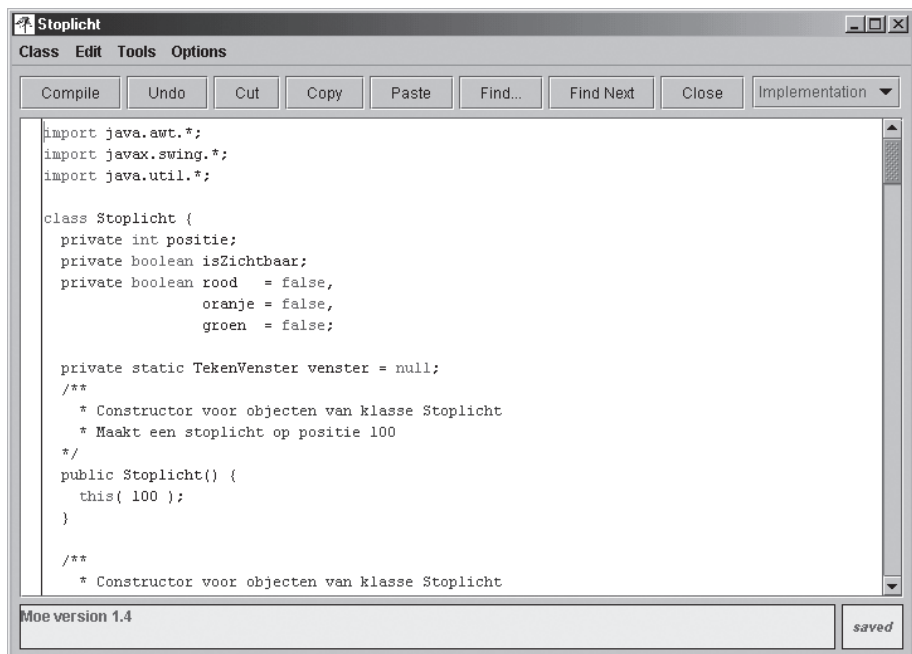
**Figuur 1.9**

## 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.

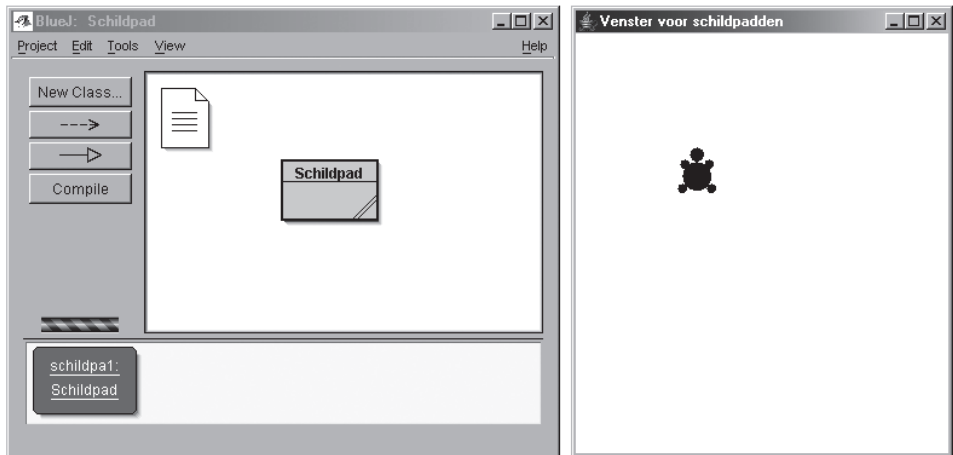
## 1.6 Een andere klasse

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.

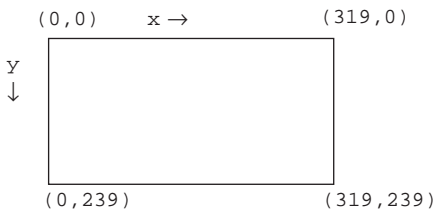
### 1.6.1 Methoden met een retourwaarde

De schildpad heeft drie methoden met een retourwaarde:

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

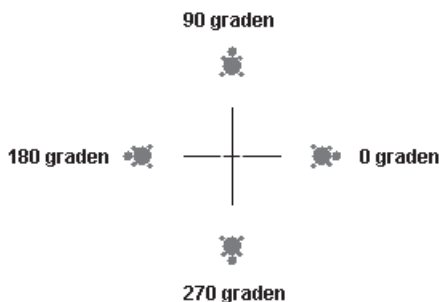
Deze methoden leveren een `int` (een geheel getal) als resultaat af. Respectievelijk de x-coördinaat van de vensterpositie van de schildpad, de y-coördinaat van de positie en de richting waarin de schildpad kijkt.

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.

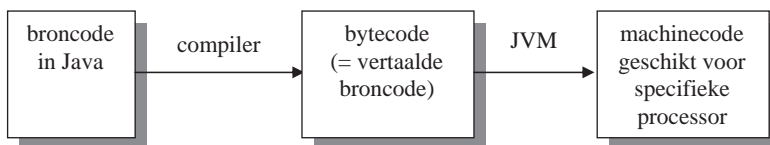
Omdat bytecode niet zo verschrikkelijk veel van alle soorten machinecodes verschilt, is het vertalen van bytecode niet zo'n grote klus en kan de JVM een tamelijk klein programma zijn. De verspreiding van die gratis JVM's is geen probleem: daar is het internet goed voor.

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](http://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 knippen.  
Aanwijzing: Kopieer de methode `slaap()` uit de klasse `Schildpad` en gebruik deze voor het knippen.
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

- Symbolen
  - @
    - voor javadoc 116
  - BorderLayout.SOUTH 277
  - interface
    - en polymorfisme 318
  - Pac-Man 363
- A
- aanroepen
  - constructor 3
  - van methode 4
- aantal cijfers
  - achter decimale punt 181
- abs() 158
- abstracte klasse 327
  - verschil met interface 331
- abstracte methode 321, 327
- AbstractOnderdeel 327
- AbstractPlaatje 356
- accessor 29, 60
  - naamgeving 31
  - naamgeving voor boolean-attribuut 140
- achtergrondkleur 270
- action 258
- ActionListener 258, 326
- actor 97
- actueel argument 35
- addActionListener() 258
- afbeelding
  - op knop of label 349
  - van de schijf opvragen 347
- afgeleide klasse 223
- algoritme 123, 133
  - bubble sort 210
  - driehoekige lay-out 208
  - kwadratisch 215
  - rechthoek tekenen 189
  - som van een rij getallen 133
  - sorteren van drie waarden 189
  - verwisselen van twee waarden 188
  - zoeken in lijst 196
- algoritmiek vi
- alles apart 205
- alles tegelijk 205
- ambigüiteit 69
- analyse v, 51
- animated GIF 348
- animatie 335, 341
- anoniem object 101, 259, 304
- append() 177
- applet 210, 251
- appletviewer 210, 252
- applicatie 254
- argument 11
  - actueel 35
  - formeel 35
  - namen van 60
- array 199
  - for-each 203
  - index 200
  - kopiëren 206
  - lengte 202
  - met doubles 203
  - van strings 199
- ArrayList 77
- assignment statement 33
- associatie 41, 54
- attribuut 1, 7, 338
- Auto 318
- awt 249
- B
- baksteen 341
- Bal 344
- Balpenen 167
- Bank 40, 77, 86, 224, 233, 294
- Bankrekening 22, 28, 40, 86, 224, 294
- bereik
  - van int 123
- bericht sturen 4
- BestandPlaatje 357
- besturingssysteem 16
- binair zoeken 221
- block tag 116
- blok 282
- BlueJ v
  - downloaden 2
  - editor 10
  - objecten maken 2
  - Object Inspector 7
  - Terminal Window 38
  - zelfgemaakte klassen testen 21
- body
  - for-statement 170, 172
  - van if-statement 84, 143
  - van methode 30
  - van while-statement 141, 178
- boolean 123, 135, 139, 193
- Boole, George 135
- BorderCSPaneel 277
- BorderLayout 275
- BorderLayout.CENTER 277
- BorderLayout.EAST 277
- BorderLayout.NORTH 277
- BorderLayoutPaneel 276
- BorderLayout.WEST 277
- botsen
  - van figuren 354
- bounding box 299, 354
- boxing 199
- break statement 149
- broncode 9, 35
  - opmaak 35
- browser 251, 252
- bubble sort
  - broncode 213
  - efficiëntie 214
- BubbleSort 210
- buffer 96, 105
- byte 123, 193
- bytecode 17

## C

case 148  
 case sensitive 31  
 cast 126  
 ceil() 158  
 char 123, 128, 176  
 Character 193  
 Cijfer 147  
 cijferinvoer 218  
 cijfers  
   achter de komma 181  
 cirkel 299, 318, 357  
   gevuld 300  
 cliëntklasse 243  
 coherent 21  
 collectie 77  
 commentaar 27  
   voor javadoc 116  
 compiler 16  
 compileren *zie* vertalen  
 component 250  
 compound assignment operator 130  
 concatenatie 39  
 concatenatie met += 147  
 concrete klasse 327  
 conditie 141, 170  
   if-statement 84  
   while-statement 178  
 constante 152  
 constructor 32  
   aanroepen 3  
   aanroepen met this() 69  
   default 65  
   en overerving 238  
   enum 155  
   overloading 67  
 constructor overloading 34, 67  
 container 250  
 content pane 253  
 controlegedeelte  
   van for-statement 170  
 controlevariabele  
   naam van 174

## converteren

String naar double 283  
 String naar int 279  
 van double naar int 127  
 van int naar double 126  
 cos() 158  
 creates 86

## D

database 77  
 Datum 57, 285  
   implementatie 59  
 declareren  
   van array 204  
 decrement-operator 131  
 deelbaarheid 140  
 default-constructor 64  
 deprecated 56  
 diepe kopie 206  
 dobbelsteen 156, 164  
 documentatie 115  
 domeinklasse 53, 100  
 domeinclassen  
   en gebruikersinterface 285  
 double 11, 123, 193  
 Double.MAX\_VALUE 124  
 Double.MIN\_VALUE 124  
 Double.parseDouble() 284  
 Draw 309  
 drawArc() 302  
 drawOval() 299  
 drawPolygon(). 316  
 drawRect() 299  
 drawRoundRect() 302  
 drawString() 297  
 DriehoekApplet 295  
 dubbele methode-aanroep  
   290  
 dynamische binding 242  
 dynamische type 226

## E

editor  
   van BlueJ 10  
 eerste twee samen 205

## ellips 299

  gevuld 300  
 en-operator 136  
 enum  
   constructor 155  
 enumeratie 153  
 equals() 161  
 equalsIgnoreCase() 161  
 Error\  
   incompatible types 13  
 error message 27  
 event 256  
 event-afhandeling 256  
 exceptie  
   ArrayIndexOutOfBoundsException-  
     Exception 202  
 exp() 158  
 expressie 127  
 extends 231  
 externe methodeaanroep 45

## F

false 123, 139  
 Fibonacci 220  
 field 7  
 FIFO 96  
 fillArc() 302  
 fillOval() 300  
 fillPolygon(). 316  
 fillRect() 300  
 fillRoundRect() 302  
 final 152  
 first in first out 96  
 fixture 94  
 flag 140  
 FlexRechthoek 240, 241  
 float 123, 193  
 floating point 123  
 floor() 159  
 Font 338  
 for-each 85  
   en array 203  
 format specifier 182  
 format string 182  
 formatteren  
   van getallen in uitvoer 181

- formeel argument 35
- for-statement 112, 169
  - andere beginwaarde 174
  - body 170
  - grotere stappen dan 1 175
  - met een char 176
  - naam van controlevariabele 174
  - terugtellen 174
  - variabele begin- of eindwaarde 175
  - waarvan body niet wordt uitgevoerd 175
- foutmelding 13, 27
- fully qualified name 337, 362
- functionaliteit 13
  
- G
- gebruikersinterface 249
  - en domeinklassen 285
- generalisatie 223, 230, 231
- genest for-statement 209
- get() 86
- getSource() 268
- getter 29, 31, 60
- GIF 347
- grafische context 296
  - doorgeven aan andere methode 305
  - naam van 297
- GridLayout 278
- GridLayoutPaneel 278
- GUI 249
  
- H
- hash-code 115, 246
- header 30 *zie* kop
- herdefinitie 241
- herdefinitie versus overlading 241
- herhalingsopdracht 112, 169
  - for-statement 169
  - while-statement 178
- hoofdlettergevoelig 31
- hoofdletters en kleine letters 31
  
- HTML
  - op knop en label 271
  
- I
- if-else-statement 144
- if-statement 84, 141
- ImageIcon 347
- image observer 348
- immutable 177
- implementatie 24, 62
- implementatieklasse 104
- implementeren 321
- import
  - static 158
- import-statement 80
- increment 131
- index 82, 178, 200
  - grenzen 202
- inheritance 223, 232
- initialisatie
  - in for-statement 171
  - van attributen 65
  - van variabele voor while-statement 180
- initialiseren
  - van array 204
- inner class 257
- instanceof 225, 244, 320
- instantie
  - van een klasse 1
- int 11, 123
  - bereik 123
- Integer 193
- Integer.MAX\_VALUE 123
- Integer.MIN\_VALUE 123
- Integer.parseInt() 86, 281
- interface 321
  - meer dan één interface implementeren 330
  - van object 4
  - verschil met abstracte klasse 331
- Internationalization 182
- interne methodeaanroep 48
- invoer
  - van getallen 279
  
- inwendige klasse 257
- is een relatie 235
- iteratie 169
  
- J
- Java
  - geschiedenis 15
- java.awt.event 258
- java.awt.List 362
- javadoc 115
- Java EE 17
- Java ME 17
- Java Runtime Environment 17
- Java SE 17
- Java SE Development Kit 1, 17
- java.util.List 362
- Java Virtual Machine 17
- JButton 250
- JDK 17
  - broncode van 326
- JFrame 255
- JLabel 250
- JPanel 250
  - coördinaten 274
- JPG 347
- JRE 17
- JTextField 264
  - event 270
  - uitschakelen 284
- JVM 17
  
- K
- Kassa 133
- kassabon 195
- keuze 144
- keuze-opdracht 141
- Kind 56, 290
- klasse 1
  - abstracte 327
  - concrete 327
  - standaardingrediënten 71
  - zelf maken 21
- klasseconstante 153
- klassendiagram 3, 53, 86
  - overerving 230

- kleur 298
  - en componenten 270
  - zelf mengen 304
- KleurPaneel 271
- knop 250
  - met afbeelding 349
- kofferslot 218
- komma
  - in format specifier 184
- kop
  - van methode 14, 29
- kopiëren
  - van array 206
- kwadratisch algoritme 215
  
- L
- laatste twee samen 205
- label 250
  - met afbeelding 349
- lay-out
  - uitzetten 273
  - van broncode 35
  - van componenten 272
- lay-outmanager 250
  - uitschakelen 272
- ledenlijst 218
- Leeftijdenlijst 201
- lege body 172
- lege string 84, 268
- length 202
- lettertype 338
- library 80
- lineair zoeken 196
- links uitlijnen 183
- List 362
- literal 123, 128
- locale 185
- log() 159
- logische
  - operator 136
- logische fontnaam 338
- lokale variabele 46, 282
  - initialisatie 46
  - met dezelfde naam 282
- long 123, 193
- loop 169
  
- Lucifers 221
- lus 169
  
- M
- Maal10Paneel 283
- machinecode 15
- main() 255
- mainframe 15
- marker interface 326
- Math 156
- Math.random() 156
- max() 159
- methode 4
  - aanroepen 4
  - abstracte 321, 327
  - body 30
  - die iets aflevert 7
  - externe aanroep 45
  - interne aanroep 48
  - kop 14, 29
  - statische 156
  - van superklasse aanroepen 232, 241
  - zelf maken 13
- methode-aanroep
  - dubbele 290
- method header *zie* kop
- microcomputer 15
- min() 159
- minteken
  - in format specifier 183
  - in UML 57
- Mobiel 110
- modifier 235
- MonoSpaced 338
- multipliciteit 54, 55, 100
- mutator 29, 60
  - naamgeving 31
  
- N
- naamloos object 101
- naamloze instantie 259
- NaamPaneel 291
- namen
  - van argumenten 60
- namen in Java 30
  
- navigeerbaarheid 102
- niet-operator 138
- no-argument constructor 65
- nul
  - in format specifier 184
- null 65, 82
- NullPointerException 282
- NumberFormatException 281
  
- O
- object 1, 246
  - anoniem 101
  - interface van 4
  - maken in BlueJ 2
  - naamloos 101
  - toestand van 6
- Object Bench *zie* objectenbank
- objecten
  - bewaren in BlueJ 93
- objectenbank 4, 95
- objectendiagram 43, 101
- objectgeoriënteerde begrippen v
- Object Inspector
  - van BlueJ 7
- object-variabele 70
- of-operator 138
- Onderdeel 321
- onderhoudbaarheid 244
- onderstreep tekens 30
- ondiepe kopie 206
- oneindige herhaling 186
- ontwerp v, 56
- ontwikkelomgeving 249
- operand 137
- operator 124
  - logische 136
  - relationele 134
- operator -- 131
- operator ! 138
- operator != 135
- operator \* 124
- operator / 124
- operator && 136
- operator % 124, 140

- operator + 124
- operator ++ 131
- operator += 29, 130, 147
- operator < 135
- operator <= 135
- operator == 135, 160
- operator > 135
- operator >= 135
- operator || 138
- opmaak
  - van broncode 35
- OptelPaneel 280
- overerving 223, 232
  - klassendiagram 230
- overloading
  - constructor 34, 67
  - methode 87
  - versus overriding 241
- overriding 240
  - versus overloading 241
- overschrijven 241
  
- P
- package 80
- paintComponent() 295
- paneel 253
- parameter 11 *zie* argument
- PC 105
- PC-printersysteem 96
- pijl
  - om relatie aan te geven 40
- Plaatje 351
- platform 16
- plusteken
  - in UML 57
- Point 354
- Polygon 315
- PolygonApplicatie 316
- polymorfie 246
- polymorfisme
  - en interface 318
- postconditie 98
- postfix 132
- pow() 159
- preconditie 98, 287
- prefix 132
  
- primitief type 123
- Printer 105
- printRechthoek() 191
- prioriteit 128
- private 57, 234
  - methode 89
- procentteken
  - afdrukken in format string 185
  - in format specifier 182
- processor 15
- programma 15
- proportioneel lettertype 338
- protected 234
- Provider 110
- public 28
- punt-operator 45, 48
  
- Q
- queue 96
  
- R
- random 156
- random() 159
- rechthoek 236, 241, 299, 318
  - gevuld 300
- rechts uitlijnen 183
- Rectangle 353
- reference *zie* referentie
- referentie 42
  - van een interface-type 321
- regels wit 37
- rekenkundige operator 124
- rekenmachine 293
- relatie 54
  - pijl 41
  - tussen klassen 40
- relationele operator 134
- Rente 179
- repaint() 313
- responsability *zie* verant-  
woordelijkheid
- retourwaarde 7, 12
- return 29
- return value *zie* retourwaarde
- reusable software 223
  
- RGB 304
- rint() 159
- Romeinse cijfers 221
- round() 159
- runtime 282
  
- S
- samengestelde toekennings-  
operator += 130
- SansSerif 338
- Schildpad 10
- schoolklas 51, 52, 91
- schreef 338
- scientific notation 123
- scope 47, 282
- selection sort 221
- Serif 338
- setBackground() 270, 301
- setBounds() 273
- setCharAt(). 178
- setColor 299
- setEditable() 284
- setEnabled() 284
- setForeground(). 270
- setLayout() 275
- setLayout( null ) 273
- setter 31, 60
- setText() 259, 265
- setVisible() 285
- short 123, 193
- signatuur 69
- sin() 159
- slashes 27
- sms 108
- SMS 110
- sorteren
  - door selectie 221
  - met Arrays.sort() 215
  - met bubble sort 212
- source code *zie* broncode
- Spaarrekening 224
  - broncode 227
- SPARC 16
- specialisatie 223, 231
- sqrt() 159
- src.zip 326

state *zie* toestand  
 statement 30  
 static 153  
   import 158  
 static final 153  
 statische constante 153  
 statische methode 156  
 statische type 226  
 Steen 341  
 Stempel 190, 208, 219  
 stereotype 86  
 Stoplicht 2  
 string  
   concatenatie 39  
 String 13  
   vergelijken van twee strings 159  
 StringBuffer 176  
 strut 289  
 stuiter() 344  
 subklasse 223  
 sublijst 362  
 super 232, 241  
 super() 239  
 superklasse 223, 230, 321  
 Swing 249  
 Swing applicatie 254  
 switch-expressie 148, 151  
 switch statement 147  
 syntax 27  
 system boundary 98  
 System.out.print() 39  
 System.out.println() 39

T

taalelementen  
   van Java v

Tafel 173  
 Tafels oefenen 293  
 tafel van 13 172  
 tagging interface 326  
 tan() 159  
 tekstvak 263  
   schoonmaken 268  
   string omzetten naar int 279  
   uitschakelen 284

TekstvakEnKnopPaneel 265  
 TekstvakPaneel 264  
 TellerApplet 335  
 temporary 189  
 Terminal Window  
   in BlueJ 38  
 terugkeerwaarde 7  
 TestBoolean 125, 135, 145  
 Test Fixture 95  
 TestIfElse 145  
 testklasse 94, 95  
 TestSwitch 149  
 this 48  
   om verwarring te voorkomen 61  
 this() 68  
 timer 335  
   starten en stoppen 339  
 toekenningsopdracht 33  
 toestand  
   van object 1, 6  
 toevalsgetallen 157  
 toString() 63, 114, 184  
   van enum 155, 156  
 translereen 317  
 true 123, 139  
 TweeKnoppenPaneel 267  
 type  
   dynamische 226  
   statische 226  
   van attributen en methoden 23  
 typecast 126, 157, 226

U

UI 249  
 uitbreidbaarheid 245  
 uitschakelen  
   lay-outmanager 272  
 UML vi  
   klasse 56  
   minteken 57  
   multipliciteit 55  
   notatieverschillen met Java 58  
   objectendiagram 101  
   plusteken 57

stereotype 86  
   usecasediagram 97  
 unboxing 194, 199  
 underscore 30  
 Unified Modeling Language  
   *zie* UML  
 unit testing tools 93  
 use case 97  
   beschrijving 98  
   diagram 97  
   tekst printen 105

V

validate() 285  
 variabele  
   lokale 46  
   object- 70  
 veld 7  
 venstercoördinaten 12  
 verantwoordelijkheid 22, 40  
 vertalen 10  
 vertaler 16  
 virtuele machine 17  
 vlag 140  
 volledige naam 362  
 voorbeelden  
   downloaden 2  
 voorgrondkleur 270

W

waarheidswaarden 135  
 wachtrij 96  
 webpagina 251  
 WelkomstPaneel 260  
 wetenschappelijke notatie 123  
 while-statement 178, 179  
 Willemijn 178  
 wrapper class 193

Z

zelfstandig naamwoord 51  
 zoeken 196  
   binair 221  
   lineair 196  
 ZonderLayoutPaneel 272



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.academicsservice.nl](http://www.academicsservice.nl) en [www.gertjanlaan.com](http://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.academicsservice.nl](http://www.academicsservice.nl)