Aan de slag met Java

Objectgeoriënteerd ontwerpen en programmeren

GERTJAN LAAN

ACADEMIC SERVICE

Vierde druk

Aan de slag met Java

Vierde druk

Gertjan Laan

Meer informatie over deze en andere uitgaven kunt u verkrijgen bij:
BIM Media B.V.
Postbus 16262
2500 BG Den Haag
Tel.: (070) 304 67 77
www.bimmedia.nl

Gebruik onderstaande code om dit boek eenmalig toe te voegen aan je boekenplank op www.academicx.nl.
Let op: je kunt deze code maar een keer gebruiken.

© 2014 BIM Media B.V., Den Haag
Academic Service is een imprint van BIM Media B.V.

1e druk 1999
2e druk 2002
3e druk 2007
4e druk 2014
De eerste drie drukken zijn verschenen onder de titel *En dan is er... Java*.

Omslag: Carlito's Design, Amsterdam
Zetwerk: Redactiebureau Ron Heijer, Markelo

ISBN 978 90 395 2757 3
NUR 123 / 989

Alle rechten voorbehouden. Alle auteursrechten en databankrechten ten aanzien van deze uitgave worden uitdrukkelijk voorbehouden. Deze rechten berusten bij BIM Media B.V.
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.
Voorzover 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). 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). 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, BIM Media B.V. neither guarantees the accuracy of the data contained here in nor accepts responsibility for errors or omissions or their consequences.

Voorwoord

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.

Bij het leren van een objectgeoriënteerde taal is het belangrijk in een vroeg stadium kennis te maken met de kernbegrippen *object* en *klasse*.

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, 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 zorgen 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.

De begrippen klasse en object worden al direct in hoofdstuk 2 geïntroduceerd, en toegepast op event handling. Het vierde hoofdstuk geeft een demonstratie van de kracht van objectgeoriënteerd programmeren in de vorm van een eenvoudige klasse met de naam Kassa. Het zesde hoofdstuk gaat in op het ontwerpen van klassen op basis van verantwoordelijkheden, daarbij gebruikmakend van UML. In hoofdstuk 7 wordt aandacht besteed aan de analyse van een eenvoudig probleem, de vertaling daarvan in een model met gebruikmaking van UML, en tot slot de implementatie in Java. De onderwerpen overerving en polymorfie in hoofdstuk 8, en abstracte klassen en interfaces in hoofdstuk 10 completeren het objectgeoriënteerde beeld.

In deze vierde editie zijn voorbeelden en opgaven gestroomlijnd, fouten verbeterd, UML-diagrammen aangepast. Twee hoofdstukken (de hoofdstukken 8 en 14 uit de vorige editie) zijn online geplaatst, en kunnen als pdf worden gedownload.

Aanvullende materiaal, zoals online vragen met directe feedback, de broncode van de voorbeelden, de uitwerkingen van de opgaven, een handleiding JCreator en PowerPoint-presentaties bij de hoofdstukken, is te vinden op de hbo-portal van de uitgever www.academicx.nl en op mijn website www.gertjanlaan.nl.

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

Zaandam, januari 2014
Gertjan Laan

Inhoud

Voorwoord v

1 Introductie 1
1.1 Inleiding 1
1.2 Achtergrond 1
1.3 En dan is er Java 2
1.4 Applicatie en applet 3
1.5 Hoe gaat het programmeren in zijn werk? 3
1.6 Wat heb je nodig? 5
1.7 Verschillende versies van Java 6
1.8 Vragen 6

2 Klassen en objecten 7
2.1 Inleiding 7
2.2 Een opstartklasse voor een Swing-applicatie 7
2.3 Oefeningen 9
2.4 Wat is een klasse? 9
2.4.1 Waaruit bestaat de opstartklasse? 10
2.4.2 De methode main() 11
2.4.3 De kop van de methode 11
2.4.4 De body van de methode 11
2.4.5 De klasse Vb0201 12
2.4.6 Importeren van een package 12
2.4.7 Commentaar 13
2.5 Een JPanel 14
2.5.1 Een JPanel maken 14
2.5.2 Declaratie van referenties 15
2.5.3 Constructor 15
2.6 Samenwerking tussen objecten 16
2.7 Oefeningen 18
2.8 Event-afhandeling 19
2.8.1 Een event handler 19
2.8.2 Een naamloze instantie 21
2.8.3 Wat gebeurt er precies bij event-afhandeling? 21
2.9 Twee knoppen en een tekstvak 22
2.10 Regels voor de opmaak van broncode 23
2.10.1 Witregels 25
2.11 Regels voor namen in Java 25
2.11.1 Gebruik van hoofdletters en kleine letters 26
2.12 Samenvatting 26

2.13	Vragen		27
2.14	Oefeningen		27
3	Het type int		29
3.1	Inleiding		29
3.2	Variabelen van het type int		29
	3.2.1	Het assignment-statement	31
	3.2.2	De methode paintComponent()	31
	3.2.3	Van int naar String	32
	3.2.4	De coördinaten van JPanel	33
	3.2.5	Verkorte declaratie en initialisatie	34
3.3	De rekenkundige operatoren +, -, *, / en %		34
	3.3.1	De gehele deling en de rest	36
	3.3.2	Het bereik van het type int	37
	3.3.3	Prioriteiten	38
	3.3.4	Onderstrepingsteken in getallen	39
3.4	Oefeningen		39
3.5	Meer operatoren		40
	3.5.1	De toekenningsoperator +=	40
	3.5.2	Andere toekenningsoperatoren	41
	3.5.3	De increment-operator ++	41
	3.5.4	De decrement-operator --	41
	3.5.5	Postfix en prefix	42
3.6	Tekenen met Java		43
	3.6.1	Kleur	44
	3.6.2	Zelf kleuren maken	44
	3.6.3	Over de naam van de grafische context	45
	3.6.4	Rechthoeken en ellipsen	46
	3.6.5	Vierkant en cirkel	46
	3.6.6	Gevulde rechthoeken en ellipsen	46
3.7	Oefeningen		48
3.8	Rondingen en bogen		48
	3.8.1	De methoden drawRoundRect() en fillRoundRect()	48
	3.8.2	De methoden drawArc() en fillArc()	49
3.9	Invoer van gehele getallen via een tekstvak		50
	3.9.1	Lokale variabelen	52
	3.9.2	Twee lokale variabelen met dezelfde naam	52
	3.9.3	Attributen en lokale variabelen	53
	3.9.4	Initialisatie van lokale variabelen en attributen	53
	3.9.5	NullPointerException	53
3.10	Oefeningen		54
3.11	De groeiende cirkel		54
	3.11.1	De methode repaint()	55
3.12	Samenvatting		56
3.13	Vragen		56
3.14	Oefeningen		57

4 Objectgeoriënteerd programmeren 59
4.1 Inleiding 59
4.2 Het type double 59
4.2.1 Rekenkundige operatoren 60
4.2.2 Delen: de gehele en de normale deling 60
4.2.3 De operator % 61
4.2.4 Omzetten van int naar double: typecasting 61
4.2.5 Omzetten van double naar int 62
4.2.6 Invoer van double via een tekstvak 62
4.2.7 Constante: final 64
4.3 JLabel 64
4.3.1 Het maken van een label 64
4.4 JTextField-event 65
4.5 Een betere lay-out 66
4.5.1 Lay-outmanager uitschakelen 67
4.5.2 Mogelijkheden van een tekstvak 69
4.6 Formatteren van uitvoer met String.format() 70
4.6.1 Scheidingstekens in getallen 72
4.7 Oefeningen 73
4.8 Algoritme: de kassa 73
4.9 Een kassa is een object 74
4.9.1 De klasse Kassa 74
4.9.2 Data hiding 74
4.9.3 Een methode voor de kassa 75
4.9.4 Opvragen van de waarde van een attribuut 76
4.9.5 De klasse Kassa 76
4.10 Objecten maken 77
4.11 Gebruikersinterface voor de kassa 78
4.12 GridLayout 81
4.12.1 Een rand om het geheel: Border 81
4.13 De kracht van objectgeoriënteerd programmeren 82
4.14 Samenvatting 83
4.15 Vragen 84
4.16 Oefeningen 84

5 Keuzes en herhalingen 87
5.1 Inleiding 87
5.2 Relationele en logische operatoren 87
5.2.1 Relationele operatoren 87
5.2.2 Logische operatoren: de en-operator && 88
5.2.3 Een consoleapplicatie 89
5.2.4 Operanden 90
5.2.5 De of-operator || 91
5.2.6 De niet-operator ! 91
5.2.7 Een boolean-variabele 92

5.3 Het if-statement 92
5.3.1 Een ingewikkelder voorbeeld 95
5.3.2 Nogmaals twee knoppen en een tekstvak 96
5.4 Het if-else-statement 97
5.4.1 Deelbaarheid onderzoeken 99
5.5 Een sorteeralgoritme 100
5.5.1 Algoritme voor het verwisselen van twee waarden 100
5.5.2 Algoritme voor het sorteren van drie waarden 100
5.6 Oefeningen 101
5.7 Wanneer zijn twee strings gelijk? 101
5.7.1 De methoden equals() en equalsIgnoreCase() 102
5.8 Het for-statement 103
5.8.1 Controlegedeelte van het for-statement 104
5.8.2 De body van het for-statement 105
5.8.3 Zet geen puntkomma na het controlegedeelte 105
5.8.4 De tafel van 13 106
5.8.5 Een nette tabel 107
5.8.6 Een Font-object 108
5.9 Andere mogelijkheden met een for-statement 109
5.9.1 Andere beginwaarden dan 0 of 1 110
5.9.2 Terugtellen 110
5.9.3 Grotere stappen 110
5.9.4 Variabele begin- en eindwaarden 110
5.9.5 Een for-statement waarvan de body niet wordt uitgevoerd 111
5.10 Een rijtje cirkels 111
5.11 Oefeningen 113
5.12 Het while-statement 113
5.13 Opmerkingen bij het while-statement 116
5.13.1 Oneindige herhaling 117
5.13.2 Gelijkwaardigheid van het for-statement en het while-statement 117
5.14 Het do-while-statement 118
5.15 Samenvatting 118
5.16 Vragen 119
5.17 Oefeningen 120

6 Het ontwerpen van een klasse 123
6.1 Inleiding 123
6.2 Ontwerp van Datum 123
6.2.1 Een klasse in UML 124
6.2.2 Getters voor Datum 124
6.2.3 Setters voor Datum 125
6.2.4 Notatieverschillen Java en UML 125
6.2.5 Implementatie van Datum 126
6.2.6 Over this 127
6.2.7 Een instantie van Datum maken 128
6.3 Oefeningen 129

Hoofdstuk 1
Introductie

1.1 Inleiding

Java is een bijzondere programmeertaal. Voor 1995 had bijna niemand ervan gehoord, en een paar jaar later is de naam Java zelfs bekend bij mensen die nog nooit een programma geschreven hebben. De populariteit van internet, de gratis verspreiding van Java en de enorme publiciteitsgolf over de mogelijkheden van Java hebben allemaal aan de bekendwording bijgedragen.

In dit hoofdstuk geef ik een korte beschrijving van de plaats van Java te midden van andere programmeertalen. Begrippen die daarbij horen, zoals processor, besturingssysteem, platform, compiler, bytecode, virtuele machine, machinecode, applet en applicatie, worden uitgelegd.

1.2 Achtergrond

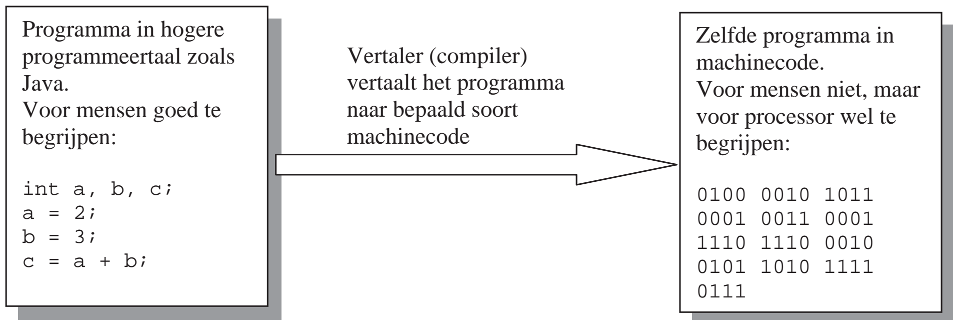
Java is in 1990 bij de firma Sun Microsystems ontworpen in de verwachting dat de taal gebruikt zou worden voor de programmering van allerlei elektronische apparatuur, zoals kopieermachines, afstandsbedieningen, televisies en talloze andere producten. Zulke apparatuur is voorzien van elektronische schakelingen 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 tegelijk alle zenders van de kabel laten opzoeken en onthouden, als ik op twee bepaalde knopjes van de afstandsbediening druk.

Instructies zijn meestal bij elkaar opgeslagen in een geheugenchip. Zo'n bij elkaar horende reeks instructies heet 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: minicomputers, of nog groter: zogenaamde *mainframes*. Of apparaten die kleiner zijn, zoals tablets en smartphones. Het verschil tussen al deze apparaten zit vooral in de belangrijkste chip, de *(micro)processor*. Bijvoorbeeld:

- een pc met een microprocessor van het merk Intel
- een iPhone met een Apple A6-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 zogeheten *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, zie figuur 1.1.



Figuur 1.1

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 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. Een programma dat specifiek vertaald is voor Windows zal niet 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.

1.3 En dan is er 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:

1. Laat elk Java-programma door een compiler vertalen naar een soort standaard *tussentaal* die betrekkelijk dicht tegen machinecode aan zit.
2. 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 JVM's is geen probleem: daar is internet goed voor. De JVM wordt geleverd als plug-in voor een webbrowser, en ook als zelfstandig programma. De JVM is onderdeel van een pakket dat bekend staat onder de naam Java Runtime Environment of JRE, en dat iedereen gratis kan downloaden van www.java.com.

Dat betekent dus dat elke computer over de Java Virtual Machine kan beschikken en daarmee is die computer in staat om Java-bytecode te begrijpen.

Het probleem van de programmeur die voor elk platform een apart programma moet creëren, is daarmee opgelost. Wel moet voor elk platform een Java Virtuele Machine gemaakt worden en deze moet gemakkelijk te krijgen zijn. De firma Sun zorgde ervoor dat dit binnen twee jaar na de introductie van Java in 1995 voor de meeste platformen het geval was. In 2010 is Sun Microsystems, en daarmee de ondersteuning van Java, overgenomen door Oracle.

1.4 Applicatie en applet

Er bestaan twee soorten Java-programma's: applicaties en applets. Een *applet* is een Java-programma dat deel uitmaakt van een webpagina. Om een applet te kunnen uitvoeren moet je webbrowser beschikken over de Java Virtual Machine als plug-in. Hoe je een applet kunt maken staat in een bijlage van dit boek.

Een *applicatie* is een Java-programma dat zelfstandig door de JVM uitgevoerd wordt. Dit boek gaat in de eerste plaats over het maken van applicaties.

1.5 Hoe gaat het programmeren in zijn werk?

Voor wie nog niet eerder geprogrammeerd heeft, is het volgende goed om te weten: Je moet eerst de tekst van een Java-programma (de *broncode*) intikken in een *editor*.

Een editor voor Java is een tekstverwerker zoals Word, maar dan veel eenvoudiger. De tekst die je intikt moet je bewaren als bestand op de schijf. De naam van zo'n bestand krijgt `.java` als achtervoegsel.

De broncode die je intikt moet eerst worden vertaald voor hij kan worden uitgevoerd.

Dat vertalen gebeurt met behulp van een *compiler*. De compiler heeft twee belangrijke taken:

1. de broncode die je hebt ingetikt controleren op fouten, en fouten melden;
2. als er geen fouten gevonden zijn de broncode vertalen naar bytecode.

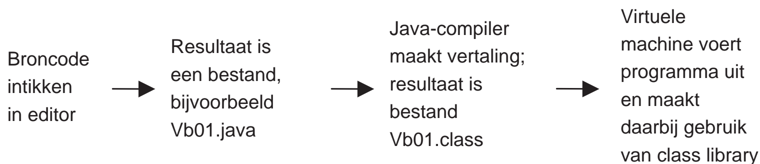
Laten we even aannemen dat de broncode taalkundig correct is, zodat er een vertaling tot stand komt. De vertaling wordt op de schijf gezet, in een bestand waarvan de naam het achtervoegsel `.class` heeft. De rol van de compiler is nu uitgespeeld.

Vervolgens moet het programma worden uitgevoerd door de virtuele machine.

Java wordt geleverd met honderden voorgedefinieerde klassen, klaar voor gebruik. Deze klassen zitten opgeborgen in bestanden: de zogenaamde *class libraries*. Het woord *library* betekent letterlijk bibliotheek, maar het gaat hier niet om boeken. De overeenkomst met een echte bibliotheek is dat er een voorraad klassen is (in plaats van boeken), waar elk Java-programma naar believen kopieën van kan maken.

Een belangrijke taak van de virtuele machine is om een kopie van de klassen die de vertaalde broncode nodig heeft, uit de bibliotheek te halen en in het geheugen van de computer te zetten, zodat de applicatie ze kan gebruiken.

In figuur 1.2 kun je een samenvatting zien van al deze stappen.



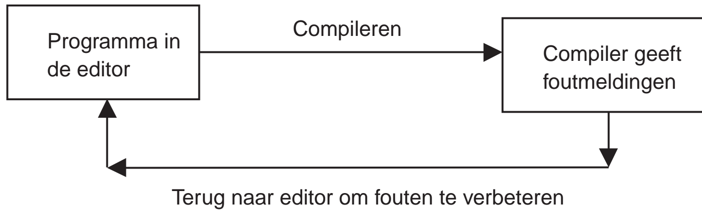
Figuur 1.2

Dit is de gang van zaken als de compiler geen fouten gevonden heeft. Wat gebeurt er als dat wel het geval is? Er verschijnen dan een of meer mededelingen over fouten, zogenaamde *foutmeldingen* (errors of error messages), en soms verschijnen er ook *waarschuwingen* (warnings).

De meeste compilers zetten in elke foutmelding en waarschuwing een nummer dat verwijst naar het nummer van de regel in de broncode waarin de fout is geconstateerd. In die regels moet je dus gaan zoeken naar eventuele fouten. Het vinden van fouten en het verbeteren daarvan kan vooral in het begin verdraaid lastig zijn. Je zondigt immers tegen de regels van een taal die je nog niet goed kent.

Zodra je de fouten verbeterd hebt, geef je opnieuw opdracht om het programma te laten runnen (vertalen en uitvoeren), in de hoop dat het nu wel goed gaat. Als dat niet zo is, zit er niets anders op dan opnieuw verbeteringen aan te brengen. Figuur 1.3 geeft een schematische voorstelling van deze verbetercyclus.

Pas als alle fouten eruit zijn kan het programma in zijn geheel worden vertaald en uitgevoerd.



Figuur 1.3

1.6 Wat heb je nodig?

Uit de vorige paragraaf blijkt dat als je wilt programmeren in Java, je in elk geval de beschikking moet hebben over:

1. Een *editor*, dat wil zeggen een simpele tekstverwerker waarmee je de Java-programma's kunt intikken en opslaan, bijvoorbeeld Notepad of Wordpad.
2. Een *compiler* die het ingetikte programma vertaalt naar *bytecode*.
3. Een *Java Virtual Machine* (JVM) die de bytecode uitvoert.

Een compiler en een JVM zijn als onderdeel van een gratis pakket verkrijgbaar bij Oracle: www.oracle.com. Het hele pakket is in de loop der tijden onder diverse namen bekend komen te staan, waarvan ik hier een paar noem: aanvankelijk *Java Development Kit* (JDK), daarna *Java 2 Software Development Kit* (Java 2 SDK), nog later *Java 2 Platform Standard Edition* (J2SE) en weer later eenvoudigweg *Java SE*. Tegenwoordig worden de namen JDK en Java SE veel gebruikt.

De JDK bevat geen editor. Voor het intikken van de programma's kun je in principe elke tekstverwerker gebruiken, mits je de tekst bewaart als een bestand van het type *text* of *txt*. Bovendien moet je het bestand de extensie `.java` geven. Het werken met de JDK is daarom een beetje omslachtig.

Om die reden zijn editor, compiler en JVM vaak samengevoegd tot een zogenaamde *geïntegreerde ontwikkelomgeving*, in het Engels Integrated Development Environment oftewel een *IDE*.
Een uitgebreide IDE is Netbeans die je samen met de JDK van www.oracle.com kunt downloaden. Een andere uitgebreide IDE is Eclipse (www.eclipse.org).

Een heel handige IDE is JCreator, gemaakt door de Nederlandse programmeur Wendel de Witte. JCreator v4.50 LE (Light Edition) is gratis en werkt goed onder moderne versies van Windows. Deze is te downloaden van www.jcreator.org/download.htm.

Voordat je de oefeningen van hoofdstuk 2 probeert te maken, is het belangrijk eerst wat vaardigheid op te doen in het werken met tenminste een van de ontwikkelomgevingen. Raadpleeg daartoe de handleiding bij de betreffende ontwikkelomgeving. Op de website bij dit boek staat een beknopte handleiding voor JCreator.

1.7 Verschillende versies van Java

Sinds 1995 zijn er verschillende versies van de JDK verschenen, met een nogal merkwaardige nummering:

1995, versie 1.0
1997, versie 1.1
1998, versie 1.2
2000, versie 1.3
2002, versie 1.4
2004, versie 5.0 of Tiger
2006, versie 6 of Mustang
2011, versie 7 of Dolphin

Versie 8 is aangekondigd voor 2014.
De versies vanaf 1.2 worden nogal verwarrend ook wel aangeduid met het "Java 2 Platform".

De versies zijn gelukkig *upward compatible*, wat betekent dat alles wat in versie 1.0 kon, nog steeds in versie 1.1 kan, en zo verder.

De voorbeelden in dit boek zijn getest met JDK 7.

1.8 Vragen

1. Wat is een platform?
2. Waarom moet een Java-programma vertaald worden?
3. Leg uit waarom het voor een programmeur lastig is een programma te schrijven dat op veel verschillende platformen draait.
4. Wat is het verschil tussen bytecode en machinecode?
5. Wat is de functie van een Java-compiler?
6. Wat doet de JVM?
7. Hoe wordt de JVM verspreid?
8. Wat is het verschil tussen een applicatie en een applet?
9. Wat is de JDK?
10. Wat is een IDE?

Hoofdstuk 2
Klassen en objecten

2.1 Inleiding

Het maken van een toepassing met Java is niet moeilijk. Zeker niet als je over een geïntegreerde ontwikkelomgeving (IDE) beschikt, waarin onder andere een tekstverwerker en compiler zijn ingebouwd. In paragraaf 1.6 staan de websites waar je geschikte ontwikkelomgevingen kunt vinden. Voor beginners is JCreator wellicht het eenvoudigst. In een beknopte handleiding die je kunt vinden op de website bij dit boek, staat beschreven hoe je een Java-toepassing kunt maken met behulp van deze IDE.

Omdat elke ontwikkelomgeving anders is, staan in dit en de volgende hoofdstukken geen aanwijzingen voor het werken met zo'n omgeving. Je vindt alleen de tekst die je moet intikken om een Java-programma te maken, de zogenoemde *broncode* (source code).

In een IDE wordt een groot gedeelte van het werk automatisch gedaan, zoals het vertalen van de broncode en het starten van de toepassing. Als programmeur kun je je daardoor concentreren op de hoofdzaken.

Om de gebruiker in staat te stellen met de applicatie te communiceren, moet je een *gebruikersinterface* maken. Voor het maken van een gebruikersinterface kent Java twee *libraries*: Abstract Window Toolkit (AWT) en Swing. Van deze twee is Swing de recentste en geavanceerdste en daarom zullen we deze hier gebruiken.

Een gebruikersinterface kan bestaan uit een venster, knoppen, invoervakken, schuifbalken, keuzelijsten en nog veel meer. In dit hoofdstuk leer je hoe je een applicatie maakt met een eenvoudige gebruikersinterface.

Hoewel het maken van een toepassing in Java niet moeilijk is, is dit hoofdstuk toch niet het meest eenvoudige. Er komen veel begrippen in voor die waarschijnlijk nieuw zijn en die onmogelijk meteen helemaal duidelijk kunnen zijn. Het gaat om belangrijke begrippen als klasse, object, constructor, methode en referentie. Het zijn niet alleen belangrijke, maar ook lastige begrippen. In dit en de volgende hoofdstukken zullen ze steeds terugkomen en gaandeweg duidelijker worden.

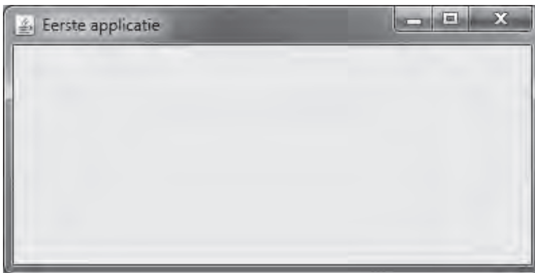
2.2 Een opstartklasse voor een Swing-applicatie

Om een applicatie te kunnen starten, moet je een *opstartklasse* maken. Zo'n opstartklasse ziet er vaak zo uit:

```
// Opstartklasse voor een applicatie
import javax.swing.*;

public class Vb0201 extends JFrame
{
  public static void main( String[] args )
  {
```

```
    // Maak een frame
    JFrame frame = new Vb0201();
    frame.setSize( 400, 200 );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setTitle( "Eerste applicatie" );
    frame.setVisible( true );
  }
}
```

In feite is deze opstartklasse een volledig Java-programma, weliswaar een programma dat nog maar heel weinig kan, maar wel met een zichtbaar resultaat. In figuur 2.1 zie je het resultaat (de uitvoer) van deze opstartklasse.



Figuur 2.1

De uitvoer bestaat uit een venster, een zogeheten `JFrame`, met knopjes in de rechterbovenhoek. Verder is het venster helemaal leeg, alleen in de titelbalk staat de tekst 'Eerste applicatie'.

Voor wie nog niet eerder geprogrammeerd heeft, is het volgende goed om te weten:

- Je moet de broncode zoals die hierboven staat eerst intikken in een editor. Denk er bij het intikken om dat Java verschil maakt tussen hoofdletters en kleine letters. Dat heet *case sensitive* of *hoofdlettergevoelig*. Tik dus `JFrame` in en `setSize`, en niet per ongeluk `Jframe` of `setsize`.
- Let bij het intikken goed op alle punten, puntkomma's, ronde en rechthoekige haakjes, accolades, aanhalingstekens en een spatie tussen de verschillende woorden. Deze leestekens zijn essentieel voor de compiler en het is bijna altijd fout als je er een weglaat of verkeerd neerzet.

Als je een Java-programma maakt en het laat uitvoeren, komen daar in principe de volgende handelingen bij kijken:

- Als je de broncode in zijn geheel hebt ingetikt, moet je deze bewaren onder de naam `Vb0201.java`. Let op: deze naam moet identiek zijn aan de naam van de opstartklasse, `Vb0201` in dit geval.
- Vervolgens moet je de broncode laten vertalen door een compiler. De compiler maakt een bestand dat uit bytecode bestaat en dat automatisch de naam `Vb0201.class` krijgt.
- Vervolgens geef je opdracht de applicatie te starten.
- De uitvoer van de applicatie (zoals in figuur 2.1) zie je dan op het beeldscherm.

Welke handelingen je precies moet verrichten om dit alles voor elkaar te krijgen, staat in de JCreator-handleiding die je op de website bij dit boek kunt vinden.

2.3 Oefeningen

Oefening 2.1
Download en installeer eventueel JCreator. Zie paragraaf 1.6.

Oefening 2.2
Maak in JCreator een nieuwe workspace. In de JCreator-handleiding, die je van de website bij dit boek kunt downloaden, staat hoe dat moet.
Maak vervolgens in deze workspace een nieuw project voor een Java-applicatie. Geef het project als naam `Vb0201`. Zie de JCreator-handleiding.

Oefening 2.3
Typ in JCreator de broncode van de opstartklasse uit paragraaf 2.2 en bewaar deze onder de naam `Vb0201.java`. In de JCreator-handleiding kun je lezen hoe dit gaat.

Oefening 2.4
Laat de broncode van `Vb0201` vertalen en uitvoeren. Zie de JCreator-handleiding. Als het goed is ziet de uitvoer eruit als in figuur 2.1.

Oefening 2.5
In de broncode van de opstartklasse wordt de afmeting van het venster van de applicatie bepaald door de opdracht `frame.setSize(400,200)`. Verander in de broncode de afmeting van het venster van in `600` bij `600`. Laat de code opnieuw vertalen en uitvoeren.

Oefening 2.6
Het venster van de applicatie komt automatisch in de linkerbovenhoek van het scherm. Met de opdracht `frame.setLocation(400,300)` kun je het venster een andere locatie geven, in dit geval 400 beeldpunten (pixels) vanaf de linkerkant en 300 beeldpunten vanaf de bovenkant van het scherm.
Voeg deze opdracht aan de opstartklasse toe en laat de code vertalen en uitvoeren.

Oefening 2.7
Laat in de opstartklasse de opdracht `frame.setSize()` weg. Wat is de afmeting van het venster als je nu de code laat vertalen en uitvoeren?

2.4 Wat is een klasse?

In paragraaf 2.2 staat de broncode van de klasse `Vb0201`. Maar wat is een klasse precies? Een mogelijk antwoord is: een klasse is een beschrijving voor objecten. Maar wat is dan een object? Deze vraag is niet zo eenvoudig te beantwoorden. Een object is bijvoorbeeld een ding dat gegevens kan bewaren. Een object kan zichtbaar zijn op het beeld-

scherm, zoals een venster of een knop. Een object kan ook een abstract ding zijn, zoals een girorekening.

In de rest van dit hoofdstuk en in de volgende hoofdstukken maak je kennis met veel verschillende objecten. Gaandeweg zul je steeds beter begrijpen wat objecten zijn, hoe je ze maakt en wat je er mee kunt doen. Alle Java-programma's werken met objecten: dat heet objectgeoriënteerd programmeren.

Voor dit moment is het belangrijkste: alle Java-programma's werken met objecten en om een object te kunnen maken, moet je eerst een klasse hebben. Zo'n klasse geeft een beschrijving van de objecten die je ervan kunt maken. Programmeren in Java bestaat dan ook voor een groot deel uit het maken van nieuwe klassen en het gebruiken van klassen die door anderen zijn gemaakt.

2.4.1 Waaruit bestaat de opstartklasse?

Laten we eens kijken waar de opstartklasse precies uit bestaat. De kern van de klasse bestaat uit de volgende regels:

```
JFrame frame = new Vb0201();
frame.setSize( 400, 200 );
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
frame.setTitle( "Eerste applicatie" );
frame.setVisible( true );
```

In elke regel staat een aparte opdracht die wordt afgesloten met een puntkomma.

De eerste regel is meteen een heel belangrijke:

```
JFrame frame = new Vb0201();
```

Met deze opdracht maak je een nieuw `JFrame`-object van de klasse `Vb0201`; een `JFrame`-object dat de naam `frame` krijgt. Altijd als je in Java het woord `new` ziet staan, wordt er een nieuw object gemaakt.

Een `JFrame`-object is in feite een leeg venster. Op het moment dat het gemaakt is, bestaat het venster alleen nog maar in het geheugen van je pc en is het nog niet zichtbaar op je scherm.

In de volgende opdracht vertel je wat de afmetingen van het venster moeten worden, in dit geval 400 beeldpunten breed en 200 hoog:

```
frame.setSize( 400, 200 );
```

Deze opdracht bestaat uit de naam van een object (`frame`), gevolgd door een punt (de *punt-operator*), gevolgd door de feitelijke opdracht die naar het object wordt gestuurd.

De volgende, nogal ingewikkeld uitziende opdracht is nodig om ervoor te zorgen dat je het venster op de normale manier kunt sluiten als je op het sluitknopje klikt:

```
frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

Met de volgende opdracht zet je een titel in de titelbalk van het venster, zie figuur 2.1:

```
frame.setTitle( "Eerste applicatie" );
```

Een tekst tussen aanhalingstekens, zoals `"Eerste applicatie"`, noemen we een *string*.

Met de laatste opdracht maak je het venster (het object) zichtbaar op het scherm:

```
frame.setVisible( true );
```

2.4.2 De methode main()

De opdrachten uit de vorige paragraaf staan in een gedeelte van de broncode dat wordt aangeduid met de naam `main()`:

```
public static void main( String[] args )
{
  // Maak een frame
  JFrame frame = new Vb0201();
  frame.setSize( 400, 200 );
  frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  frame.setTitle( "Eerste applicatie" );
  frame.setVisible( true );
}
```

Dit deel van het programma is een zogeheten *methode*. Een methode is een groepje opdrachten dat bij elkaar hoort en dat een naam heeft. De naam van deze methode is `main()`. Elke Java-applicatie *moet* een methode hebben met de naam `main()`. In de methode `main()` zet je altijd opdrachten die als eerste moeten gebeuren als je de applicatie start, zoals het maken van een venster voor de applicatie.

2.4.3 De kop van de methode

De methode `main()` begint, net als elke methode in Java, met een kop:

```
public static void main( String[] args )
```

In deze kop staat de naam van de methode, in dit geval `main()`, voorafgegaan door de woorden `public static void`. De betekenis van deze woorden zal ik in een later hoofdstuk verklaren. De kop van de methode heet in het Engels *method header* of kortweg *header*.

Achter de naam `main` staan een stel ronde haakjes met daartussen `String[] args`. De betekenis daarvan is nu niet belangrijk.

Ik zal de methode die *main* heet, aanduiden met `main()` in plaats van met de volledige kop `public static void main(String[] args)`.

2.4.4 De body van de methode

Het gedeelte vanaf de openingsaccolade tot en met de sluitaccolade van de methode `main()` heet de *body* van de methode:

```
{
  // Maak een frame
  JFrame frame = new Vb0201();
  frame.setSize( 400, 200 );
  frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
```

```
  frame.setTitle( "Eerste applicatie" );
  frame.setVisible( true );
}
```

Tussen de beide accolades kun je zoveel Java-opdrachten zetten als je wilt. Deze zullen dan van boven naar beneden worden uitgevoerd. Een opdracht in Java noemen we ook wel een *statement*. Erg belangrijk is dat elk statement *moet* eindigen met een puntkomma. Beginnende programmeurs vergeten nog wel eens een puntkomma en dan is het programma dus fout.

2.4.5 De klasse Vb0201

De methode `main()` is zelf weer een onderdeel van een *klasse* (*class*). In dit geval de klasse die `Vb0201` heet. Ook de klasse bestaat uit een kop en een body. Dit is de kop:

```
public class Vb0201 extends JFrame
```

De woorden `extends JFrame` betekenen dat de klasse `Vb0201` een uitbreiding is van de klasse `JFrame` (een venster). Dat betekent dat `Vb0201` zelf ook een `JFrame` is. De definitie van de klasse `JFrame` is vastgelegd in een zogenoemde *package* die `javax.swing` heet.

De namen van klassen die je zelf maakt mag je zelf verzinnen. In dit geval heb ik als naam `Vb0201` gekozen. Niet elke naam is geldig in Java. Voor correcte namen gelden regels, zie paragraaf 2.11. Die komen erop neer dat je een geldige naam maakt met letters, cijfers en het onderstreepteken. In elk geval mag je geen spatie in zo'n naam gebruiken.

Behalve een kop heeft de klasse ook een body. Hieronder zie je de body van de klasse, van openingsaccolade tot en met sluitaccolade:

```
{  // Begin van body van de klasse
  public static void main( String[] args )
  {
    JFrame frame = new Vb0201();
    frame.setSize( 400, 200 );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setTitle( "Eerste applicatie" );
    frame.setVisible( true );
  }
}  // Einde van body van de klasse
```

In dit voorbeeld staat in de body van de klasse alleen de methode `main()`, maar in minder eenvoudige applicaties kunnen er een heleboel verschillende methoden en nog andere dingen staan.

2.4.6 Importeren van een package

Vrijwel boven in het programma staat een regel die met het woord `import` begint.

```
import javax.swing.*;
```

Deze regel geeft aan dat je in dit programma voorzieningen wilt gebruiken uit een *package* die bij Java geleverd wordt: de package `javax.swing`. Een package bestaat doorgaans

uit een verzameling klassen die bij elkaar horen. Met het sterretje geef je aan dat je in principe alle klassen uit de package wilt importeren.

Door een package te importeren kun je van de klassen uit de package gebruikmaken in je eigen programma. Er zijn veel verschillende packages, maar voorlopig is het genoeg te onthouden dat *elke* Swing-applicatie gebruikmaakt van bovenstaande package, en dat dus in elke applicatie deze `import`-opdracht een van de eerste regels is.

2.4.7 Commentaar

De eerste regel van het programma bestaat uit zogeheten *commentaar*:

```
// Opstartklasse voor een applicatie
```

En ook de eerste regel van de methode `main()` bestaat uit commentaar:

```
// Maak een frame
```

Deze regels beginnen allebei met twee schuine strepen `//`, twee *slashes*. De compiler negeert alles wat op dezelfde regel na een dubbele slash staat. In de vertaalde versie, de bytecode, is dus niets meer van dit commentaar terug te vinden. Het commentaar is er alleen maar aan toegevoegd ter verduidelijking voor de menselijke lezer. Commentaar kan voor jezelf vaak erg nuttig zijn, vooral als je het programma een tijd geleden geschreven hebt. Ook wanneer iemand anders je programma leest, kan het erg verhelderend werken als er commentaar bij staat.

Commentaar hoeft niet per se aan het begin van een regel te staan. Commentaar kun je ook als volgt neerzetten:

```
public class Vb0201 extends JFrame                // Klasse
{
  public static void main( String[] args )        // Methode
  {
    ...
  }
}
```

Het programma verandert hiermee voor de compiler niet. Alles rechts van de dubbele slash wordt immers overgeslagen.

Als je veel regels met commentaar wilt maken, kun je dat als volgt doen:

```
// Zo maak je drie regels commentaar:
// Voorbeeld 0201
// Opstartklasse
```

Een andere manier om veel regels commentaar te maken is door slash sterretje aan het begin, en sterretje slash aan het einde van het commentaar neer te zetten, zoals in het volgende voorbeeld:

```
/* Zo kan het ook:
   Voorbeeld 0201
   Opstartklasse
*/
```

Op deze manier kun je een heleboel regels commentaar in je programma opnemen zonder dat je elke regel met een dubbele slash hoeft te beginnen.

Java kent nog een derde manier om commentaar te maken. Dit begint met een slash en twee sterretjes en eindigt met een sterretje en een slash.

```
/**
Dit is commentaar bedoeld voor documentatie die met het programma
javadoc gegenereerd kan worden
*/
```

Dit commentaar dient om documentatie bij je programma te genereren, zie paragraaf 7.7 voor meer uitleg.

2.5 Een JPanel

De opstartklasse zorgt alleen voor het maken van het `JFrame`-object, dat wil zeggen voor het maken van het lege venster met een titel in de titelbalk en voor de mogelijkheid het venster te kunnen sluiten met het sluitknopje. Om het venster ook inhoud te geven moet je er een `JPanel` aan toevoegen. Hoe dat gaat, lees je in de volgende paragrafen.

2.5.1 Een JPanel maken

Een `JPanel` is een zogeheten *container*. Het is een soort doos waarin je verschillende *componenten* kunt stoppen. Een component is bijvoorbeeld een knop, een menu, een tekstvak of een keuzelijst. Met behulp van zulke componenten maak je een gebruikersinterface. Het maken van de gebruikersinterface is een taak voor een aparte klasse die ik de naam `Paneel` geef.

In het volgende stuk broncode zie je hoe je een eenvoudige gebruikersinterface kunt maken met daarin een knop en een tekstvak.

```
// Een paneel met een knop en een tekstvak
import javax.swing.*;

public class Paneel extends JPanel
{
  private JButton knop;
  private JTextField tekstvak;

  public Paneel()
  {
    knop = new JButton( "Klik" );
    tekstvak = new JTextField( 10 );
    add( knop );
    add( tekstvak );
  }
}
```

Een knop is een component van de standaardklasse `JButton` en een tekstvak is een component van de klasse `JTextField`. De standaardklassen `JButton` en `JTextField` zijn allebei in de package `javax.swing` gedefinieerd.

De kop van de klasse ziet er zo uit:

```
public class Paneel extends JPanel
```

Dit betekent dat de klasse `Paneel` een uitbreiding is van de standaardklasse `JPanel`. Deze bevindt zich, net als de klasse `JFrame`, in de package `javax.swing`. De klasse `JPanel` is in feite alleen een lege container. De klasse `Paneel`, die een uitbreiding is van `JPanel`, ga ik vullen met een knop en een tekstvak.

Zowel een knop als een tekstvak zijn objecten. Verschillende soorten objecten. Om het knop-object en het tekstvak-object te kunnen gebruiken moet ik ze eerst een naam geven. Als namen kies ik `knop` en `tekstvak`.

2.5.2 Declaratie van referenties

In de body van de klasse `Paneel` vertel je als eerste aan de compiler welke namen je wilt gaan gebruiken:

```
private JButton knop;
private JTextField tekstvak;
```

De officiële benaming van elk van deze regels is: een *declaratie*. Met deze twee declaraties vertel je het volgende:

1. Ik wil een `JButton` gaan gebruiken en ik noem deze `knop`.
2. Ik wil een `JTextField` gaan gebruiken die ik de naam `tekstvak` geef.

Op de betekenis van het woord `private` kom ik later terug.

Een declaratie is alleen de bekendmaking van de namen. Het zijn in dit geval namen voor objecten. Een naam voor een object heet ook wel een *referentie*.

2.5.3 Constructor

Het volgende gedeelte in de body van de klasse `Paneel` ziet er zo uit:

```
public Paneel()
{
  knop = new JButton( "Klik" );
  tekstvak = new JTextField( 10 );
  add( knop );
  add( tekstvak );
}
```

Dit is een zogeheten *constructor*. Een constructor is een soort methode (een groepje bij elkaar horende opdrachten) die *altijd* dezelfde naam heeft als de klasse en die uitgevoerd wordt als je met de opdracht `new` aangeeft een nieuw object van een klasse te willen maken. Het is de constructor die het object maakt en vervolgens de eventuele opdrachten uitvoert die in zijn body staan.

Dus als je de opdracht `new Paneel()` geeft, gebeurt er het volgende: er wordt een nieuw `Paneel`-object gemaakt en de body van de constructor met de naam `Paneel()` met daarin de volgende opdrachten wordt uitgevoerd:

```
knop = new JButton( "Klik" );
tekstvak = new JTextField( 10 );
add( knop );
add( tekstvak );
```

In de eerste opdracht maakt de constructor een `JButton`-object, een knop waarop de tekst `"Klik"` komt te staan:

```
knop = new JButton( "Klik" );
```

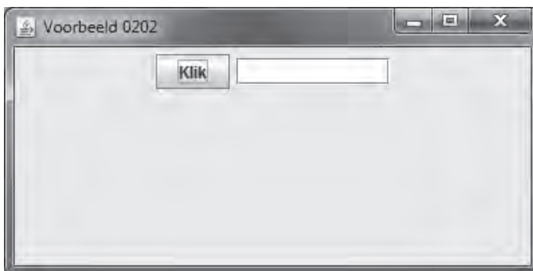
Vervolgens maakt de constructor een tekstvak (een `JTextField`-object) waarin tien tekens passen:

```
tekstvak = new JTextField( 10 );
```

De `add()`-opdrachten in de constructor zorgen ervoor dat de knop en het tekstvak in het `JPanel` komen:

```
add( knop );
add( tekstvak );
```

In figuur 2.2 zie je het resultaat op het scherm. Hoe dit resultaat precies tot stand komt lees je in de volgende paragraaf.



Figuur 2.2

2.6 Samenwerking tussen objecten

In paragraaf 2.2 staat een opstartklasse die een venster maakt voor een applicatie en in paragraaf 2.5 staat een klasse die een paneel maakt met een knop en een tekstvak. Je moet er nu voor zorgen dat het paneel de inhoud van het venster wordt. De volgende opdrachten spelen daarbij een rol.

Maak eerst een `JFrame`-object:

```
JFrame frame = new Vb0202();
```

Maak vervolgens een `JPanel`-object:

```
JPanel paneel = new Paneel();
```

Voeg dit paneel toe als inhoud (content) van het frame:

```
frame.setContentPane( paneel );
```

Deze opdrachten zet je in de methode `main()` van de opstartklasse. De broncode van die klasse wordt dan als volgt:

```
// Voorbeeld 0202
// Opstartklasse voor applicatie met paneel
import javax.swing.*;

public class Vb0202 extends JFrame
{
  public static void main( String args[] )
  {
    JFrame frame = new Vb0202();
    frame.setSize( 400, 200 );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setTitle( "Voorbeeld 0202" );
    JPanel paneel = new Paneel();
    frame.setContentPane( paneel );
    frame.setVisible( true );
  }
}
```

Merk op dat deze opstartklasse bijna identiek is aan die van voorbeeld 0201 in paragraaf 2.2. De paar verschillen zijn in de code hiervoor vetgedrukt.

Dit is de broncode van de klasse `Paneel`:

```
class Paneel extends JPanel
{
  private JButton knop;
  private JTextField tekstvak;
  public Paneel()
  {
    knop = new JButton( "Klik" );
    tekstvak = new JTextField( 10 );
    add( knop );
    add( tekstvak );
  }
}
```

De objecten `frame` en `paneel` werken samen om het resultaat van figuur 2.2 te krijgen. Hieronder staat nog een keer kort samengevat wat er precies gebeurt:

- Bij het starten van de applicatie wordt als eerste de methode `main()` uitgevoerd.

De methode `main()` doet een groot aantal dingen:

- Hij maakt een vensterobject door middel van `JFrame frame = new Vb0202();`
- Hij geeft het venster de juiste afmeting met `frame.setSize( 400, 200 );`
- Hij zorgt dat het sluitknopje van het venster goed werkt door middel van `frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );`
- Hij geeft het venster een titel met `frame.setTitle( "Voorbeeld 0202" );`

Vervolgens maakt hij een `JPanel`-object met behulp van de opdracht:

```
JPanel paneel = new Paneel();
```

De volgende stap is dat hij dit paneelobject tot de inhoud van het venster maakt met de opdracht:

```
frame.setContentPane( paneel );
```

En de laatste stap is het venster (en zijn inhoud) zichtbaar maken:

```
frame.setVisible( true );
```

Maar dit is niet het hele verhaal. Want bij het maken van het `JPanel`-object gebeuren ook verschillende dingen:

- Bij het maken van het `JPanel`-object wordt de constructor van de klasse `Paneel` aangeroepen.

De constructor maakt een knop-object en een tekstvak-object en voegt deze aan het paneel toe zodat ze te zien zijn als het venster op het scherm zichtbaar wordt.

2.7 Oefeningen

Oefening 2.8

Maak in JCreator een nieuw project met de naam `Vb0202`. Tik de broncode in van de opstartklasse en daaronder in hetzelfde bestand de klasse `Paneel` uit paragraaf 2.6. Laat de code vertalen en uitvoeren. Als het goed is krijg je een uitvoer als in figuur 2.2 te zien.
Merk op dat je op de knop kunt klikken en tekst kunt invoeren in het tekstvak, maar dat er verder niets gebeurt.

Oefening 2.9

Declareer in de klasse `Paneel` uit de vorige oefening een referentie voor een tweede tekstvak. Zet deze declaratie als derde regel in de body van de klasse `Paneel`, bijvoorbeeld zo:

```
private JTextField tekstvak2;
```

Maak vervolgens in de constructor van de klasse `Paneel` een tweede tekstvak-object:

```
tekstvak2 = new JTextField( 20 );
```

Voeg dit tweede tekstvak toe aan het paneel:

```
add( tekstvak2 );
```

Laat de code vertalen en uitvoeren. Ziet het resultaat eruit zoals je verwacht?

2.8 Event-afhandeling

In de applicatie van voorbeeld `0202` kun je op de knop klikken wat je wilt, er gebeurt verder niets: alleen het uiterlijk van de knop verandert iets als je erop geklikt hebt. Het klikken op een knop heet in het Engels een event, een gebeurtenis. Als er zo'n event plaatsvindt, moet het programma daarop reageren. Dat heet event-afhandeling (event handling).

In een objectgeoriënteerd programma verrichten objecten alle taken. Een object heeft daarvoor de beschikking over een of meer methoden waarin gedefinieerd is welke handelingen het object kan verrichten. Voor specialistische taken heb je gespecialiseerde objecten nodig. Een voorbeeld van een gespecialiseerd object is een knop. Wat een knop kan, is vastgelegd in de methoden van de klasse `JButton`, een klasse die is voorgedefinieerd in de package `javax.swing`. Maar een knop kan geen klik-event afhandelen. De makers van de Java-library kunnen immers niet weten hoe jij als programmeur het programma wilt laten reageren op een bepaalde knop.

Het afhandelen van een event, zoals het klikken op een knop, is ook een gespecialiseerde taak. Daar heb je een *event handler* voor nodig. Daarvoor moet je eerst een zogeheten *inwendige klasse* (inner class) definiëren. Hoe dat in zijn werk gaat, kun je zien in het voorbeeld in de volgende paragraaf.

2.8.1 Een event handler

Als eerste zie je hier de opstartklasse. Deze is in wezen hetzelfde als in het vorige voorbeeld, er staat alleen een extra `import`-opdracht. Die is nodig omdat je daarmee de package importeert die de voorgedefinieerde klassen en zogeheten *interfaces* bevat die nodig zijn om event-afhandeling goed te laten verlopen.

```
// Voorbeeld 0203
// Opstartklasse voor applicatie met paneel en event handler
import javax.swing.*;
import java.awt.event.*;

public class Vb0203 extends JFrame
{
  public static void main( String args[] )
  {
    JFrame frame = new Vb0203();
    frame.setSize( 400, 200 );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setTitle( "Voorbeeld 0203" );
    JPanel paneel = new Paneel();
    frame.setContentPane( paneel );
    frame.setVisible( true );
  }
}
```

De belangrijkste verandering zie je vetgedrukt in de klasse `Paneel`:

```
// Paneel met event handler
class Paneel extends JPanel {
  private JButton knop;
  private JTextField tekstvak;

  public Paneel()
  {
    knop = new JButton( "Klik" );
    KnopHandler kh = new KnopHandler();
    knop.addActionListener( kh );
    tekstvak = new JTextField( 10 );
    add( knop );
    add( tekstvak );
  }

  // Inwendige klasse
  class KnopHandler implements ActionListener
  {
    public void actionPerformed( ActionEvent e )
    {
      tekstvak.setText( "Je hebt geklikt!" );
    }
  }  // Einde van de inwendige klasse
}  // Einde van de klasse Paneel
```

De inwendige klasse heb ik `KnopHandler` genoemd en deze is gedefinieerd *in* de body van de klasse `Paneel`. Daarom heet `KnopHandler` een *inwendige klasse*.

De klasse `KnopHandler` moet een methode krijgen met de naam `actionPerformed()`. Deze methode handelt in feite de event af. Het woord action is een ander woord voor event.

Achter de naam van de klasse `KnopHandler` staat `implements ActionListener`. Je zult later zien wat dat precies betekent. Eenvoudig gezegd komt het erop neer dat een `KnopHandler`-object een `ActionListener` is, dat wil zeggen: een object dat luistert of een bepaalde actie (event) plaatsvindt.

In de constructor van `Paneel` staat de volgende opdracht:

```
KnopHandler kh = new KnopHandler();
```

Hiermee maak je een object van de klasse `KnopHandler`. Met de volgende opdracht maak je dit bekend aan de knop:

```
knop.addActionListener( kh );
```

Dit heeft tot gevolg dat als je op de knop klikt, de methode `actionPerformed()` van de knop-handler geactiveerd wordt. Deze methode ziet er zo uit:

```
public void actionPerformed( ActionEvent e )
{
  tekstvak.setText( "Je hebt geklikt!" );
}
```

Door de opdracht `tekstvak.setText( "Je hebt geklikt!")` komt er een tekst in het tekstvak, zie de uitvoer in figuur 2.3.

Figuur 2.3

Via `ActionEvent e` die tussen de haakjes van de methode `actionPerformed()` staat, komt informatie over de event de methode binnen. In sommige gevallen is het nuttig van deze informatie gebruik te maken, maar hier is dat niet nodig. In plaats van de naam `e` kun je ook een andere naam kiezen, `f` bijvoorbeeld, of `event`.

2.8.2 Een naamloze instantie

In plaats van het woord object gebruiken we ook vaak het woord *instantie* (Engels: instance). Als het om een knop-object gaat, kun je dan bijvoorbeeld zeggen:

- een instantie van de klasse `JButton`;
- een instantie van `JButton`;
- een object van de klasse `JButton`;
- een `JButton`-object.

In de constructor van `Paneel` staat een opdracht die een instantie maakt van `KnopHandler`:

```
KnopHandler kh = new KnopHandler();
```

Vervolgens wordt deze instantie bekendgemaakt bij de knop:

```
knop.addActionListener( kh );
```

Deze twee opdrachten kun je ook in één doen:

```
knop.addActionListener( new KnopHandler() );
```

In deze opdracht maak je een naamloze instantie van `KnopHandler` en geeft deze aan `knop` door. Zo'n naamloze instantie is handig als je verder toch geen referentie naar deze instantie nodig hebt. In sommige voorbeelden in dit boek zal ik gebruikmaken van deze korte manier.

2.8.3 Wat gebeurt er precies bij event-afhandeling?

Als je de applicatie `Vb0203` laat uitvoeren, gebeurt er het volgende:

- Als eerste wordt de methode `main()` van de opstartklasse uitgevoerd. Deze maakt een frame-object en een paneel en voegt dit paneel toe aan het venster.
- Het paneel bestaat uit een knop en een tekstvak. De knop is gekoppeld aan een instantie van `Knophandler` die een methode heeft met de naam `actionPerformed()`.

Als je niet op de knop klikt gebeurt er niets. Als je dat wel doet gebeurt er het volgende:

- De knop zorgt ervoor dat `actionPerformed()` wordt geactiveerd. Dat wil zeggen dat de opdracht in de body van deze methode wordt uitgevoerd en daardoor komt er een tekst in het tekstvak. De event is afgehandeld.

2.9 Twee knoppen en een tekstvak

Het vorige voorbeeld wordt interessanter als je er een tweede knop aan toevoegt die het tekstvak schoonveegt zodra je op deze knop klikt. Er zijn dan twee knoppen die iets verschillends moeten doen. Elke knop krijgt zijn eigen event-handler: in het ene geval moet er een tekst in het tekstvak komen, in het andere geval moet je het tekstvak leegmaken. De volgende broncode zorgt hiervoor.

Eerst de opstartklasse:

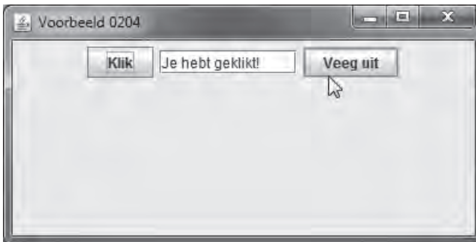
```
// Voorbeeld 0204
// Opstartklasse voor applicatie met paneel
// voor twee knoppen en tekstvak
import javax.swing.*;
import java.awt.event.*;

public class Vb0204 extends JFrame
{
  public static void main( String args[] )
  {
    JFrame frame = new Vb0204();
    frame.setSize( 400, 200 );
    frame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    frame.setTitle( "Voorbeeld 0204" );
    JPanel paneel = new Paneel();
    frame.setContentPane( paneel );
    frame.setVisible( true );
  }
}
```

En vervolgens het paneel:

```
// Paneel met twee knoppen en een tekstvak
class Paneel extends JPanel
{
  private JButton knop, herstelknop;
  private JTextField tekstvak;

  public Paneel()
  {
    knop = new JButton( "Klik" );
    knop.addActionListener( new KnopHandler() );

    herstelknop = new JButton( "Veeg uit" );
    herstelknop.addActionListener( new HerstelknopHandler() );
```

```
    tekstvak = new JTextField( 10 );
    add( knop );
    add( tekstvak );
    add( herstelknop );
  }

  // Inwendige klasse
  class KnopHandler implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
      tekstvak.setText( "Je hebt geklikt!" );
    }
  }

  // Nog een inwendige klasse
  class HerstelknopHandler implements ActionListener {
    public void actionPerformed( ActionEvent e ) {
      tekstvak.setText( "" );
    }
  }
}  // Einde van de klasse Paneel
```

De uitvoer (vlak voor het uitvegen) is te zien in figuur 2.4.



Figuur 2.4

Een tekstvak maak je leeg door de zogenoemde *lege string* erin te zetten. De lege string bestaat uit twee dubbele aanhalingstekens met niets ertussen: `""`. Het tekstvak maak je leeg met deze opdracht:

```
tekstvak.setText( "" );
```

Een alternatieve manier om de events van de twee knoppen af te handelen zie je in paragraaf 5.3.2.

2.10 Regels voor de opmaak van broncode

Naarmate je het programmeren in Java beter beheerst, kan de broncode een aanzienlijke lengte krijgen. Om dergelijke programma's toch leesbaar te houden, is het verstandig van meet af aan aandacht te besteden aan de opmaak van de broncode.

Er zijn voor de opmaak van broncode veel verschillende stijlen in omloop. Dit is een veel toegepaste:

- zet een openingsaccolade in zijn eentje op een regel;
- spring na elke openingsaccolade twee spaties in;
- zorg dat de regels daarna recht onder elkaar staan;
- spring bij elke sluitaccolade weer twee spaties terug.

De opmaak van een stukje code ziet er dan als volgt uit:

```
// Stijl 1
class Voorbeeld1
{                  // openingsaccolade
  declaratie1;     // 2 spaties inspringen
  declaratie2;
                   // regel wit
  methode()        // methode
  {                // openingsaccolade
    statement1;    // 2 spaties inspringen
    statement2;
  }                // sluitaccolade: spring terug
}                  // sluitaccolade: spring terug
```

Zoals je ziet, staan de bij elkaar horende openings- en sluitaccolade recht onder elkaar. Door consequent deze stijl toe te passen kun je de opmaak van elk programma overzichtelijk houden, waardoor bijvoorbeeld het opsporen van fouten makkelijker wordt. Deze manier van opmaken is toegepast in alle voorbeelden van dit hoofdstuk.

De hier gegeven opmaak is maar één manier om een programma vorm te geven. Een andere opmaakstijl is die waarbij de openingsaccolade niet meer alleen op de regel staat:

```
// Stijl 2
class Voorbeeld2
{ declaratie1;      // openingsaccolade
  declaratie2;      // inspringen
                    // regel wit
  methode()         // methode
  { statement1;     // openingsaccolade
    statement2;     // inspringen
  }                 // sluitaccolade: spring terug
}                   // sluitaccolade: spring terug
```

Ten slotte een opmaakstijl waarbij de openingsaccolade niet op een nieuwe regel staat, maar aan het eind van de regel waar hij bij hoort:

```
// Stijl 3
class Voorbeeld3 {  // kop van klasse met openingsaccolade
  declaratie1;      // inspringen
  declaratie2;
                    // regel wit
  methode() {       // kop van methode met openingsaccolade
    statement1;     // inspringen
    statement2;
  }                 // sluitaccolade: spring terug
}                   // sluitaccolade: spring terug
```

Ik heb een lichte voorkeur voor de laatste stijl, en zal die dan ook in de volgende hoofdstukken toepassen. Het is een kwestie van smaak en gewoonte. Het maakt niet veel uit welke stijl je kiest, maar probeer die stijl wel vol te houden, zodat de broncode er overzichtelijk uitziet.

2.10.1 Witregels

Afgezien van de plaats van de accolades is het verstandig af en toe een regel wit in je programma aan te brengen als scheiding tussen de verschillende onderdelen, bijvoorbeeld tussen twee methoden van een klasse. Gebruik niet te veel witregels. De broncode krijgt daardoor een verbrokkeld aanzien en wordt minder goed te begrijpen.

2.11 Regels voor namen in Java

Niet elke naam die je zelf verzint is geschikt als naam in een Java-programma. Namen van objecten, en ook die van klassen en methoden, moeten voldoen aan vaste regels. Een naam mag je opbouwen uit:

- kleine letters, zoals a, b, c;
- hoofdletters, zoals A, B, C;
- cijfers, zoals 0, 1, 2;
- het onderstreepteken _ (de underscore);
- het dollarteken $.

Je mag namen zo lang maken als je wilt, maar een naam mag nooit beginnen met een cijfer. Voorbeelden van namen die aan deze regels voldoen zijn:

```
teller      ditIsEenNaam   aantalPersonen   $bedrag
x           x1             x2               x3
hoogte      Hoogte         Jeannette        aap
breedte     BREEDTE        x_1              x_2
```

Namen die niet aan deze regels voldoen zijn bijvoorbeeld:

`2a`	begint met een cijfer
`Dit is geen naam`	bevat spaties
`d'66`	bevat apostrof
`4711`	begint met een cijfer
`C&A`	bevat & (ampersand)
`C++`	bevat plustekens

2.11.1 Gebruik van hoofdletters en kleine letters

Houd er rekening mee dat Java verschil maakt tussen hoofdletters en kleine letters; hoogte is dus een andere naam dan Hoogte (en HOOGTE is weer een andere). Wat betreft de naamgeving is er een aantal vaste gewoonten waar elke goede Java-programmeur zich aan houdt:

- Namen van klassen beginnen altijd met een hoofdletter; daarvan heb je al enkele voorbeelden gezien: JFrame, JPanel, JButton, ActionListener, KnopHandler.
- Namen voor objecten beginnen altijd met een kleine letter: f, g, knop, tekstvak, teller.
- Namen van objecten geven bij voorkeur zo precies mogelijk aan wat de inhoud van dat object is. De naam `temperatuur` of `tijd` is veel duidelijker dan de naam `t`. En `nieuwSaldo` en `oudSaldo` zijn duidelijker dan `saldo1` en `saldo2`.
- Bestaat de naam van een object uit twee of meer woorden, dan begint het eerste woord met een kleine letter en het volgende woord met een hoofdletter vast aan het voorafgaande woord. Bijvoorbeeld: `aantalEuros`, `nieuwSaldo`, `percentageEersteSchijf`.
- Voor namen van methoden gelden dezelfde regels als voor de namen van objecten. Een methode kun je herkennen aan de ronde haakjes openen en sluiten, bijvoorbeeld `actionPerformed()`.

Ik raad elke beginnende Java-programmeur sterk aan om deze regels heel nauwkeurig toe te passen. In dit boek zal ik dat ook doen. Ook de programmeurs van de Java-bibliotheek houden zich aan deze regels. Hierdoor wordt broncode voor iedereen leesbaarder en begrijpelijker.

2.12 Samenvatting

- Elke Java-applicatie heeft een opstartklasse met daarin een methode met de naam `main()`.
- Een methode heeft een kop en een body.
- In de body van een methode staan opdrachten die van boven naar beneden worden uitgevoerd.
- In de methode `main()` maak je een `JFrame`-object, een venster voor de applicatie.
- De inhoud van het venster maak je door middel van een `JPanel`-object.
- Voor het paneel maak je een aparte klasse.
- Ook een klasse heeft een kop en een body.
- In de body van een klasse staan een of meer methoden.
- In een `JPanel`-object kun je componenten zoals knoppen en tekstvakken zetten.
- Om een knop werkend te krijgen, moet je event-handling toepassen.
- Voor de afhandeling van een event maak je een inwendige klasse.
- Achter de naam van de inwendige klasse voor event-handling staat `implements ActionListener` en deze klasse moet een methode krijgen met de naam `actionPerformed()`.
- Java is case sensitive, hoofdlettergevoelig.
- Het is belangrijk je te houden aan de gewoonten voor het gebruik van kleine letters en hoofdletters in namen.
- Namen in Java mogen alleen letters, cijfers, onderstreeptekens en het dollarteken bevatten, maar niet beginnen met een cijfer.

- Een naam voor een object heet een referentie.
- Een object maak je met `new` gevolgd door de naam van de klasse waarvan je een instantie wilt maken.
- Als je `new` gebruikt, wordt de constructor van de betreffende klasse uitgevoerd.

2.13 Vragen

1. Wat is een klasse?
2. Wat is een object?
3. Wat is een instantie?
4. Wat is een package?
5. Wat is een declaratie?
6. Wat is een methode?
7. Wat is een referentie?
8. Wat is een event?
9. Java is case sensitive. Wat betekent dat?
10. Java is een objectgeoriënteerde taal. Wat betekent dat?
11. Welke gewoonten bestaan er voor de naamgeving van klassen, methoden en objecten?
12. Aan welke regels moeten namen in Java voldoen?
13. Welke namen zijn toegestaan in Java:
```
contanteWaarde
muisknop
U2
4U
c:\temp
Coca Cola
xs4all
```
14. Welke toevoegingen aan de broncode zijn nodig zijn om een klik op een knop af te handelen?

2.14 Oefeningen

Oefening 2.10

Maak een applicatie met twee knoppen en twee tekstvakken. Als je op de eerste knop klikt, moet je voornaam in het eerste tekstvak komen; als je op de tweede knop klikt, komt je achternaam in het tweede tekstvak.

Oefening 2.11

Maak een applicatie met drie knoppen en een tekstvak. Bij elke knop hoort een eigen tekst, dat wil zeggen als je op een knop klikt, verandert de tekst in het tekstvak.

Oefening 2.12

Ga na of je de broncode voor de vorige twee oefeningen hebt geschreven volgens de opmaakregels uit paragraaf 2.10. Verbeter zo nodig. Voldoet de naamgeving die je hebt gebruikt aan de regels uit paragraaf 2.11?

Index

Symbolen
@
voor javadoc 173
@author 176
.class 4
@Override 199
@param 177
@return 177
@see 176
@throws 177, 267
@version 176

A
aanroep
van methode 32
abstract 243
abstracte klasse 235, 243
verschil met interface 246
abstracte methode 235, 238, 243
abstractie 238
AbstractOnderdeel 244
Abstract Window Toolkit 7
access modifiers 190
accessor 124, 127
action 20
ActionListener 20, 242
actionPerformed() 20
actueel argument 135
add() 16
addActionListener() 20
afbeelding
uit bestand 312
afgeleide klasse 181
algoritme 73
sorteren van drie waarden 100
verwisselen van twee waarden 100
ambiguïteit 139
analyse 153
analyseren 166
animated gif 283, 313
animatie 283, 287
annotatie 199
anonieme klasse 228
hoe maak je ze? 230
append() 323
applet 3
applet maken 321
applicatie 3, 7
argument 32, 75
actueel 135
formeel 135
lokale variabele 132
ArithmeticException 266
ArrayList 161, 167
assignment-operator 31
assignment-statement 31
associatie 157
attribuut 53, 108, 127
initialisatie 53
author 176
Auto 235
autoboxing 170
AWT 7

B
Bal 290
balpennen 120
Bank 181
Bankrekening 130, 132, 181
beeldpunt 9
bereik
van int 37
besturingssysteem 2
Blocked 308
block tag 173
blok 52
body
van anonieme klasse 229
van for-statement 104, 105
van if-statement 92, 94
van klasse 12
van methode 11
van while-statement 113
boolean 29, 169
Boolean 87, 92
Boole, George 87
border 218
Border 81
BorderFactory 220
BorderLayout 211
BorderLayout.CENTER 212
BorderLayout.EAST 212
BorderLayout.NORTH 212
BorderLayout.SOUTH 212
BorderLayout.WEST 212
bounding box 46
boxing 170
BoxLayout 215
broncode 7
buitenklasse 226
burgerlijke stand 75
byte 29, 169
bytecode 3

C
call stack 275
case sensitive 8
cast 61
catch-blok 269
char 29
Character 169
checked exception 272
cirkel 46, 236
gevuld 46
class 12
class library 4
class member 123
coherent 130
collectie 153, 161
commandvenster 89
commentaar 13
voor javadoc 173
compiler 4
compileren
zie vertalen

component 14, 207
compound assignment
operator 40
CompoundBorder 219
concatenatie
van string en int 33
concrete klasse 244
concurrency 283
conditie 92, 104
while-statement 113
consoleapplicatie 89
consoleprogramma 88
consolevenster 89
constante 64
constructor 15, 77, 133
aanroepen met this() 138
en overerving 195
constructor overloading 135
container 14, 207, 217
controlegedeelte
van for-statement 103
controlevariabele
naam van 110
converteren
String naar double 62
String naar int 51
van double naar int 62
van int naar double 61
van int naar String 32
coördinaten 33
Courier 107

D
data hiding 75
Datum 140
Dead 308
declaratie 15
verkorte 34
decrement 41
deelbaarheid 99
default constructor 77, 195
onzichtbaar 136
zichtbaar 136
default initialisatie 53
default toegang 191
documentatie 172
domeinbegrip 156
domeinklasse 156
domeinmodel 156
dood 302
double 29, 169
Double.MAX_VALUE 60
Double.MIN_VALUE 60
Double.parseDouble() 63
do-while-statement 118
draad 283
drawArc() 48
drawLine() 43
drawOval() 46
drawRect() 46
drawRoundRect() 48
drawString() 32
dynamische binding 200
dynamisch type 184

E
Eclipse 5
editor 3
een-op-veelrelatie 159
ellips 46
gevuld 46
EmptyBorder 219
enclosing class 226
en-operator 88, 90
equals() 102
equalsIgnoreCase() 103
Error 271
error message 4
EtchedBorder 219
even 99, 119
event 19
listener 225
event handler 19
exceptie 257
checked exception 272
geconctroleerde 272
genereren 257
IndexOutOfBoundsException 272
-klassen 271
ongecontroleerde 272
opwerpen 264
unchecked exception 272
exception 271
handling 257
Exception
ArithmeticException 266
expressie 38
extends 189

F
false 92
Fibonacci 122
fillArc() 48
filler 216
fillOval() 46
fillRect() 46
fillRoundRect() 48
final 64
finally 263
flag 92
FlexRechthoek 197, 198
float 29, 169
floating point 59
FlowLayout 64, 207
ruimte tussen
componenten 209
Font 108
for-each
-loop 163
format specifier 70, 107
format string 70, 107
formatteren
van uitvoer 70
formeel argument 135
for-statement 103
andere beginwaarde 110
body 104
gelijkwaardigheid met
while-statement 117
grotere stappen dan 1 110
naam van controlevariabele
110
terugtellen 110
variabele begin- of
eindwaarde 110
waarvan body niet wordt
uitgevoerd 111
foutmeldingen 4

friendly toegang 191
fully qualified name 285
functiewaarde
  tekenen 249
  uitrekenen 249

G
gebruikersinterface 7
gecontroleeerde exceptie 272
gehele deling 36
generalisatie 181, 187, 188
getBounds() 224
getGraphics() 301
getHeight() 311
getMessage() 267
getSource() 97
getter 76, 124, 127
getText() 51
getWidth() 311
gif 312
glue 216
grafische context 31
  naam van 45
Graphics-argument
  methode 145
GridLayout 81, 214
  lege plekken vullen 86
GUI 207
  builder 207

H
hashcode 141, 203
header 11
  zie kop
herdefinitie 198
  versus overloading 198
herhaling
  oneindige 117
herhalingsopdracht 87
  for-statement 103
  while-statement 113
hoofdlettergevoelig 8
hoofdletters en kleine letters 26
html-code
  voor applet 322

I
IDE 5
if-else-statement 97
if-statement 92
ImageIcon 312
image observer 313
immutable 323
implementatie 126, 166
import 12, 31
increment 41
IndexOutOfBoundsException 272
Infinity 264
inheritance 181, 190
initialisatie
  boolean attribuut 92
  default 53
  in for-statement 104
  van attributen 53
  van lokale variabelen 53
  van objectvariabelen 139
  van variabele voor while-statement 115
initialiseren 34
inner class 19, 225
instance 21
instance variable 53
instantie 21, 128
  naamloze 21
int 29
  bereik 37
  negatieve waarden 38
Integer 51, 169
Integer.MAX_VALUE 37
Integer.MIN_VALUE 37
Integer.parseInt() 51
interface 235, 238
  implementeren in anonieme klasse 228
  in UML 239
  meer dan één interface implementeren 246
  tagging 243
  verschil met abstracte klasse 246
Internationalization 72
inwendige klasse 19, 20, 225
is een relatie 192
iteratie 87

J
J2SE 5
Java
  verschillende versies 6
Java 2 Platform 6
Java 2 SDK 5
Java-besturingspaneel 325
Java Control Panel 325
javadoc 172, 267
  bij klassen 175
  commentaar 173
Javadoc
  bij constructors en methoden 176
Java Runtime Environment 3
Java SE 5
Java Virtual Machine 3
JButton 15
JCreator 5, 7
JDK 5
  broncode van 242
JFrame 8, 12
JLabel 64
  leeg 86
JPanel 14
  coördinaten 33
  grafische context opvragen 301
jpeg 312
jpg 312
JRE 3
JScrollPane 166
JTextArea 165
JTextField 15
  event 65
  uitlijnen 70
  uitschakelen 69
JUnit 142
JVM 3

K
kassa 73
-bon 171
Kassa 85
klasse 76
keuze 97
-opdracht 92
klasse 12
abstracte 235
anonieme 228
concrete 244
Datum 141
testen 142
uitbreiden met methode 146
voor een rechthoek 144
voor excepties 271
klassendiagram 124
overerving 188
klassenlid 123
kleur 44
zelf mengen 44

L
label 64
leeg 86
lay-out
van broncode 24
lay-outmanager 64, 207
FlowLayout 207
uitschakelen 67
Leerling 152
lege body 106
lege string 23, 33, 51
Les 160
Lesrooster 161
lettertype 108
library 4
LIFO 276
LineBorder 219
links uitgelijnd 71
ListStack 256
literal
int 31
locale 72
logische
operator 88
logische fontnaam 108
lokale variabele 52, 75, 103
argument 132
twee variabelen met dezelfde naam 52
long 29, 169
loop 87
loose coupling 225
loosely coupled 225
losjes gekoppeld 225
LoweredBevelBorder 219
lus 87

M
machinecode 1
main() 11
mainframe 1
main thread 293
MatteBorder 219
method call 32
methode 11
aanroep 32
abstracte 238
kop 11
microcomputer 1
minteken
in UML 124
model 78, 220
model en view
scheiden 220
modelleren 155
MonoSpaced 109
multipliciteit 157, 158
multithreading 283
mutator 125, 127

N
naamloos object
van anonieme klasse 229
naamloze instantie 21
namen
van constanten 64
van objecten, klassen en methoden 25
navigeerbaarheid 158
New Thread 308
niet-operator 91
niet-proportioneel lettertype 107
no-argument constructor 136
notify() 306
null 53, 77
NullPointerException 53

O
object 182, 203
de klasse 141
en toString() 141, 203
Object Bench
zie objectenbank
objectgeoriënteerd 10
objectvariabele 53
initialisatie 139
of-operator 91
Onderdeel 238
onderhoudbaarheid 201
onderstrepingsteken 39, 60
oneindige herhaling 117
oneven 99, 119
ongecontroleerde exceptie 272
ontwerpen 166
ontwerpmodel 159
operand 90
operator
- 34, 60
-- 41
! 91
!= 88
* 34, 60
/ 34, 60
&& 88
% 34, 60, 61, 99
+ 34, 60
++ 41
+= 40
< 88
<= 88
== 88, 102
> 88
>= 88
|| 91

en 90
logische 88
of 91
relationele 87
opmaak
van broncode 24
opstartklasse 7
Oracle 3
outer class 226
overerving 181, 190
klassendiagram 188
overloading
constructor 135
versus overriding 198
overriding 197
versus overloading 198
overschrijven 198

P
package 12, 191
package-toegang 191
param 177
parameter 32, 75
parseDouble() 63
parseInt() 51
en exceptie 261
pauzeren 302
peek 256
pen 45
pixel 9
platform 2
plusteken
in UML 125
png 312
polymorfie 203
pop 256
postfix 42
prefix 42
primitief type 29
int 29
printf 172
printStackTrace() 276
prioriteit 39
private 75, 190
in UML 124
procentteken
in format string 186
processor 1
programma 1
proportioneel lettertype 107, 109
protected 190
public 190
in UML 125
klasse 74
punt-operator 10
push 255

R
RaisedBevelBorder 219
rand 81, 218
realisatie 239
rechthoek 46, 144, 194, 198, 236
gevuld 46
rechts uitgelijnd 71
referentie 15, 29
van een interfacetype 238
regels wit 25
rekenkundige operator 34, 60
relatie 157
relationele operator 87
repaint() 55
rest 36, 61
retourwaarde 125
return 76, 177
value 125
return value
zie retourwaarde
reusable software 181
RGB 44
rigid area 216
Romeinse cijfers 121
run()
beëindigen 302
Runnable 296, 308
runtime 53
RuntimeException 272

S
samengestelde toekennings-operator 60
samengestelde toekennings-operator += 40
SansSerif 109
scheduler 307
scheiden
van model en view 220
scheidingsteken
in getallen 72
schema
voor applet met thread 309
schreef 109
schrikkeljaar 120
schuifbalk 166
scientific notation 59
scope 52
van objectvariabele 53
SDK 5
see 176
selecteren
tekst in tekstvak 263
Serif 109
setBackground() 44
setBounds() 69
setColor 44
setDefaultCloseOperation() 10
setEditable() 69
setEnabled() 69
setHorizontalAlignment() 70
setLayout() 211
setLayout(null) 67, 207
setLocation 67
setLocation() 9
setSize 67
setSize() 10
setter 125, 127
setTitle() 10
setVisible() 11, 69
short 29, 169
signatuur 139
slaap() 295
slashes 13
sleep() 296

sorteeralgoritme 100
source code 7
 zie broncode
Spaarrekening 181
 broncode 184
SPARC 1
specialisatie 181, 188
src.zip 242
stack 255, 275
stapel 275
start()
 van Timer 285
state 133
 zie toestand
statement 12
statisch type 184
Steen 287
string 11
 lege 23, 33, 51
 vergelijken van strings 101
StringBuffer 323
String.format() 70
strut 216
stuiter() 291
subklasse 181
Sun 1, 3
super 190
super() 196
superklasse 181, 187, 238
super.paintComponent(g) 32
Swing 7
synchronized 306
System.out.print() 89
System.out.printf() 172
System.out.println() 89

T
tabelvorm 81
tafel van dertien 106
tagging interface 243
Team 152
tekstgebied 165
tekstvak
 uitlijnen 70
 uitschakelen 69
temporary 100
terugkeerwaarde 125
this 127
 om verwarring te
  voorkomen 127
this() 138
thread 283, 293
 Blocked 308
 Dead 308
 main 293
 New Thread 308
 Not Runnable 308
 pauzeren 302
 Runnable 308
 scheduler 307
 schema voor applet 309
 stoppen 301
 toestand van 307
Thread.sleep() 296
throw 177, 264
Throwable 271
Tijdstip 130, 159
timer 283
 starten en stoppen 286
TitledBorder 219
toegangsregels 190, 191
toestand 133
toString() 140
 in domeinklassen 164
translate() 291
true 92
try-catch-blok 259
type
 dynamisch 184
 statisch 184
 van attributen en methoden
  131
typecast 61, 182, 183

U
UI 207
uitbreidbaarheid 202
uitlijnen 71
 in tekstvak 70
uitwendige klasse 226
uitzondering
 zie exception
UML 124
 interface 239
 minteken 124
 multipliciteit 158
 notatieverschillen met Java
  124, 125
 plusteken 125
 private 124, 125
UML-klassendiagram 222
unboxing 170
unchecked exception 272
Unified Modeling Language
 124

V
variabele 29
 lokale 103
veld 53
verantwoordelijkheid 130
Verkeerslicht 151, 222
version 176
vierkant 46
view 78, 220
virtuele machine 3
vlag 92
void 75, 125
voorloopnullen 71, 140

W
waarheidswaarden 87
wait() 306
warnings 4
wetenschappelijke notatie 59
while-statement 113, 114
 gelijkwaardigheid met
  for-statement 117
wikkelklasse 169
wrapper class 169

X
XOR-modus 310

Z
zelfstandig naamwoord 154

Aan de slag met … is een boekenreeks voor hbo-ict. Elk boek behandelt een specifiek softwarepakket of programmeertaal. Op de portal AcademicX.nl kunnen studenten toetsen en oefeningen maken en extra studiemateriaal raadplegen. Daarmee bereiden zij zich optimaal voor op het tentamen.

Aan de slag met Java maakt je wegwijs in de wereld van Java 7, en met name in het maken van Swing-applicaties. Het boek is bestemd voor beginnende Java-programmeurs, studenten die Java moeten leren en iedereen die wil weten wat een klasse, object, methode of interface is.

Java is een objectgeoriënteerde programmeertaal. Alle – vaak abstracte – concepten die daarbij horen worden vanaf het eerste hoofdstuk glashelder uitgelegd, en aan de hand van zo'n 150 goedgekozen voorbeelden gedemonstreerd. *Aan de slag met Java* richt zich naast het programmeren van applicaties met behulp van Swing op de basisprincipes van objectgeoriënteerd ontwerpen en programmeren.

Dankzij de vele vragen en gevarieerde opgaven in het boek en de portal AcademicX.nl (met directe feedback!) is *Aan de slag met Java* zeer geschikt als cursusmateriaal dat door studenten zelfstandig kan worden doorgewerkt.

Gertjan Laan is auteur van diverse succesvolle boeken, waaronder *Aan de slag met C++*, *Datastructuren in Java*, en *OO-programmeren in Java met BlueJ*.

ISBN 978 90 395 2757 3
NUR 123 / 989

www.academicservice.nl