Aan de slag met Java en JavaFX

Objectgeoriënteerd ontwerpen en programmeren

GERTJAN LAAN

+ONLINE OPGAVEN MET DIRECTE FEEDBACK

Boom

Vijfde druk

Aan de slag met Java en JavaFX

vijfde druk

Gertjan Laan

Boom

Meer informatie over deze en andere uitgaven kun je vinden via www.boomhogeronderwijs.nl.

inclusief website!

Met behulp van onderstaande unieke activeringscode kun je een studentaccount maken op **www.aandeslagmetjava.nl**, voor toegang tot extra materiaal bij dit boek. Deze code is persoonsgebonden en gekoppeld aan de vijfde druk. Na activering van de code is de website twee jaar toegankelijk. De code kan tot zes maanden na het verschijnen van een volgende druk geactiveerd worden.

Opmaak: Nu-nique grafische vormgeving, Goor
Omslag: Carlito's Design, Amsterdam
Omslagstramien: Studio Bassa, Culemborg

ISBN: 9789024415663 (paperback)
ISBN: 9789024424573 (e-book)
NUR: 123/989

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, implementatie en testen.

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 objectgeorienteerde 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. 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, en op het testen van de methoden van een klasse met JUnit. 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 (hoofdstuk 8) en abstracte klassen en interfaces (hoofdstuk 9) completeren het objectgeoriënteerde beeld.

De voorbeelden en oefeningen in deze vijfde editie maken gebruik van JavaFX, met praktische oefeningen en tips voor NetBeans en Eclipse.

Aanvullend materiaal, zoals onlinevragen, de broncode van de voorbeelden, de uitwerkingen van de opgaven, is te vinden op de website die Boom uitgevers ten aanzien van dit boek heeft ingericht (*www.aandeslagmetjava.nl*) en op mijn eigen website (*www.gertjanlaan.nl*).

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

Zaandam, maart 2018
Gertjan Laan

Inhoud

Hoofdstuk 1

Introductie

1.1 Inleiding

De programmeertaal Java is in 1995 op de markt gebracht, werd binnen een paar jaar zeer populair, en is dat tot op de dag van vandaag gebleven.
In dit hoofdstuk geef ik een korte beschrijving van de werking van Java en van begrippen die daarbij horen, zoals processor, besturingssysteem, platform, compiler, bytecode, virtuele machine, machinecode, applet, servlet en applicatie.

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.

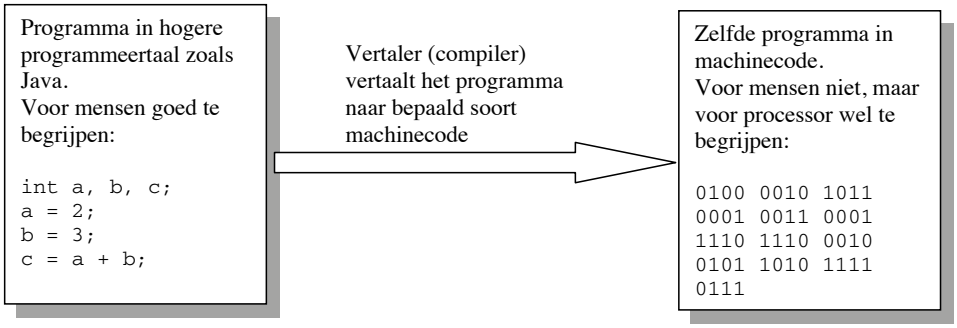
Instructies zijn meestal bij elkaar opgeslagen in een geheugenchip. Zo'n bij elkaar horende reeks instructies heet een *programma*. Een programma wordt uitgevoerd door een andere chip: de *processor*. Nu is het lastige dat de processor geen Nederlands, Engels of Java begrijpt. De enige taal die de processor 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 zij 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: zogeheten *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 iPad met een Apple A9-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 een Apple, dat wil zeggen dat vertaald is naar machinecode voor een Apple, 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. Veel mobiele apparaten hebben weer een ander besturingssysteem, zoals Android.

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 bijna 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 bijvoorbeeld geleverd als plug-in voor webbrowsers, en ook als zelfstandig programma voor allerlei besturingssystemen. 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 en uit te voeren. 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, applet en servlet

Er bestaan verschillende soorten Java-programma's onder andere: applicaties, applets en servlets. Een *applet* is een Java-programma dat deel uitmaakt van een webpagina. Om een applet te kunnen uitvoeren moet je browser beschikken over de Java Virtual Machine als plug-in. Een *servlet* is een programma dat op een (web)server draait. 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:

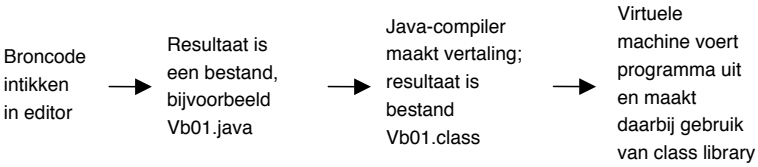
- Je moet eerst de tekst van een Java-programma (de *broncode*) intikken in een *editor*. Een editor voor Java is een tekstverwerker zoals Word, maar meestal veel eenvoudiger. De tekst die je intikt moet je bewaren als bestand op de schijf. De naam van zo'n bestand krijgt `.java` als achtervoegsel.
- De broncode die je hebt ingetikt 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 controleren op fouten, en eventuele 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 (JVM).

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 een kopie van kan krijgen.

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.

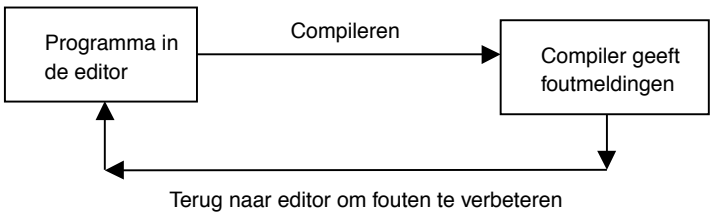


Figuur 1.2

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, zogeheten *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.



Figuur 1.3

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

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;
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, en eventueel een editor, zijn als onderdeel van een gratis pakket verkrijgbaar bij Oracle: *www.oracle.com*. Het hele pakket is in de loop der tijd 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.

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

In dit boek maak ik gebruik van NetBeans 8.2 en van Eclipse 4.7 Oxygen, beide met JDK 8.

In hoofdstuk 2 staat uitgelegd hoe je met NetBeans of met Eclipse de oefeningen kunt maken.

1.7 De software installeren

Om met Java en JavaFX te leren programmeren heb je een geïntegreerde ontwikkelomgeving nodig. Hetzij NetBeans, hetzij Eclipse. Welke van de twee je kiest, maakt niet veel uit, ze zijn allebei goed en gratis. Behalve NetBeans of Eclipse heb je ook de JDK 8 nodig.

Prettige bijkomstigheid van NetBeans is dat deze als compleet pakket met de JDK komt. Bij Eclipse moet je de JDK apart downloaden.

1.7.1 NetBeans en JDK installeren

De eenvoudigste manier om NetBeans met de JDK te vinden, is door te googelen naar 'Oracle Java SE download':
- Kies voor NetBeans met de JDK.

Er zijn versies beschikbaar voor Linux, Mac OS X en Windows. Op het moment van schrijven van dit boek is NetBeans 8.2 met JDK 8u151 de laatste versie. Alle voorbeelden in het boek en de uitwerkingen van de oefeningen zijn met deze versie gemaakt.

1.7.2 JDK en Eclipse installeren

Installeer eerst de JDK voordat je Eclipse installeert. Google naar 'Java Se Development Kit Downloads'.

- Installeer de laatste versie van de JDK.

Er zijn versies beschikbaar voor Linux, Mac OS X en Windows. Op het moment van schrijven van dit boek is JDK 9.0.1 de laatste versie. Alle voorbeelden in dit boek en de uitwerkingen van de oefeningen zijn met versie 8 gemaakt, maar zouden ook met versie 9 moeten werken.

Ga vervolgens naar *www.eclipse.org/downloads*, en klik op de link *Download Packages*. Op de pagina waar je dan komt, kun je kiezen tussen Linux, Windows en Mac OS X. Het handigst is het de Eclipse Installer te gebruiken, en de aanwijzingen te volgen die bij Find out More op deze pagina staan.

- Installeer Eclipse Oxygen.

Ook als je niet via de Eclipse Installer installeert, kies dan in elk geval voor Eclipse IDE for Java Developers.

1.8 Verschillende versies van Java

Sinds 1996 zijn er verschillende versies van de JDK verschenen, met een nogal merkwaardige nummering en naamgeving:

1996, JDK 1.0
1997, JDK 1.1
1998, J2SE 1.2 (met Swing)
2000, J2SE 1.3
2002, J2SE 1.4
2004, J2SE 5.0 of Tiger
2006, Java SE 6 of Mustang
2011, Java SE 7 of Dolphin of JDK 7
2014, Java SE 8 of Spider (met JavaFX) of JDK 8
2017, Java SE 9 of JDK 9

De versies vanaf 1.2 worden nogal verwarrend ook wel aangeduid met het "Java 2 Platform".

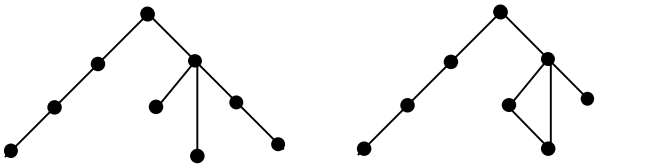
De versies zijn gelukkig *upward compatible*, wat betekent dat alles wat in versie 1.0 kon, nog steeds in versie 1.1 kan, en zo verder.

Voor het maken van een gebruikersinterface (knoppen, invoervakken, menu's, schuifbalken) kent Java drie verschillende bibliotheken: vanaf JDK 1.0 de AWT (Abstract Window Toolkit), vanaf J2SE aangevuld met Swing, en vanaf Java SE 8 ook met JavaFX.

De voorbeelden in dit boek zijn getest met JDK 8 en de gebruikersinterfaces zijn gemaakt met JavaFX.

1.9 Bomen en JavaFX

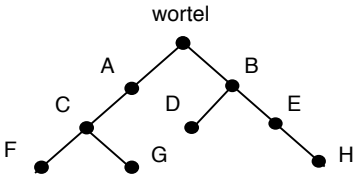
De wijze waarop je met JavaFX een gebruikersinterface opbouwt, kun je je het best voorstellen als een *boomstructuur* of kortweg een *boom* (Engels: *tree*). In figuur 4.14 in paragraaf 4.9 zie je een voorbeeld van de boomstructuur van een zogeheten *scene-graph* in JavaFX.
Het is dan ook handig te weten wat we onder een boom verstaan, en welke begrippen daarbij een rol spelen.



Figuur 1.4

Bomen kennen allerlei toepassingen, zowel binnen als buiten de informatica. Een heel bekende toepassing is de stamboom met voorouders, kinderen en verdere nakomelingen. Een ander bekend voorbeeld van een boom is de bestandsstructuur (de ordening van mappen en bestanden) zoals een programma als Verkenner van Windows laat zien.

In het algemeen is een boom een structuur met verbindingen tussen verschillende elementen waarin in elk geval geen lussen voor komen. De elementen in een boom heten *knopen* (*nodes*) die verbonden zijn door *takken* (*edges*). Als je een boom tekent, stel je de knopen meestal als cirkels of vierkantjes voor en de takken als lijntjes.



Boom met namen bij de knopen

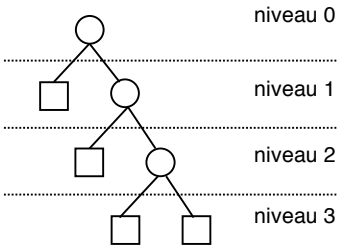
Figuur 1.5

Elke boom heeft één speciale knoop die *wortel* heet (root). De wortel wordt meestal bovenaan getekend en de rest van de knopen eronder. Een knoop die direct onder een andere knoop hangt heet een *kind* (child). In figuur 1.5 zijn bijvoorbeeld de knopen A en B kinderen van de wortel, en F en G zijn kinderen van C. Omgekeerd is C de *ouder* (parent) van F en G, en bijvoorbeeld E is de ouder van H. Alle knopen in een boom hebben precies één ouder, behalve de wortel die er geen heeft.

Een knoop zonder kinderen heet een *blad* of *blaadje* (leaf). In figuur 1.5 zijn F, G, D en H blaadjes. Blaadjes worden ook wel *externe knopen* genoemd (*external nodes*). Knopen die een of meer kinderen hebben, heten *interne knopen* (*internal nodes*).

Kinderen die dezelfde ouder hebben, heten in het Engels *siblings*. Wij zeggen: broertjes en zusjes (of *brusjes*).

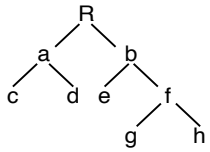
Wanneer je een knoop met al zijn afstammelingen in gedachten losknipt van de boom waarin hij hangt, dan fungeert die knoop als wortel voor een boom die een deel is van de oorspronkelijke boom. Het is een *subboom*. In feite is elke knoop de wortel van een subboom.
De afstand die een bepaalde knoop van de wortel verwijderd is, heet het *niveau* (level) of de *hoogte* (height) van die knoop, zie figuur 1.6.



Figuur 1.6

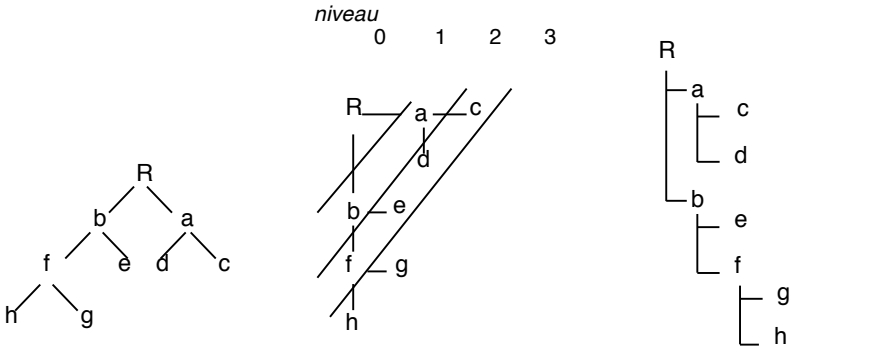
In boeken wordt een boom meestal voorgesteld door een afbeelding zoals in de figuren hiervoor. Nadeel van deze manier van voorstellen is dat dergelijke bomen in drie richtingen kunnen groeien: naar links, naar rechts en naar beneden.

Een boom op een beeldscherm wordt dan ook vaak op een iets andere manier afgebeeld. De boom in Windows Verkenner (of een andere File Manager) die de mappenstructuur van een harde schijf weergeeft, is daar een voorbeeld van. In Verkenner zijn de knopen van links naar rechts geordend naar toenemend niveau. Zo'n boom groeit alleen naar rechts en naar beneden.



Figuur 1.7

Figuur 1.7 zou de boom kunnen zijn van een mappenstructuur, met bovenaan de root R, en daaronder een aantal mappen a tot en met h. In figuur 1.8 zie je hoe de transformatie van deze boom naar de 'verkennerboom' van figuur 1.8c in zijn werk gaat.



a. Gespiegelde boom

b. Draai 45 graden

c. Gelijke niveaus recht onder elkaar

Figuur 1.8

Merk op dat, hoewel de vier bomen verschillend getekend zijn, ze toch precies dezelfde informatie bevatten.

Hoofdstuk 2

Een JavaFX-applicatie maken

2.1 Inleiding

Het maken van een eenvoudige toepassing met Java is niet zo moeilijk. Zeker niet als je over een geïntegreerde ontwikkelomgeving (IDE) beschikt, waarin onder andere een tekstverwerker en compiler zijn ingebouwd. In paragraaf 1.7 staat waar je de ontwikkelomgeving NetBeans of Eclipse kunt downloaden.

In een IDE wordt een gedeelte van het werk vrijwel automatisch gedaan, zoals het vertalen van de broncode en het opslaan en 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 drie *library's*: Abstract Window Toolkit (AWT), Swing en JavaFX. Van deze drie is JavaFX de nieuwste en meest geavanceerde en daarom zullen we deze hier gebruiken.

Een gebruikersinterface bestaat in het algemeen uit een venster met daarin 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 eenvoudige 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 JavaFX-applicatie

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

```
// Opstartklasse voor een JavaFX-applicatie
package fxvb0201;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;

public class FXVb0201 extends Application {
    @Override
    public void start(Stage primaryStage) {
        FlowPane root = new FlowPane();
        Scene scene = new Scene(root, 300, 250);

        primaryStage.setScene(scene);
```

```
        primaryStage.setTitle("Eerste applicatie");
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Dit is nogal een intimiderende hoeveelheid tekst, maar gaandeweg zullen in dit hoofdstuk en de volgende hoofdstukken alle regels van de tekst meer duidelijkheid krijgen.

De opstartklasse `FXVb0201` waarvan je de broncode hierboven ziet, is in feite een volledig JavaFX-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 met een paar knopjes in de rechterbovenhoek. Verder is het venster helemaal leeg, alleen in de titelbalk staat de tekst *Eerste applicatie*. In verschillende besturingssystemen (IOS, Linux, Windows) ziet het venster er verschillend uit. De hier getoonde versie is afkomstig uit Linux.

In JavaFX is het woord *stage* de benaming voor een venster. Een venster is dus het toneel waarbinnen zich van alles afspeelt. De inhoud van het toneel is een *scene*, en de scene heeft een bepaalde opbouw (lay-out), die lay-out is in dit geval een `FlowPane`. Deze heeft in de broncode de naam *root* gekregen, want hij gaat in de volgende paragrafen dienen als de wortel van de boomstructuur voor de onderdelen van deze scene, de root van de scene-graph. Zie paragraaf 1.9 voor de begrippen wortel en boomstructuur.

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 `FlowPane` in en `setSize`, en niet per ongeluk `flowPane` 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:
- Je moet de broncode intikken, en bewaren onder een naam, bijvoorbeeld `FXVb0201.java`. Let op: deze naam moet identiek zijn aan de naam van de opstartklasse, `FXVb0201` in dit geval. Let ook op de drie hoofdletters aan het begin van de naam.
- Vervolgens moet je de broncode laten vertalen door een compiler. De compiler maakt een bestand dat uit bytecode bestaat en dat automatisch de naam `FXVb0201.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.

Hoe een en ander in NetBeans in zijn werk gaat, kun je zelf ervaren door de volgende reeks oefeningen te doen.

2.3 Oefeningen

Afhankelijk van de keuze die je maakt voor NetBeans of Eclipse, voer je de oefeningen van paragraaf 2.3.1 uit of die van paragraaf 2.3.2. De oefeningen in paragraaf 2.3.3 zijn voor beide IDE's hetzelfde.

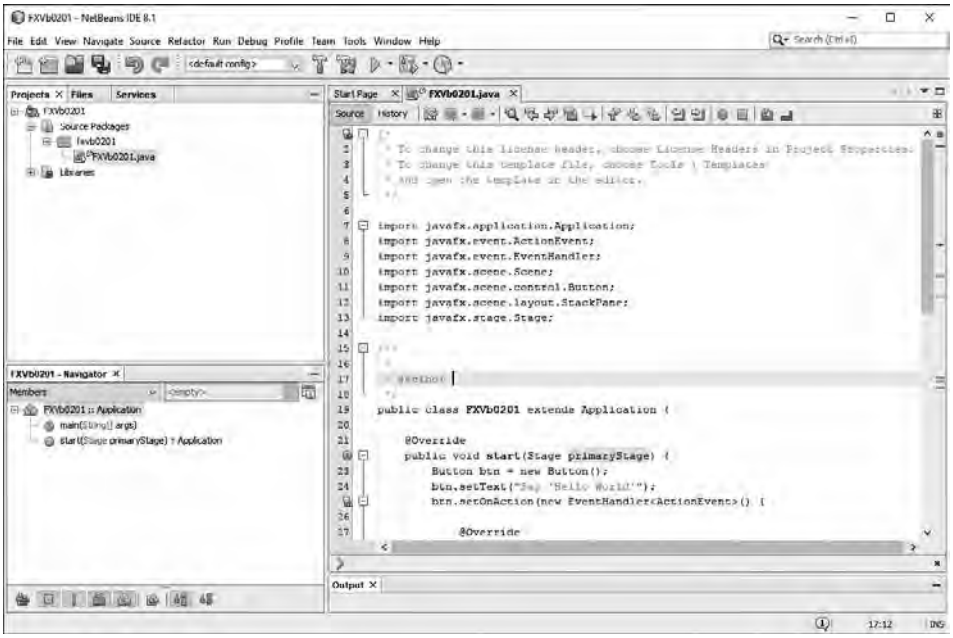
2.3.1 NetBeans-oefeningen

Oefening 2.1 (NB)
Download en installeer NetBeans met de JDK. Zie paragraaf 1.6 en 1.7.

Oefening 2.2 (NB)
Alle bestanden die behoren tot een applicatie die je maakt, zijn in NetBeans georganiseerd in een *project*. Een nieuw project maak je als volgt:
- Open NetBeans
- Kies File | New Project ...
- Klik in het venster dat verschijnt onder Categories op JavaFX
- Klik in hetzelfde venster onder Projects op JavaFX Application
- Klik op de knop Next

Er verschijnt een nieuw venster waarin je de naam van het project moet aangeven en de plaats waar je het wilt opslaan:
- Geef als naam bij Project Name op: FXVb0201 (let op: de eerste drie letters zijn hoofdletters)
- Verander eventueel de locatie waar je het project wil opslaan via de knop Browse, of accepteer de voorgestelde locatie door niets te veranderen
- Klik op Finish

Het project wordt nu door NetBeans gemaakt, en als het goed is zie je een resultaat als dat in figuur 2.2.



Figuur 2.2 *(NB)*

In het hoofdvenster van NetBeans zie je de broncode van het bestand FXVb0201.java. Deze broncode is standaardcode die door NetBeans is gegenereerd.
Mocht je de broncode in het hoofdvenster niet zien, dan kun je deze openen door in het kleine venster linksboven onder het tabblad Projects te klikken op FXVb0201 | Source Packages | fxvb0201 en dan te dubbelklikken op FXVb0201.java.

Oefening 2.3 (NB)

Een project (applicatie) in NetBeans laten uitvoeren betekent dat de broncode opgeslagen en vertaald wordt, en dat de virtuele machine (JVM) wordt gestart om het venster met de applicatie te laten zien.
Een applicatie laten uitvoeren kan op verschillende manieren:

- Druk op het toetsenbord op F6

of

- Kies in de menubalk Run | Run Project

of

- Klik met de muis op het groene driehoekje in de menubalk (de play-knop)

Welke van de drie je kiest, maakt niet veel uit. De keuze voor F6 werkt waarschijnlijk het snelst. Het vertalen van de broncode en het starten van de JVM kost enige tijd, je ziet daarover wat mededelingen verschijnen in het Output-venster onder het hoofdvenster.

Als het goed is verschijnt het venster van de applicatie in beeld met een knop met de tekst 'Say Hello World'. Als je op de knop klikt, verschijnt de tekst *Hello World* in NetBeans in het Output-venster onder het hoofdvenster. Klik enkele malen op de knop om dat te controleren.

Sluit tot slot het venster van de applicatie.

Oefening 2.4 (NB)

Wijzig de broncode uit de vorige oefening zodanig, dat hij exact hetzelfde is als de opstartklasse FXVb0201 in paragraaf 2.2. Typ alles zeer nauwkeurig over (of kopieer de broncode van de website bij dit boek).

Als je klaar bent met de aanpassingen, laat dan het programma uitvoeren (de Play-knop of de toets F6). Als het goed is verschijnt een leeg venster als in figuur 2.1.
Als er geen venster verschijnt, staat er waarschijnlijk een onvolkomenheid in je broncode. Vergelijk deze dan nog een keer goed met die van de opstartklasse in paragraaf 2.2.

Sluit tot slot het venster van de applicatie.
Ga verder met oefening 2.5.

2.3.2 Eclipse-oefeningen

Oefening 2.1 (Ec)

Download en installeer de JDK en Eclipse. Zie paragraaf 1.6 en 1.7.

Oefening 2.2 (Ec)

- Open Eclipse
- Verander eventueel de directory waar je bestanden worden opgeslagen, de *Workspace*, en klik op Launch

Je krijgt eventueel een welkomstscherm te zien. Als je dit scherm niet ziet, kun je het oproepen via Help | Welcome. Dit scherm geeft toegang tot nuttige handleidingen, maar ik sla het voor nu even over. Sluit het welkomstscherm door rechtsboven op Workbench te klikken, of op het kruisje naast het woord Welcome op de tab van het venster.

Alle bestanden die behoren tot een applicatie die je maakt, zijn in Eclipse georganiseerd in een *project*. Een nieuw project maak je als volgt:

- Kies File | New | Java Project

Er verschijnt een venster waarin je de naam van het project moet aangeven:

- Geef als naam bij Project Name op: Voorbeeld0201
- Klik op Finish

Er wordt nu door Eclipse op de schijf ruimte voor het project gemaakt, en als het goed is zie je de naam van het project aan de linkerkant van je scherm in de Package Explorer (als de Package Explorer niet zichtbaar is, kun je hem zichtbaar maken via Window | Show View | Package Explorer).

Voeg nu aan het project een Java-klasse toe:

- Klik in de menubalk op de knop New Java Class, dat is de knop met een groen cirkeltje met daarin C+
- Typ bij Package: fxvb0201 (met kleine letters; dit is de naam van de map waarin de Java-klasse komt)
- Typ bij Name: FXVb0201 (de eerste drie letters zijn hoofdletters; dit is de naam van de Java-klasse)
- Zet een vinkje bij: `public static void main(String[] args)`
- Klik op Finish

Als het goed is, ziet het venster er nu uit als in figuur 2.2a (Ec).

Figuur 2.2a *(Ec)*

In het hoofdvenster van Eclipse zie je de broncode van het bestand FXVb0201.java. Deze broncode is standaardcode die door Eclipse is gegenereerd.

- Verwijder uit de broncode de regel die met `// TODO` begint, en vervang deze door de volgende uitdrukking (neem dit zeer nauwkeurig over):

```
System.out.println("Test geslaagd!");
```

Een project (applicatie) in Eclipse laten uitvoeren betekent dat de broncode opgeslagen en vertaald wordt, en dat de virtuele machine (JVM) wordt gestart om het resultaat van de applicatie te laten zien.
Een applicatie laten uitvoeren kan op verschillende manieren:

- Druk op het toetsenbord op Ctrl F11

of

- Kies in de menubalk Run | Run

of

- Klik met de muis in de menubalk op de knop Run (de 'Play'-knop)

Welke van de drie je kiest, maakt niet veel uit. Er komt de vraag of je het bestand wilt bewaren. Bevestig dat.

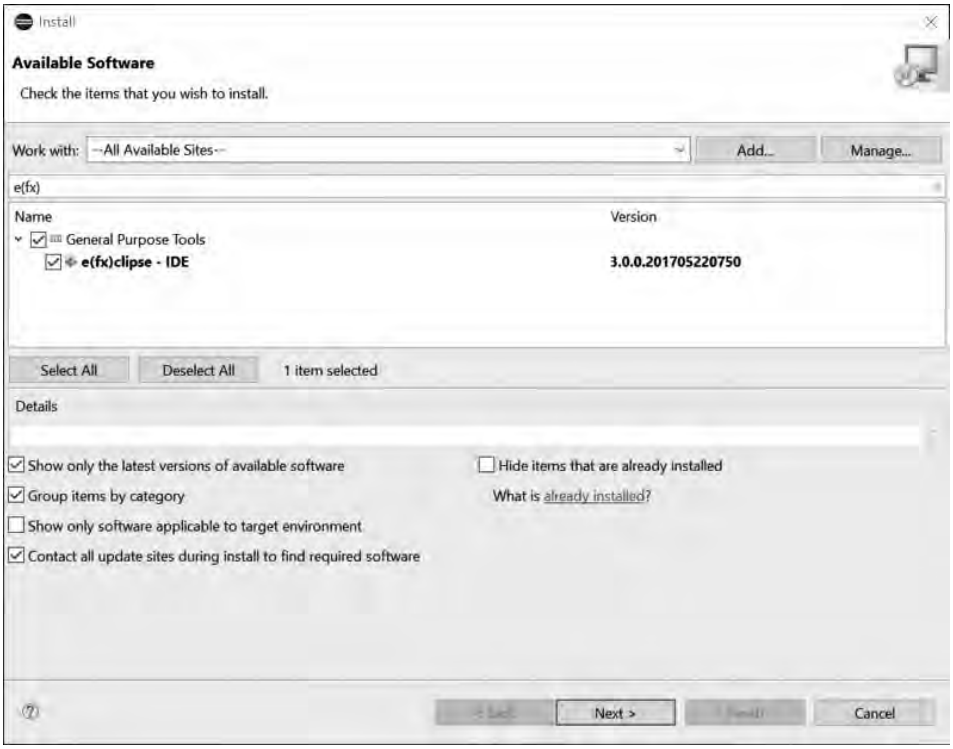
Als het goed is verschijnt onder in beeld, in het zogeheten Console-venster, de tekst "Test geslaagd!".

Oefening 2.3 (Ec)

Wanneer Eclipse vers is geïnstalleerd, is het programma wel geschikt voor het uitvoeren van Java-applicaties, maar nog niet voor JavaFX-applicaties. Om het daar wel geschikt voor te maken, moet je het volgende doen:

- Kies Help | Install New Software…
- Typ in het vak bij 'type filter tekst' de tekst: e(fx)
- Kies bij Work With voor: All Available Sites

Wacht even op de resultaten.
Als het goed is komt er een item met General Purpose Tools, klap dit item uit, en zet een vinkje bij e(fx)clipse – IDE. Zie figuur 2.2b (Ec).



Figuur 2.2b *(Ec)*

- Klik op Next en maak de Wizard verder af.
- Het kan even duren voor alles geïnstalleerd is: onder in beeld zie je 'Installing Software' met een percentage. Wacht tot Eclipse met een popup-venster komt waarin je gevraagd wordt het programma opnieuw op te starten.
- Bevestig in het popup-venster het opnieuw opstarten van Eclipse.

Eclipse is nu klaar voor het toepassen van JavaFX.

Oefening 2.4 (Ec)

Wijzig de broncode uit de vorige oefening zodanig, dat hij exact hetzelfde is als de opstartklasse FXVb0201 in paragraaf 2.2. Typ alles zeer nauwkeurig over (of kopieer de broncode van de website bij dit boek).

Als je klaar bent met de aanpassingen, laat dan het programma uitvoeren (de Play-knop of de toetsen Ctrl F11). Als het goed is verschijnt een leeg venster als in figuur 2.1. Als er geen venster verschijnt, staat er waarschijnlijk een onvolkomenheid in je broncode. Vergelijk deze dan nog een keer goed met die van de opstartklasse in paragraaf 2.2.

Sluit tot slot het venster van de applicatie.
Ga verder met oefening 2.5.

2.3.3 Oefeningen voor beide IDE's

Oefening 2.5

In de broncode van de opstartklasse in oefening 2.4 wordt de afmeting van de scene, en daarmee van het venster van de applicatie, bepaald door deze opdracht:

```
Scene scene = new Scene(root, 300, 250);
```

De scene wordt hiermee 300 pixels breed en 250 pixels hoog.

Verander in de broncode de afmeting van de scene in `600` bij `600`. Laat de code opnieuw vertalen en de applicatie uitvoeren. Sluit tot slot het venster van de applicatie.

Oefening 2.6

Met de volgende opdracht kun je ervoor zorgen dat het programma met een volledig scherm (zonder randen) begint:

```
primaryStage.setFullScreen(true);
```

Voeg deze opdracht aan de opstartklasse toe, vlak boven de drie regels die over de `primaryStage` gaan en laat de code vertalen en uitvoeren.

2.4 Wat is een klasse?

In paragraaf 2.2 staat de broncode van de klasse `FXVb0201`. 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 beeldscherm, 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 zal je steeds beter begrijpen wat objecten zijn, hoe je ze maakt en wat je er mee kunt doen: 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

Aan de slag met Java en JavaFX maakt je wegwijs in de wereld van de objectgeoriënteerde programmeertaal Java. Met de komst van Java 8 is JavaFX een geïntegreerd onderdeel van Java, wat het mogelijk maakt modern uitziende applicaties te maken. Naast het programmeren van applicaties is er aandacht voor de basisprincipes van objectgeoriënteerd ontwerpen en programmeren.

Deze vijfde druk is geheel geactualiseerd. Nieuw is de toevoeging van JavaFX en het gebruik van de geïntegreerde ontwikkelomgevingen NetBeans en Eclipse.

Het boek is bestemd voor beginnende Java-programmeurs, studenten hbo-ict en iedereen die wil weten wat een klasse, object, methode, interface of lambda-functie is. Al deze abstracte concepten worden vanaf het eerste hoofdstuk glashelder uitgelegd. Aan de hand van zo'n 150 goedgekozen voorbeelden worden ze vervolgens gedemonstreerd.

Dankzij de vele vragen en gevarieerde opgaven in het boek en op de website www.aandeslagmetjava.nl (met directe feedback) is *Aan de slag met Java en JavaFX* zeer geschikt om zelfstandig door te werken. Op de website vind je ook de uitgewerkte oefeningen en de broncode van de voorbeelden uit het boek.

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

www.aandeslagmetjava.nl
www.boomhogeronderwijs.nl

9 789024 415663 >